Service Recommendation Based on Targeted Reconstruction of Service Descriptions

Yushi Hao, Yushun Fan*
Tsinghua National Laboratory for Information Science and Technology
Department of Automation
Tsinghua University
Beijing 100084, China
haoys14@mails.tsinghua.edu.cn
fanyus@tsinghua.edu.cn

Wei Tan
IBM Thomas J. Watson Research Center
Yorktown Heights
NY 10598, USA
wtan@us.ibm.com

Jia Zhang
Carnegie Mellon University
Silicon Valley
jia.zhang@sv.cmu.edu

Abstract—**With the rapidly increasing number of services, there is an urgent demand for service recommendation algorithms that help to automatically create mashups. However, most traditional recommendation algorithms rely on the original service descriptions given by service providers. It is detrimental to the recommendation performance because original service descriptions often lack comprehensiveness and pertinence in describing possible application scenarios, let alone the possible language gap existing between service providers and mashup developers. To solve the above issues, a novel method of Targeted Reconstructing Service Descriptions (TRSD) for a specific mashup query is proposed, resorting to the valuable information hidden in mashup descriptions. TRSD aims at introducing mashup descriptions into service descriptions by analyzing the similarity between existing mashups and the specific query, while leveraging service system structure information. Benefit from this approach, missing application scenarios in original service descriptions, query-specific application scenario information, mashup developers' language habits, and service system structure information are all integrated into the reconstructed service descriptions. Based on the reconstructed service description by TRSD, a new service recommendation strategy is developed. Comprehensive experiments on the real-world data set from ProgrammableWeb.com show that the overall MAP of the proposed TRSD model is 6.5% better than the state-of-the-art methods.**

Keywords: service recommendation; mashup creation; service descriptions; mashup descriptions; LDA topic model

I. INTRODUCTION

With the wide adoption of Service-Oriented Architecture (SOA) and Cloud Computing, the quantity and diversity of published web services on the Internet have been rapidly growing [1]. To reuse existing services and shorten software development cycle, mashup has emerged as a popular technique to create value-added composite services by combining multiple individual services as components [2]. However, as the number of services grows rapidly, it has become a significant challenge for inexperienced mashup developers to manually select proper candidates to meet specific functional requirements [3]. This challenge thus calls for new techniques to make recommendation for the mashup creation problem [4].

Most existing service recommendation approaches are semantics-based and function-oriented [5, 6], which analyze and compare features extracted from user queries and service descriptions. However, these approaches mainly rely on the static and raw service descriptions offered by service providers, which have the following three limitations:

1) The lack of comprehensiveness in describing the potential application scenarios: Service providers do hope that their services be used in different application scenarios. However, it is almost impossible for service providers to envision all possible usage scenarios. Original service descriptions thus usually only describe the main functions of the services. For example, in ProgrammableWeb.com[1], *Facebook* is a widely used *social* service, and the most frequent words in its original description are *social*, *network*, and *friend*. It is found to be used in mashup *Pinkbigmac*, which lets users virtually explore destinations in a *travel* scenario. The description of mashup *Pinkbigmac*, however, contains only *Travel*-related words. In this case, since *Facebook* description does not cover its potential usage in a *travel* scenario, common recommendation algorithms will fail to suggest *Facebook* to *Pinkbigmac*. In fact, *Facebook* has been used dozens of times for *Travel*-related application scenarios, although its description does not contain the word *travel*. If an algorithm can introduce the *Travel* application scenario information into *Facebook*'s original description, it can help improve the recommendation accuracy.

2) The weakness of pertinence in describing the particular application scenario of a query: Different from services, each query and each mashup often describes a specific application scenario. When a new query appears, the applicability of each service in the certain scenario described by this query should get more concerned. However, the original service descriptions do not highlight this pertinence. For example, the lengthy description of service *Bing Maps* mentions *birds-eye* only once in the sentence *"...and can summon the birds-eye, 3D, and..."* without a detailed explanation of *birds-eye*. Mashup *BMMTS* actually uses the *birds-eye* function of *Bing Maps* to *"appreciate miniature photography of landscape images."* For a new query aimed at *miniature photography*, it is difficult to design an algorithm to evaluate the applicability of *Bing Maps* in this application scenario. A typical recommendation algorithm may fail to rank *Bing Maps* in the recommendation list, since its description has a large amount of unwanted noise and the introduc-

*Communication Author

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

tion to *birds-eye* is not obvious and adequately. When faced with such a query, *Bing Maps* will be more easier to be recommended if its description can focus on the *birds-eye* function and the *miniature photography* application scenario.

3) The language gap between service providers and mashup developers: In general, service descriptions are written by service providers while queries are proposed by mashup developers. Since service providers and mashup developers are likely to have different language habits and word preferences, they may use different ways and vocabularies to describe the same functional characteristics, which will have a negative effect on the recommendation performance. For example, the description of *Google Maps* uses words *map*, *local*, etc. to introduce its function, while mashup developers often use *location*, *city*, *place*, etc. when describing similar functions. If an algorithm can eliminate this language gap, its recommendation performance will be improved.

To the best of our knowledge, no existing method overcomes all the aforementioned limitations. In this paper, we propose to tackle the issues by reconstructing and refining service descriptions using four major strategies. First, different mashups represent various application scenarios of services. Therefore, mashup descriptions can be used to supplement the missing application scenarios in the original service descriptions. Second, for a specific query, the mashups whose application scenarios are more similar to that of the query will be weighted higher to emphasize the applicability of the service in the certain scenarios. Third, mashup developers' language habits and word preferences are considered, thereby eliminating the language gap between service providers and mashup developers. Last, service usage history is encoded into service descriptions to reflect the structure information of the service system. The added information in the reconstructed service descriptions is mined to improve service recommendation.

The main contributions of this paper are summarized as below:

1) A novel way of refining service descriptions for a specific query, a Targeted Reconstruction of Service Descriptions (TRSD) model, is proposed. TRSD model assigns a reconstructed weight to each mashup and then introduces the mashup descriptions into service descriptions by the weights. Introducing mashup descriptions into service descriptions supplements application scenarios of services, and bridges the language gap between service providers and mashup developers. The reconstructed weight calculated by the TRSD model represents the pertinence of mashups in the particular application scenario described by the query.

2) A novel service recommendation approach is developed based on the TRSD model. Benefiting from the appropriate reconstruction of the service descriptions, the proposed recommendation algorithm is capable of comprehensively utilizing the various types of information in the service system, such as service and mashup descriptions, structure information, service popularity information, etc., which greatly enhance the recommendation performance.

3) Comprehensive experiments on the real-world data set from ProgrammableWeb.com show that the overall MAP of the proposed TRSD model is 6.5% better than the state-of-the-art methods.

The rest of this paper is organized as follows. Section II summarizes the related work. Section III gives definitions to the service system and formulates the recommendation problem. Section IV introduces the framework of the reconstruction process and the recommendation strategy. Section V illustrates the TRSD model and the subsequent service recommendation approach. Section VI reports the experimental results and Section VII concludes the paper.

II. Related Work

In recent years, service recommendation has become one of the core problems in the field of services computing. In general, service recommendation algorithms fall into two categories: functional service recommendation and non-functional service recommendation.

A. Functional Service Recommendation

Functional service recommendation methods focus on meeting the user's functional requirements. Previous works often performed keyword search-based methods on WSDL documents of services [11]. [12] represented user queries by keywords and proposed a user-based collaborative filtering (CF) algorithm. However, these methods suffer from poor performance in practice and detailing WSDL documents is becoming increasingly difficult to obtain [13]. In recent works, LDA model has been widely used to characterize the latent topic features of services and user queries leveraging natural language descriptions instead of WSDL documents [5, 6]. Our previous work [10] proved that taking service evolution information and service popularity information into consideration can improve the recommendation performance. [22] applied collaborative filtering technique on user's interactions and improved recommendation's efficiency. [14] and [15] modelled the evolving patterns of the service system and improved the recommendation performance by considering temporal information. However, existing approaches mainly rely on the original service descriptions offered by service providers, which have many limitations and are harmful to the recommendation performance.

B. Non-functional Service Recommendation

Non-functional service recommendation methods pay more attention to the Quality of Service (QoS) or service network analysis. Neighborhood-based collaborative filtering, which is based on the assumption that similar users tend to consume similar items, has been widely used to predict unknown QoS values [16, 17]. [18] took location information into consideration and improved the recommendation performance. [19] modelled service usage patterns of an evolving service system. [20] improved the performance of service ranking by performing services ranking and clustering mutually in a heterogeneous service network. [21] modelled social network and incorporated the trust relationship in social networks with user feedback for service recommendation, thereby improving the service recommendation performance. However, these methods are difficult to adequately analyze the user's individual needs.

III. Problem Definition

This section will stretch some symbolic definitions to abstractly describe the service system and formulate the service recommendation for the mashup creation problem.

Topology Definition: Service System. In this paper, a 5-tuple $SS = (S, SD, M, MD, R)$ is used to define a service system. $S = \{s_1, s_2, ..., s_{SN}\}$ represents the set of services, where $SN = |S|$ is the number of services in the service system. $SD_i = \{w_{i1}, w_{i2}, ..., w_{in_i}\}$ is the collection of original service descriptions provided by service providers, and w_{ik} stands for the kth word in the ith service description. $M = \{m_1, m_2, ..., m_{MN}\}$ represents the set of mashups, where $MN = |M|$ is the number of existing mashups in the service system. $MD_j = \{w_{j1}, w_{j2}, ..., w_{jn_j}\}$ is the collection of mashup descriptions provided by mashup developers, and w_{jk} stands for the kth word in the jth mashup description. $R = (r_{ij})_{i=1, j=1}^{MN \times SN}$ represents the historical usage records between mashups and services, where $r_{ij} = 1$ when service j is called by mashup i, while $r_{ij} = 0$ if otherwise.

Problem Definition: Service Recommendation for Mashup Creation. In the service system SS, given a new mashup query $q \in Q$ described by $QD_q = \{w_{q1}, w_{q2}, ..., w_{qn_q}\}$ which consists of n_q words, the service recommendation algorithm is aimed to provide the user with a ranked list of services called RL_q. As shown in Figure 1, a mashup query describing specific application scenario from mashup developer is the input of the algorithm, and the ranked service list is the output. In the ranked list, a service with a higher ranking is more likely to meet the user's need for mashup creation thus has a higher probability to be adopted by query q.

IV. Model Framework

This section will make a brief introduction to the whole methodology aiming to solve the aforementioned problem. A novel service recommendation approach is proposed based on a Targeted Reconstruction of Service Descriptions (TRSD) model, whose overall framework is shown in Figure 1 comprising of three steps. When a new query comes, Steps 1 and 2 represent the process of the TRSD model, which includes reconstructed weight vector calculation and service description reconstruction. Step 3 performs the service recommendation process based on the reconstructed service descriptions, and results in a ranked list of services for the user.

In Step 1, the Latent Dirichlet Allocation (LDA) [7] model is applied to calculate the topic features of the existing mashups and the new query using their descriptions, and then the Jensen-Shannon (JS) divergence [8] is leveraged to calculate the similarity of the topic features between the mashups and the query. Typically, when a mashup is similar to a query, its comprising services are likely to be used again in the similar application scenario. Thus in the process of service description reconstruction, giving such a mashup a higher weight can emphasize the applicability of the service in this particular scenario. In this case, the reconstructed weight of each mashup is computed by leveraging the above-mentioned similarity information and some other structure information. The weights of all mashups form a weight vec-

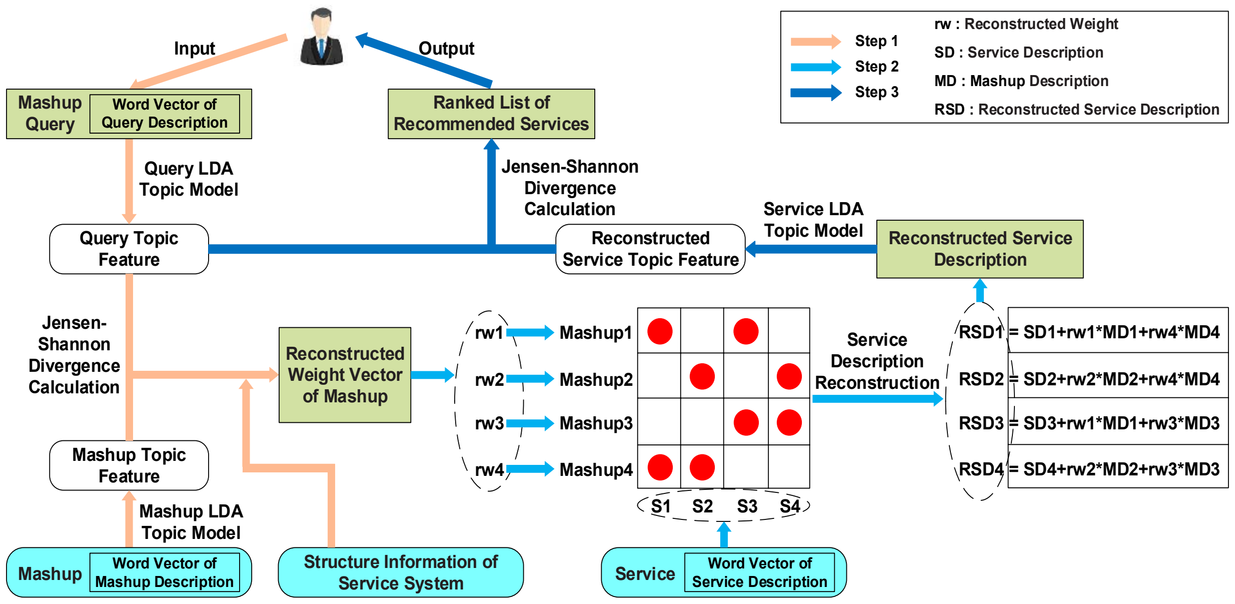
Input
Output
Step 1
Step 2
Step 3
rw : Reconstructed Weight
SD : Service Description
MD : Mashup Description
RSD : Reconstructed Service Description
Mashup Query
Word Vector of Query Description
Ranked List of Recommended Services
Query LDA Topic Model
Jensen-Shannon Divergence Calculation
Service LDA Topic Model
Query Topic Feature
Reconstructed Service Topic Feature
Reconstructed Service Description
Jensen-Shannon Divergence Calculation
Reconstructed Weight Vector of Mashup
rw1
rw2
rw3
rw4
Mashup1
Mashup2
Mashup3
Mashup4
Service Description Reconstruction
RSD1 = SD1+rw1*MD1+rw4*MD4
RSD2 = SD2+rw2*MD2+rw4*MD4
RSD3 = SD3+rw1*MD1+rw3*MD3
RSD4 = SD4+rw2*MD2+rw3*MD3
S1 S2 S3 S4
Mashup Topic Feature
Mashup LDA Topic Model
Mashup
Word Vector of Mashup Description
Structure Information of Service System
Service
Word Vector of Service Description

Figure 1. Framework of Service Recommendation Based on Targeted Reconstruction of Service Descriptions

Steps 1 and 2 represent the process of the TRSD model, which assigns a reconstructed weight to each mashup and then introduces the mashup descriptions into service descriptions by the weights to achieve the reconstruction of service descriptions. Benefiting from the reconstructed service descriptions, Step 3 performs the service recommendation process utilizing the LDA model and the JS divergence calculation and generates a ranked list of services.

tor and it will be used in the subsequent reconstruction process.

In Step 2, service descriptions are reconstructed by introducing mashup descriptions into them, synthetically utilizing the reconstructed weight vector and the service usage history information. For each service, description word vectors of the mashups that use this service are multiplied by the corresponding reconstructed weights and added to the original service description word vector. Since different mashups represent various application scenarios of services, this step can supplement the missing application scenarios in the original service descriptions. Moreover, the reconstructing process introduces mashup developers' language habits into service descriptions, thereby eliminating the language gap.

In Step 3, based on the reconstructed service descriptions, the LDA model and the JS divergence are utilized once again to calculate the similarity between services and the query, considering the service popularity information synthetically. All resulting services are sorted in descending order of similarity scores and the ranked list of services is generated for the user.

V. SERVICE RECOMMENDATION BASED ON TRSD

In this section, we describe the three steps in detail.

A. *Step 1: Calculation of Reconstructed Weight Vector*

One fundamental principle is that a mashup that is more similar to the query's application scenario should be given a higher weight in the reconstruction process. To quantify the application scenarios of mashups and queries, they are viewed as latent topics thus the concepts of LDA can be employed to model mashups and queries. Given a set of mashups $M = \{m_1, m_2, ..., m_{MN}\}$ and their respective descriptions $MD_j = \{w_{j1}, w_{j2}, ..., w_{jn_j}\}$, the generation process of MD_j can be modeled as follows:

1) For each topic $k = 1, 2, ..., T$:
   Draw $\phi_k \sim Dirichlet(\beta)$
2) For each mashup $m_j \in M$:
   a) Draw $\theta_j \sim Dirichlet(\alpha)$
   b) For each $w \in MD_j$
      i. Draw a topic $z \sim Multinomial(\theta_j)$
      ii. Draw a word $w \sim Multinomial(\phi_z)$

where T is the number of topics and ϕ_k is the Multinomial distribution over words specific to topic k and θ_j is the Multinomial distribution over topics specific to mashup m_j. α and β are the prior parameters of Dirichlet distribution for θ_j and ϕ_k, respectively.

The Gibbs sampling [9] is then applied to infer the desired parameters θ_j and ϕ_k. θ_j, named as "Mashup Topic Feature" vector, is a $1 \times T$ vector $(\theta_j^1, \theta_j^2, ..., \theta_j^T)$ for each $m_j \in M$ that represents the distribution of each mashup over topics. Given a new query $q \in Q$, the "Query Topic Feature" vector θ_q can be obtained by utilizing the ϕ_k. The topic feature vector θ satisfies the following constraint:

$$\sum_{t=1}^{T} \theta^t = 1 \qquad \theta^t \in [0,1] \tag{1}$$

Afterwards, the JS divergence is applied to calculate the similarity between mashup topic features and the query topic features, in other words, the similarity of the application scenarios between mashups and the query. The equations for calculating the JS divergence between mashup topic feature θ_j and query topic feature θ_q ($JSD(\theta_j \| \theta_q)$) are as follows:

$$\begin{gathered} KLD(\theta_j \| \theta_q) = \sum_{t=1}^{T} \theta_j^t \log\left(\theta_j^t / \theta_q^t\right) \\ \theta_m = (\theta_j + \theta_q)/2 \\ JSD(\theta_j \| \theta_q) = (KLD(\theta_j \| \theta_m) + KLD(\theta_q \| \theta_m))/2 \end{gathered} \tag{2}$$

After calculating the JS divergences between all the mashup topic features and the query topic features, a $1 \times MN$ vector $JSDM = (jsd_1, jsd_2, ..., jsd_{MN})$ is obtained, where jsd_j represents $JSD(\theta_j \| \theta_q)$ and the lower jsd_j, the higher the similarity exists between mashup topic feature θ_j and query topic feature θ_q.

The more services a mashup contains, the less valuable its description is for each service. Therefore, the following equation is used to calculate the weight of each mashup in the reconstruction process (rw_j):

$$rw_j = \lambda \frac{1}{jsd_j} + (1-\lambda) \frac{1}{\sum_{i=1}^{SN} r_{ji}} \tag{3}$$

where $\sum_{i=1}^{SN} r_{ji}$ represents the number of services that mashup m_j contains and $\lambda \in [0,1]$ is a parameter. According to the above analysis, the higher rw_j, the more valuable the description of mashup m_j, thus the reconstructed weight of m_j should be higher. After calculating the reconstructed weights of all mashups, a $1 \times MN$ vector $RW = (rw_1, rw_2, ..., rw_{MN})$ will be obtained, which is called the reconstructed weight vector.

B. *Step 2: Reconstruction of Service Description*

In this step, mashup descriptions are utilized to reconstruct service descriptions in order to introduce the valuable information contained in mashup descriptions into service descriptions.

For a service $s_i \in S$, according to the previous definition, its historical usage records can be represented by vector $(r_{1i}, r_{2i}, ..., r_{MNi})$. Since only mashups using service s_i should be reconstructed into the service, the following equations are utilized to pre-process the reconstructed weight vector:

$$\begin{gathered} RW'_{s_i} = (rw_1, rw_2, ..., rw_{MN}) \circ (r_{1i}, r_{2i}, ..., r_{MNi}) \\ RW''_{s_i} = RW'_{s_i} (RW'^{\sim}_{s_i} = 0) \end{gathered} \tag{4}$$

where $RW''_{s_i} = (rw_{s_i 1}, rw_{s_i 2}, ..., rw_{s_i N_{s_i}})$ contains only the reconstructed weights of mashups that use service s_i and

$N_{s_i} = \sum_{j=1}^{MN} r_{ji}$ is the number of mashups that use service s_i. Afterwards, the elements in RW''_{s_i} are sorted in descending order as follows:

$$RW_{s_i} = sort(RW''_{s_i}, 'descend') = (rw_{s_i 1'}, rw_{s_i 2'}, ..., rw_{s_i N'_{s_i}}) \quad (5)$$

After obtaining the pretreated reconstructed weight vector RW_{s_i}, the description of service s_i can be reconstructed using the following equation:

$$RSD_i = \frac{\sum_{k=1}^{MN} rw_k}{MN} SD_i + \sum_{j=1}^{\min\{N_{s_i}, N_{\max}\}} rw_{s_i j'} MD_{s_i j'} \quad (6)$$

where $\frac{\sum_{k=1}^{MN} rw_k}{MN}$ is the average of the reconstructed weight vector elements. To avoid over-dilution, the original service description is given an average reconstructed weight. $N_{\max} \in N^+$ is the upper limit of the number of mashups introduced. When $N_{\max} \geq N_{s_i}$, all mashups that use service s_i are used for reconstruction. When $N_{\max} < N_{s_i}$, only $N_{\max}$ mashups with larger reconstructed weights are used for reconstruction. By adjusting the parameter $N_{\max}$, the maximum number of mashups introduced can be controlled in order to achieve the best reconstruction effect. $N_{POP_i} = \min\{N_{s_i}, N_{\max}\}$ is defined to denote the number of mashups actually introduced into service s_i during the reconstruction process. Since the more popular services are likely to be used by more mashups, N_{POP_i} contains the popularity information of service s_i. The word vector RSD_i represents the reconstructed service description of service s_i.

Using this process, all service descriptions will be reconstructed into a collection RSD and complete the TRSD process. The pseudo code is summarized in Algorithm 1.

Algorithm 1. TRSD

```
Input:
1. $SS = (S, SD, M, MD, R)$: The service system
2. $q$: User query
3. $T$: Number of topics in LDA
4. $N_{iter}$: Number of iterations in Gibbs sampling
5. $\alpha$ & $\beta$: Hyper-parameters in LDA
6. $\lambda$: Parameter in Eq. (3)
7. $N_{\max}$: Threshold for the number of mashups introduced
Output:
1. $RSD$: The reconstructed service descriptions
Procedure:
1. $\{\theta, \theta_q\} = GibbsSampling(\alpha, \beta, T, N_{iter}, M, MD, q)$
2. Calculate $JSDM$ by Eq. (2)
3. Calculate $RW$ by Eq. (3)
4. For each service $s_i \in S$
5.     Calculate $RW''_{s_i}$ by Eq. (4)
6.     Sort $RW''_{s_i}$ by Eq. (5) and obtain $RW_{s_i}$
7.     Reconstruct $SD_i$ by Eq. (6) and obtain $RSD_i$
8. End
9. Return the reconstructed service descriptions $RSD$
```

C. Step 3: Service Recommendation

Via the TRSD model, the reconstructed service descriptions RSD will contain a wealth of valuable information to help improve recommendation performance.

Similar to Step 1, the LDA model and the Gibbs sampling are applied to calculate the topic feature vectors of the reconstructed service descriptions and the query description. Then, Eq. (2) is used to calculate the JS divergences between all reconstructed service topic features and the query topic features, and a $1 \times SN$ vector $JSDS = (jsd_1, jsd_2, ..., jsd_{SN})$ is obtained, where jsd_i represents $JSD(\theta_i \| \theta_q)$. The lower jsd_i, the higher the similarity exists between reconstructed service topic feature θ_i and query topic feature θ_q.

As described above, the reconstructed service description contains service popularity information. Thus, N_{POP_i} can be used to enhance the recommendation performance. The equation for calculating the recommendation score for each service s_i utilizing jsd_i and N_{POP_i} is as follows:

$$score_i = \frac{1}{jsd_i} \times \log_{10}(N_{POP_i} + 10) \quad (7)$$

In order to avoid the excess effect of N_{POP_i}, a logarithm is used to smooth it. Since $N_{POP_i} \geq 0$, the above equation satisfies $\log_{10}(N_{POP_i} + 10) \geq 1$. $score_i$ is the recommendation score for service s_i, and higher score means better performance. The recommendation scores for all services are sorted in descending order after being calculated and generate a ranked list of services for user.

The pseudo code of the recommendation method is listed in Algorithm 2.

Algorithm 2. Service Recommendation Based on TRSD

```
Input:
1. $SS = (S, SD, M, MD, R)$: The service system
2. $q$: User query
3. $T_1, T_2$: Number of topics in LDA
4. $N_{iter1}, N_{iter2}$: Number of iterations in Gibbs sampling
5. $\alpha_1, \alpha_2$ & $\beta_1, \beta_2$: Hyper-parameters in LDA
6. $\lambda$: Parameter in Eq. (3)
7. $N_{\max}$: Threshold for the number of mashups introduced
Output:
1. $RL$: Ranked list of services
Procedure:
1. $RSD = TRSD(\alpha_1, \beta_1, T_1, N_{iter1}, SS, q, \lambda, N_{\max})$
2. $\{\theta, \theta_q\} = GibbsSampling(\alpha_2, \beta_2, T_2, N_{iter2}, S, RSD, q)$
3. Calculate $JSDS$ by Eq. (2)
4. Calculate $SCORE$ by Eq. (7)
5. Return $RL = sort(S, SCORE, 'descend')$
```

VI. EXPERIMENTS

The real-world data set crawled from ProgrammableWeb.com is used to evaluate our service recommendation algorithms. Comprehensive experiments are designed to compare the proposed TRSD method with the state-of-the-art methods.

A. Data Set Preparation

To test the algorithms, the data of services and mashups from September 2005 to June 2016 were crawled from ProgrammableWeb.com, which is the largest online repository of web services and mashups. Each service and each mashup contains metadata such as name, tags, and descriptions. These descriptions contain bags of words that describe their functionality and features.

The characteristics of the data set are summarized in Table I. In the experiments, services that have never been used or have been deprecated were removed.

TABLE I. DATA SET ON PROGRAMMABLEWEB.COM

Total # of services	13,269
Total # of available services used in at least one mashup	1,196
Total # of mashups	5,840
Total # of vocabulary	21,891
Average # of services in one mashup	1.94

B. Evaluation Metric

The widely accepted metric, *Mean Average Precision @ top N* (MAP@N), is used to evaluate the performances of the recommendation algorithms, which is defined as follows:

$$MAP@N = \frac{1}{|Q|}\sum_{q\in Q}\frac{1}{N_q}\sum_{s\in S_q}\left(\frac{n(q,s)}{r(q,s)}\cdot I(q,s)\right) \quad (8)$$

where $|Q|$ represents the number of queries, S_q is the set of actually used services in query $q\in Q$ and $N_q = \min\{N,|S_q|\}$. For each service $s\in S_q$, $r(q,s)$ represents the ranking position of s in the recommendation list, and $n(q,s)$ represents the ranking position of s in the list that only contains the services in both S_q and the recommendation list. $I(q,s)=1$ when $r(q,s)\le N$ and $I(q,s)=0$ when otherwise. Besides, the overall MAP means MAP@J.

$MAP@N\in[0,1]$ and the higher MAP@N indicates a better accuracy of the recommendation algorithm.

C. Baseline Methods

Seven baseline methods were chosen to compare with our proposed service recommendation approach.

Baseline Method 1: Service Usage Frequency (SUF)

This method ranks and recommends services only in descending order of service usage frequency as follows:

$$SUF(q,s_i) = \frac{\sum_{j=1}^{MN} r_{ji}}{\sum_{i=1}^{SN}\sum_{j=1}^{MN} r_{ji}} \quad (9)$$

Baseline Method 2: Service-description-based Content Matching (SDCM)

The method proposed in [5] applies the probabilistic model LDA to characterize the latten topics between services and queries. By modeling the generation of service descriptions, this method estimates the topic distribution of services $p(t|s)$ and the word distribution of topics $p(w|t)$ and leverages them to calculate the relevance score of services against user query q as follows:

$$SDCM(q,s_i) = \prod_{w\in QD_q}\sum_{t=1}^{T} p(w|t)p(t|s_i) \quad (10)$$

Baseline Method 3: Mashup-description-based Collaborative Filtering (MDCF)

The basic assumption of this traditional neighborhood-based collaborative filtering method is that similar mashups tend to use the same services. MDCF uses LDA to calculate the topic feature vectors of mashup descriptions and the query description and calculates the relevance score as follows:

$$MDCF(q,s_i) = \frac{\sum_{m_j\in U(N,q)} sim(q,m_j)r_{ji}}{\sum_{m_j\in U(N,q)} sim(q,m_j)} \quad (11)$$

where $U(N,q)$ contains the top N similar mashups with q, and $sim(q,m_j)$ calculates the cosine similarity of the topic feature vectors of query q and mashup m_j.

Baseline Method 4: MDCF*SDCM

The method proposed in [10] utilizes service descriptions and mashup descriptions for service recommendation synthetically. The recommendation rating is defined as follows:

$$MDCF*SDCM(q,s_i) = MDCF(q,s_i)\times SDCM(q,s_i) \quad (12)$$

Baseline Method 5: MDCF*SDCM*SUF

This method extends Baseline Method 4 by taking service popularity information into account [10]. The recommendation rating is defined as follows:

$$MDCF*SDCM*SUF = MDCF\times SDCM\times SUF \quad (13)$$

Baseline Method 6: LDA + Matrix Factorization (MF)

This method first uses LDA to calculate the topic feature vectors of mashup descriptions and the query description. Afterwards, the matrix R is factorized by solving the following optimization problem:

$$\min_{v_i}\left(\sum_j\left(\sum_i\left(r_{ji}-\eta_j^T v_i\right)^2+\lambda\|v_i\|^2\right)\right) \quad (14)$$

The recommendation rating is defined as follows:

$$MF(q,s_i) = \zeta^T v_i \quad (15)$$

where η_j and ζ are the resulting topic feature vectors of LDA.

Baseline Method 7: Service Recommendation Based on Non-targeted Reconstruction of Service Descriptions (NTRSD)

The NTRSD model is a simplified version of our proposed TRSD model, but its reconstructed service descriptions are the same for different queries. This method is used to evaluate whether the targeted reconstruction process for different queries enhances the recommendation performance.

The NTRSD model can be obtained by setting $RW\equiv(1,1,...,1)$ and $N_{\max}=MN$ in the TRSD model.

D. *Experiment Results*

1) *Experiment Settings*

A 20-fold cross validation is adopted to examine the performance of the proposed TRSD model and the baseline methods. Each fold of data takes turns at being the test set, while the remaining 19 folds being the training set. Mashups in training set are used to reconstruct service descriptions. For each mashup appeared in the test set, its descriptions are used as user query and its component services as the ground truth.

The parameters are set as follows. For the TRSD model, $\alpha_1 = 50/T_1$, $\alpha_2 = 50/T_2$, and $\beta_1 = \beta_2 = 0.01$ are empirically set. Next, $T_1 = 30$, $T_2 = 60$, $N_{iter1} = 1000$, $N_{iter2} = 100$, $\lambda = 0.4$, and $N_{\max} = 170$ are set after the pre-adjustment. Baseline Method 7 (NTRSD) can be obtained by setting $RW \equiv (1,1,...,1)$ and $N_{\max} = MN$ in the TRSD model. Other baseline methods are all set to their respective optimal parameters.

2) *Performance Comparison*

Figure 2 illustrates the MAP@N of different recommendation algorithms on different sizes of recommendation list.

SDCM utilizes only the original service descriptions for service recommendation. Since the original service descriptions have many shortcomings, SDCM's recommendation performance is obviously worse than other methods. SUF uses only the service popularity information to realize service recommendation. Because users are willing to use more popular services, the recommendation performance of SUF is better than SDCM, but falls behind other methods. MDCF and MF utilize mashup descriptions and service usage histories to bridge the language gap between mashup developers and service providers, thereby significantly improving the recommendation results. MDCF*SDCM comprehensively utilizes service descriptions and mashup descriptions but treats them separately, and further improves the recommendation performance. MDCF*SDCM*SUF takes service popularity information into consideration and carries out a slightly better performance than MDCF*SDCM.

The NTRSD method utilizes mashup descriptions to reconstruct service descriptions. By this way, missing application scenarios in the original service descriptions, mashup developers' language habits, service system structure information and service popularity information are all introduced into the reconstructed service descriptions, thereby improving the quality of service descriptions greatly. Consequently, NTRSD gets the highest MAP among all the baseline methods. By making the reconstructed service description emphasize the applicability of the service in the particular application scenario described by query, our proposed TRSD promotes the accuracy of service recommendation significantly and outperforms all the state-of-the-art baseline methods.

The detailed performance of the methods tested is summarized in Table II. Comparing the performance of TRSD and NTRSD, it can be inferred that targeted reconstruction which taking the application scenarios of different queries into account brings a promotion of about 5.3%. By comparing the performance of TRSD and MDCF*SDCM*SUF, it can be found that the proposed TRSD model is 6.5% better than the state-of-the-art methods, which benefits from the targeted reconstruction of service descriptions.

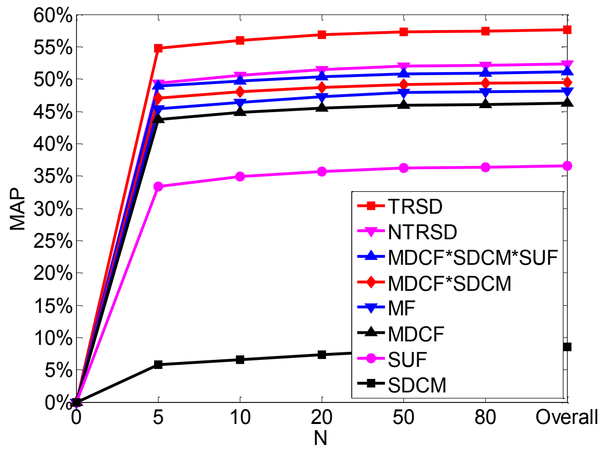

Figure 2. MAP@N of Different Methods

TABLE II. PERFORMANCE COMPARISON OF DIFFERENT METHODS ON MAP@20 AND MAP@J (OVERALL MAP)

Recommendation Method	MAP@20	MAP@J (Overall MAP)
TRSD	**56.80%**	**57.59%**
NTRSD	51.46%	52.29%
MDCF*SDCM*SUF	50.35%	51.13%
MDCF*SDCM	48.73%	49.50%
MF	47.31%	48.15%
MDCF	45.52%	46.30%
SUF	35.71%	36.60%
SDCM	7.30%	8.55%

3) *Effect of* $N_{\max}$

$N_{\max}$ is the threshold for the number of mashups introduced into the service descriptions in the reconstruction process. When $N_{\max}$ is too small, the number of mashups used in the reconstruction process is insufficient, which may make the reconstructed service descriptions miss some important information. When $N_{\max}$ is too large, however, it may introduce too much noise. By tuning the parameter $N_{\max}$, the

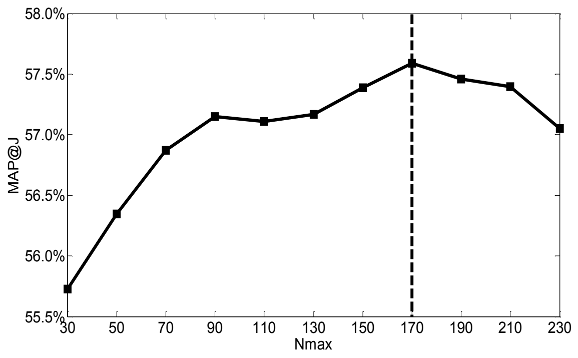

Figure 3. The Effect of $N_{\max}$

maximum number of mashups introduced can be controlled in order to achieve the best reconstruction effect.

Figure 3 shows the overall MAP of TRSD with $N_{\max}$ varied from 30 to 230 with a step value of 20. It shows that the trend of MAP@J is consistent with the previous analysis as $N_{\max}$ increases, the MAP@J first increases and then decreases. Since $N_{\max}$ peaked at 170, $N_{\max} = 170$ is set in the experiments.

VII. Conclusions

Most traditional service recommendation methods depend on the original service descriptions, which have a series of deficiencies that may lower service recommendation performance. In this paper, a novel model named TRSD has been presented, which reconstructs service descriptions for a specific query leveraging service system structure information and mashup descriptions synthetically. The TRSD model shows three features: 1) it complements the missing application scenarios in original service descriptions; 2) it highlights the applicability of services in the exact application scenario described in the query; and 3) it bridges the language gap between service providers and mashup developers.

Based on the TRSD model, a service recommendation algorithm was developed by mining the valuable information contained in the reconstructed service descriptions. Comprehensive experiments on the real-world data set show that the proposed method can gain a 6.5%~11.3% improvement compared with the state-of-the-art methods.

Our future work is to extend the TRSD model to solve the service-side cold-start problem. With the service system evolving and new services published constantly, recommendation over newly released services will become important. We plan to dig the co-occurrence and similarity between pre-existing services and newly released services so that descriptions of new services can be reconstructed using the reconstructed descriptions of pre-existing services, thus making better recommendation for cold-start services.

Acknowledgment

This research has been partially supported by the National Natural Science Foundation of China (No.61673230).

References

[1] V. Andrikopoulos, S. Benbernou, and M.P. Papazoglou, “On the Evolution of Services,” IEEE Transactions on Software Engineering, Vol. 38, 2012, pp. 609-628.

[2] J. Zhang, W. Tan, J. Alexander, I. Foster, and R. Madduri, “Recommend-As-You-Go: A Novel Approach Supporting Services-Oriented Scientific Workflow Reuse,” in Proceedings of IEEE International Conference on Services Computing (SCC), 2011, pp. 48-55.

[3] M. Weiss and G.R. Gangadharan, “Modeling the Mashup Ecosystem: Structure and Growth,” R&D Management, Vol. 40, 2010, pp. 40-49.

[4] W. Gao, L. Chen, J. Wu, and A. Bouguettaya, “Joint Modeling Users, Services, Mashups, and Topics for Service Recommendation,” in Proceedings of IEEE International Conference on Web Services (ICWS), 2016, pp. 260-267.

[5] C. Li, R. Zhang, J. Huai, X. Guo, and H. Sun, “A Probabilistic Approach for Web Service Discovery,” in Proceedings of the IEEE International Conference on Services Computing (SCC), 2013, pp. 49-56.

[6] Y. Zhang, T. Lei, and Y. Wang, “A Service Recommendation Algorithm Based on Modeling of Implicit Demands,” in Proceedings of IEEE International Conference on Web Services (ICWS), 2016, pp. 17-24.

[7] D.M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation,” The Journal of Machine Learning Research, Vol. 3, 2003, pp. 993-1022.

[8] J. Lin, “Divergence Measures Based on the Shannon Entropy,” IEEE Transactions on Information Theory, Vol. 37.1, 1991, pp. 145-151.

[9] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, “Fast Collapsed Gibbs Sampling for Latent Dirichlet Allocation,” in Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 569-577.

[10] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang, “Time-Aware Service Recommendation for Mashup Creation in an Evolving Service Ecosystem,” in Proceedings of IEEE International Conference on Web Services (ICWS), 2014, pp. 25-32.

[11] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, “Similarity Search for Web Services,” in Proceedings of the 13th International Conference on Very Large Data Bases, Vol. 30, 2004, pp. 372-383.

[12] S. Meng, W. Dou, X. Zhang, and J. Chen, “KASR: A Keyword-Aware Service Recommendation Method on MapReduce for Big Data Application,” IEEE Transactions on Parallel & Distributed Systems, Vol. 25(12), 2014, pp. 3221–3231.

[13] C. Ye and H. A. Jacobsen, “Whitening SOA Testing via Event Exposure,” IEEE Transactions on Software Engineering, Vol. 39, 2013, pp. 1444-1465.

[14] Y. Hu, Q. Peng, and X. Hu, “A Time-Aware and Data Sparsity Tolerant Approach for Web Service Recommendation,” in Proceedings of IEEE International Conference on Web Services (ICWS), 2014, pp. 33-40.

[15] Y. Lei, J. Zhou, J. Zhang, F. Wei, and J. Wang, “Time-Aware Semantic Web Service Recommendation,” in Proceedings of IEEE International Conference on Services Computing (SCC), 2015, pp. 664-671.

[16] Z. Zheng, H. Ma, M. R. Lyu, and I. King, “QoS-Aware Web Service Recommendation by Collaborative Filtering,” IEEE Transactions on Services Computing, Vol. 4(2), 2011, pp. 140–152.

[17] Q. Zhang, C. Ding, and C. Chi, “Collaborative Filtering Based Service Ranking Using Invocation Histories,” in Proceedings of IEEE International Conference on Web Services (ICWS), 2011, pp. 195-202.

[18] M. Tang, Y. Jiang, J. Liu, and X. Liu, “Location-Aware Collaborative Filtering for QoS-Based Service Recommendation,” in Proceedings of IEEE International Conference on Web Services (ICWS), 2012, pp. 202-209.

[19] W. Tan, J. Zhang, and I. Foster, “Network Analysis of Scientific Workflows: A Gateway to Reuse,” IEEE Computer, Vol. 43, 2010, pp. 54–61.

[20] Y. Zhou, L. Liu, C. S. Perng, and A. Sailer, “Ranking Services by Service Network Structure and Service Attributes,” in Proceedings of IEEE International Conference on Web Services (ICWS), 2013, pp. 26-33.

[21] S. Deng, L. Huang, and G. Xu, “Social Network-Based Service Recommendation with Trust Enhancement,” Expert Systems with Applications, Vol. 41(18), 2014, pp. 8075–8084.

[22] NN. Chan, W. Gaaloul, and S. Tata, “A Recommender System Based on Historical Usage Data for Web Service Discovery,” Springer-Verlag New York, Inc., Vol. 6(1), 2012, pp. 51–63.