

Article

# Web Service Instant Recommendation for Sustainable Service Mashup

Feng Zhang <sup>1,2</sup> , Benming Chen <sup>1</sup> and Cong Liu <sup>3,\*</sup>

<sup>1</sup> College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China; zhangfeng@sdust.edu.cn (F.Z.); chen9950918@163.com (B.C.)

<sup>2</sup> Shandong Key Laboratory of Wisdom Mine Information Technology, Qingdao 266590, China

<sup>3</sup> School of Computer Science and Technology, Shandong University of Technology, Zibo 255000, China

\* Correspondence: liucongchina@sdut.edu.cn

Received: 16 August 2020; Accepted: 13 October 2020; Published: 16 October 2020



**Abstract:** Service Mashups can help users to integrate data of multiple sources based on Web services composition. Considering a kind of sustainable service Mashup whose data requirement cannot be predetermined, so users need to choose and compose services in a tentative manner. Meanwhile, users can choose and compose services continually to obtain more data based on existing composition results. In such Mashups, a Web service is chosen according to the data provided by the service. Because it is difficult for users to choose from large amounts of services manually, it is a challenge to recommend services instantly for users during the construction of a sustainable service Mashup. This paper proposes an approach to recommend Web services instantly for a sustainable service Mashup. According to the services used in the service Mashup under construction, candidate services are chosen based on the Mashups that are similar to the constructing Mashup, as well as the parameter correlations of services from the perspective of actual operations of Web service composition. Experimental results indicate that the proposed approach has better precision, recall, and coverage values compared to existing state-of-the-art approaches, and therefore, it is more suitable for instant service recommendation of sustainable service Mashups.

**Keywords:** sustainable service Mashup; Mashup similarity; parameter correlation; Web service recommendation

## 1. Introduction

### 1.1. Sustainable Service Mashups

Service-oriented architecture (SOA) is a paradigm where services are used as building blocks for distributed software applications. Web services are the basic elements for implementing SOA, and they have been widely employed in building distributed systems. Moreover, Web services are also widely adopted in cloud computing [1,2], Internet of things [3,4], and big data [5]. Nowadays, there are a large number of available Web services on the Internet, including services based on SOAP [6], services based on REST [7], as well as micro-services [8,9]. In a practical application, individuals and organizations can share and integrate data by composing Web services [10,11]. As a way of Web service composition, service Mashups [12,13] can assist users to integrate data from different data sources on the Internet through composing multiple Web services. There is a kind of service Mashup steered by the user [14], whose key features include: (1) The user requirement cannot be decided in advance; (2) the required services and their composition logics cannot be predefined; (3) the users need to choose and compose Web services manually according to his/her instant requirement; (4) the users can continuously choose and compose other services to obtain more data after getting some required

data based on the constructing service Mashup. Based on the concept of the exploratory service composition [15,16], we call this kind of service Mashups as sustainable service Mashups because a user can compose the services in a sustainable way. During the construction of a sustainable service Mashup, a user has to select from hundreds of Web services, which is extremely difficult and time-consuming. Service recommendation is an effective technique to assist a user to select services. The efficiency of service Mashups can be improved if subsequent required services can be recommended to the user instantly according to the existing Mashup fragment. Therefore, instant service recommendation is needed in the building process of a sustainable service Mashup.

Web Service recommendation is widely studied, and it is one of the hotspots in the service computing field. Personalized information (preference [17], location [18], etc.) of users, time [19], and QoS (Quality of Service) [20] can be used as the basis for Web service recommendation. However, for a sustainable service Mashup, the goal is to obtain required data. As a result, service recommendation in a sustainable service Mashup is irrelevant to a user's preference, location, time, and QoS. Furthermore, only services in the constructing Mashup can be regarded as the basis for recommendation because the requirement of a sustainable Mashup cannot be expressed accurately. These features bring challenges to the service recommendation of sustainable service Mashups.

## 1.2. Related Work

In this section, we introduce two types of approaches most relevant with the proposed approach in this paper. These approaches are based on service composition history (Mashups or service compositions) and the correlations of service parameters.

### 1.2.1. Composition History-Based Service Recommendation

This type of work first builds relations or networks among services according to their co-existence relations extracted from existing service compositions. Then, services are recommended based on their relations or networks [21]. For example, Huang et al. [22] build the network between the composite services and the individual services, and use the link prediction algorithm to recommend services. Similarly, Ni et al. [23] utilize the co-existence and negative-connection to recommend services using existing service compositions.

Some work uses the implicit service composition knowledge in the composition library to recommend services for a user. This type of work searches the similar composition patterns based on existing composition library [24]. For example, MashupAdvisor [25] recommends the possible building results of a Mashup. It uses the Mashup library to estimate the popularity of some outputs (a service or a data source) and rank these outputs. Thus, multiple outputs are recommended to the user based on the structure of the constructing Mashup. Abiteboul et al. [26] argue that different Mashups generally share some common features, such as similar Mashup components. Therefore, Mashup components and their linkages that can complete the constructing Mashup are recommended automatically based on the similarity.

The above work uses existing Mashups or compositions library to recommend Web services by statistics or composition patterns matching. However, if existing Mashups or service compositions cannot reach a certain scale, these approaches cannot work effectively. Moreover, those less frequently used and newly added services cannot be recommended in this case, making the coverage of these approaches very limited.

### 1.2.2. Parameter Correlation-Based Service Recommendation

Some work draws on the idea of Web hyperlinks to describe parameter correlation among Web services. Several service hyperlink models are introduced for different types of services, and services are recommended directly through the service hyperlinks. Han et al. [16] introduce the service hyperlink model for the business building block that is a kind of resource abstraction of the underlying business action. A correlation among the building blocks is presented by the mapping between the former

block's output parameter and the latter one's input parameter. Similarly, Yan et al. [27] present the service hyperlink that are used to describe the binary action constraint relation among business services. Zhao et al. [28] propose the HyperService that contains four types of service relations among the OpenAPI, REST, and SOAP services. Similarly, the link data relation in HyperService presents the mapping between the source service's output parameters and the target service's input parameters. In addition, for data services [29], Liu et al. [30] and Zhu et al. [31] propose the data service hyperlink model that is introduced to describe data links among data services.

The above-mentioned service hyperlink models mainly focus on the composability of Web services, i.e., the invoking relation based on the mapping between the source service's output parameters and the target service's input parameters. However, for services that are used in Mashups, their composition ways contain not only the way based on the invoking relation, but also other ways, such as the aggregation of the output of two services. Furthermore, the emphasis of the above-mentioned work is the service hyperlink model and the modeling techniques. However, the service recommendation is simply implemented by selecting services directly linked by hyperlinks, and the application of service hyperlink on recommendation is not fully addressed.

The above two types of work are introduced comprehensively in Table 1.

**Table 1.** The existing work.

Reference	Recommendation Basis	Approach	Problem
[21,22]		The network among the composite and individual services is built by the existing compositions, which is used to recommend services.	
[23]		The co-existence and negative-connection of services are mined by existing compositions, which are used to recommend services.	(1) If existing Mashups or compositions cannot reach a certain scale, they do not work effectively. (2) The less frequently used and newly added services cannot be recommended.
[24]	Composition History	Similar composition patterns are mined by existing compositions to recommend services.	
[25]		The popularity of services is ranked to recommend services by the structure of the constructing Mashup.	
[26]		Services and their linkages that can complete the constructing Mashup are recommended based on the similarity of Mashup components.	
[16,27,28,30,31]	Parameter Correlations	A correlation of two services is built based on the mapping between the former's output parameter and the latter's input parameter.	(1) The aggregation based on the outputs of two services is ignored. (2) Services are recommended simply by selecting services directly linked by correlations.

### 1.3. Contributions

In this paper, we propose an approach to recommend Web Services instantly for a sustainable service Mashup. Candidate services are chosen from two aspects in this paper. First, the idea of collaborative filtering recommendation based on users [32] is applied by assuming that similar service Mashups use similar Web services. Therefore, we can obtain candidate services from Mashups that are similar with the constructing one through services that are used. Second, by the observation of existing service Mashups, services in a Mashup generally have correlations in their input or output parameters. These correlations are the foundation for Web service composition [33]. Therefore, services that have parameter correlations with those in the constructing Mashup can also be regarded as candidates.

Based on the above analysis, we propose to recommend  $N$  services instantly to the user in the constructing process of a sustainable service Mashup as follows. First, some Mashups that are similar with the constructing Mashup are selected based on services used in the constructing one. Thus, services in these Mashups are chosen as candidates. Second, if the number of obtained candidate services is less than  $N$ , other candidate services are selected by a Web service network that is built on all services and their correlations among their input and output parameters. Finally, recommended services can be instantly decided for a sustainable service Mashup by combining these two parts of the services.

The main contributions of this paper are three-fold. (1) An instant Web service recommendation approach that combines Mashup similarity and service parameter correlations is proposed for sustainable service Mashups. (2) A service recommendation approach based on similar Mashups is proposed by taking both the used same services and their used ways into account to compute the similarity between two service Mashups. (3) An approach for selecting services based on services and their parameter correlations are proposed. Learning from the Random Surfer Model [34] based PersonalRank algorithm [35], a service network is built based on all the services and correlations among their input and output parameters, and the algorithm for sorting services using this service network is given.

The rest of the paper is structured as follows. A motivating scenario is first introduced in Section 2 to illustrate the features of a sustainable service Mashup. The overall framework of the proposed approach is introduced in Section 3. Then, Section 4 presents the recommendation approach based on similar Mashups, and Section 5 presents the recommendation approach based on service parameter correlations. Next, in Section 6, the instant service recommendation approach that combines the proposed two approaches for a sustainable service Mashup is presented. Section 7 introduces experimental results. Finally, Section 8 concludes the paper.

## 2. A Motivating Scenario

This section illustrates the features of a sustainable Mashup and the details in the actual construction through an example of investigating a criminal case [13], in which the policemen need to obtain the data related to the case from different business departments when they investigate a criminal case. If each business department provides data access interfaces using Web services, the policemen can implement data integration through composing available Web services. Thus, they can obtain the required data. Figure 1 shows the process of the investigation.

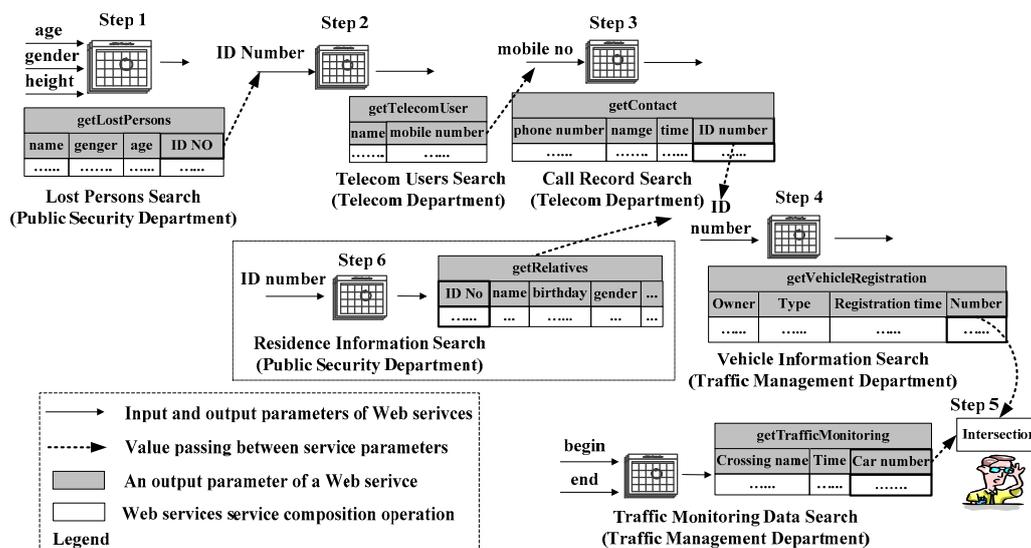


Figure 1. An example of the sustainable service Mashup.

For example, there is a murder case somewhere. The time range of the crime and the height, gender, and approximate age of the victim can be determined through on-the-spot investigation. At the same time, there are tire marks around the scene and surveillance cameras around the junction. So, how to get the potential suspects? How to get relevant clues from a large number of available data sources? The policemen first try to determine the victim's identity by the Web service `getLostPersons`, provided by the public security department according to the obtained height, gender, and age of the victim. Assuming that the identity of the victim is obtained in the example, which services should be used to obtain the required data in the next step? The policemen try to use the victim's recent telephone contacts as candidate suspects. According to the obtained ID number of the victim, they use the service `getTelecomUser` provided by the telecom department to obtain the names and mobile phone numbers of the victim's recent contact persons in the 2nd step. Next, how to determine whether these persons are suspects? The policemen try to obtain the vehicles registered under the names of these persons, and then judge whether these vehicles appear at the designated junction within the time period of the murder. If so, they may be suspects. Therefore, the ID numbers of these contact persons can be obtained through the service `getContact` in the 3rd step, and the license plate numbers owned by them can be obtained through the service `getVehicleRegistration` in the 4th step. Then, the license plates of all vehicles that pass through the scene junction during the time period of the murder are obtained by the service `getTrafficMonitoring`. Through the service composition operation `Intersection`, the license plates are compared with those of the vehicles owned by the recent contacts obtained previously in the 4th step. If there are outputs, the persons in the output can be determined as the candidate suspects. If there are no outputs, the policemen have to choose other Web services to get new clues. On the basis of the obtained data, the policemen may try to take the victim's relatives as candidate suspects and obtain the information of the victim's relatives through the service `getRelatives` in the 6th step. Next, they can obtain the license plates of their vehicles and reuse the `Intersection` operation. Finally, they can decide whether candidate suspects are contained in the comparison results.

In this example, the data requirement of the policemen cannot be accurately expressed before the service Mashup is constructed. Furthermore, the data requirement may change based on obtained data during the construction process. From the perspective of the overall process of a service Mashup, it is impossible to determine exactly what services are required and how to compose these services in advance. The policemen need to select and compose services manually, determine the subsequent services according to the composition results. Based on the obtained composition results, they can continue to select services for composition and obtain more data continuously. Therefore, this is a typical sustainable service Mashup. Instant service recommendation can help the policemen to select services at each step efficiently. Thus, the overall efficiency of building a service Mashup is improved.

### 3. The Overall Framework of the Proposed Approach

From the example in Figure 1, we can see that the services that have been selected in the constructing process of a sustainable service Mashup imply the requirement of a user. Therefore, the services that may be used next step can be recommended according to the services currently used by the user. So, how to discover these services according to the services already used? Through the analysis of existing service Mashups, it can be seen that service Mashups that meet the similar requirements of users may use similar services. Therefore, in the construction process of a sustainable service Mashup, existing Mashups that are similar to the currently constructed Mashup can be queried using the used services, and services that are used in these similar Mashups can be used as recommended services. Moreover, from the perspective of the actual operation of composing Web services, the composition strategy of services should be determined according to the input and output parameters of the chosen services. For example, in Figure 1, the values of the attribute `ID NO` in the output of the service `getLostPersons` can be used as the input of the service `getTelecomUser`. Therefore, the correlations between the input and output parameters of Web services are also a basis for determining candidate services.

The effectiveness of services recommendation using similar Mashups is affected by the number of existing Mashups. Therefore, when recommended services cannot be obtained through existing similar Mashups, the services can be further selected through the parameter correlations of the used services. In this paper, service recommendation based on similar Mashups is combined with the recommendation based on parameter correlations of Web services to provide instant service recommendation for sustainable service Mashups. Using this approach, the following  $N$  services can be recommended to the user according to the used services. Specifically, assume that  $N$  Web services need to be recommended, and  $M$  services are chosen by similar Mashups first. If  $M$  is less than  $N$ , other  $N-M$  services are chosen based on the parameter correlations of the used services.

#### 4. Similar Mashups-Based Service Recommendation

For the similarity measure between the constructing Mashup and an existing Mashup, a fundamental hypothesis is that, the more the same services are contained in these two Mashups, the more similar they are. Moreover, the way that the same services are used should also be taken into account when computing their similarity. Specifically, first, during the construction of a service Mashup, a user needs to assign values to some input parameters of the used services. Second, after a service is invoked, the user may need only a part of its output. For example, the user may delete some parameters of the output to retain what he/she needs. Therefore, both input parameters that are assigned values by a user and output parameters that are retained in a service Mashup imply the data requirement of this Mashup more accurately.

Take the two service Mashups in Figure 2 for example. These two Mashups contain the service `getOwnerInfo` that receives the plate number of a car and outputs the information of the car's owner. In the first Mashup, the user assigns values to the input parameters crossing number, monitoring start time, and monitoring end time. Then, the user gets the output describing the cars passing X street between the start and end time. After editing the output, the user only retains the car's license plate number. Next, the user takes the obtained plate number as the input of the service `getOwnerInfo` and invokes this service. Finally, the user only retains the owner name and ID number of the output. In this Mashup, the user aims to get the information of the owners whose cars pass the X street in the specified period of time. In the second Mashup, the user first assigns the value for the input parameter plate number for the service `getOwnerInfo` and gets the data about the car with the number XX567. The user only retains the attribute type of the output. Then, the user uses the obtained type value as the input and invokes the service `getAutoInfo`. Finally, the user gets the manufacturer, setting, and price of the output. In this Mashup, the user aims to search for the specific information of a car through its plate number. Both Mashups use the service `getOwnerInfo`. If their similarity is computed only by the same services using the Jaccard formula, their similarity is  $1/3$ . However, from the actual requirements of these two Mashups, their similarity is much lower than  $1/3$ . This is because the same service is used in completely different ways. For the service `getOwnerInfo`, the retained parameters of the output are different in two service Mashups, which reflects the difference of the user's required data. Meanwhile, the input parameters assigned by the user are also different, which indicates the different initial states of these two Mashups. Therefore, if two service Mashups contain the same services, input parameters assigned and output parameters retained by the user should all be taken into account for their similarity measure.

Let  $s$  be a Web service contained in service Mashups  $u$  and  $v$ . The input and output parameters of  $s$  are denoted as  $s.input = (i_1, i_2 \dots i_n)$  and  $s.output = (o_1, o_2 \dots o_m)$ , respectively. Then,  $s.io = (i_1, i_2 \dots i_n, o_1, o_2 \dots o_m)$  is an  $n + m$  dimension vector. In addition,  $s_u.io$  and  $s_v.io$  are both  $n + m$  dimension vectors that describe the input parameters assigned by the user and output parameters retained in the final result, respectively. For  $s_u.io$ , let its  $x$ th ( $1 \leq x \leq n$ ) input parameter be  $i_x$ , and the value of the dimension that  $i_x$  stands for is: If  $s.i_x$  in  $u$  is assigned by the user,  $s_u.io[i_x] = 1$ , otherwise  $s_u.io[i_x] = 0$ . Similarly, for  $s_u.io$ , the value of the dimension that the  $y$ th ( $1 \leq y \leq m$ ) output parameter  $o_y$  stands for is: If  $s.o_y$  in  $u$  is retained by the user,  $s_u.io[o_y] = 1$ , otherwise  $s_u.io[o_y] = 0$ . Let  $Sim_{u,v}(s)$  be the similarity

that  $s$  is used in  $u$  and  $v$ . Based on the way that  $s$  is used in  $u$  and  $v$ ,  $Sim_{u,v}(s)$  can be computed by the cosine similarity in Equation (1). We denote the set of services contained in  $u$  and  $v$  as  $N(u)$  and  $N(v)$ , respectively. Then, the similarity between  $u$  and  $v$  that is denoted as  $Similarity(u, v)$  can be measured by Equation (2).

$$Sim_{u,v}(s) = \frac{s_u \cdot io \cdot s_v \cdot io}{|s_u \cdot io| \times |s_v \cdot io|} \tag{1}$$

$$Similarity(u, v) = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)| \times |N(v)|}} \times \frac{\sum_{s \in N(u) \cap N(v)} Sim_{u,v}(s)}{|N(u) \cap N(v)|} \tag{2}$$

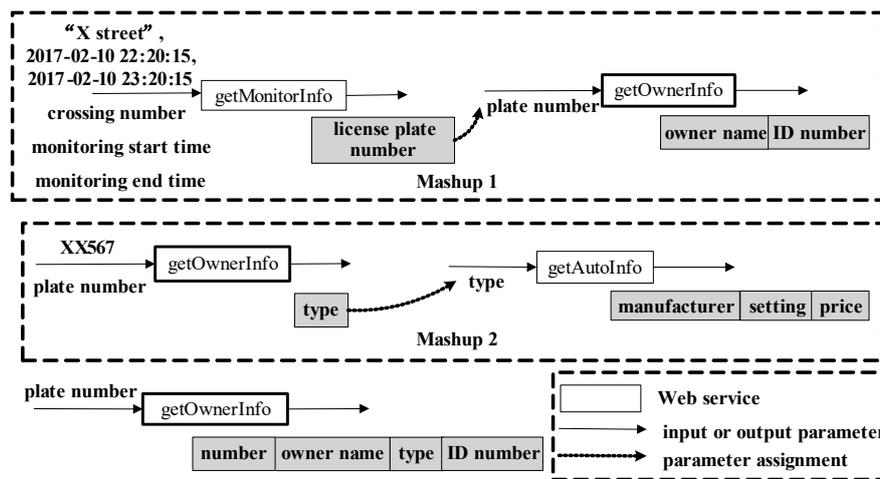


Figure 2. Examples of service Mashups.

The left part of Equation (2) is used to compute the similarity of  $u$  and  $v$  through the same services in them, and the right part is used to compute the average similarity of all the same services. After the same part of the numerator and denominator is divided out, the final formula for measuring  $Similarity(u, v)$  is shown as Equation (3).

$$Similarity(u, v) = \frac{\sum_{s \in N(u) \cap N(v)} Sim_{u,v}(s)}{\sqrt{|N(u)| \times |N(v)|}} \tag{3}$$

Go back to the example in Figure 2. The first and second Mashup are noted as  $u$  and  $v$ , respectively. For the service `getOwnerInfo`, its input parameter is assigned by the output of the former service in  $u$ . While in  $v$ , this input parameter is assigned by the user. Therefore, there is no same input parameter assigned by the user in  $u$  and  $v$ . Meanwhile, the output parameters of `getOwnerInfo` retained in  $u$  and  $v$  are different. By Equation (1),  $Sim_{u,v}(getOwnerInfo) = 0$ . Therefore,  $Similarity(u, v) = 0$  according to Equation (3). The reason for this result is that the way that `getOwnerInfo` is used in  $u$  and  $v$  is completely different, which is in accordance with the actual situation of the example in Figure 2.

Through the similarity of the constructing Mashup and existing Mashups, each candidate service can be assigned a score called a Mashup relevancy. Through Equation (3), we can obtain a group of Mashups, in which each Mashup's similarity with the constructing one is higher than a threshold  $\lambda$ . Then, for each service that is used in these Mashups, but not in the constructing one, we compute its total Mashup relevancy. Let the constructing Mashup be  $c$ , and services in  $c$  be  $S_c$ . The Mashups whose similarity with  $c$  are higher than  $\lambda$  is  $c_1, \dots, c_i, \dots, c_n (1 \leq i \leq n)$ , and the similarity between  $c_i$  and  $c$  is  $Similarity(c_i, c)$ . For  $c_i$ , the services in  $c_i$  are denoted as  $S_{c_i}$ . For any service  $s \in S_{c_i} - S_c$ , the Mashup relevancy obtained through  $c_i$  is  $MRel_i(s) = Similarity(c_i, c)$ . Since  $s$  may be used in multiple Mashups,

the Mashup relevancy of  $s$  (denoted as  $MRel(s)$ ) that is obtained through  $c_1, c_2, \dots, c_i \dots c_n$ , can be measured by Equation (4).

$$MRel(s) = \sum_{i=1}^n MRel_i(s) (s \in S_{c_i} - S_c) \quad (4)$$

The algorithm `getWSFromMashups` ( $m, M, \lambda, N$ ) to recommend  $N$  services for the constructing Mashup  $m$  is shown in Algorithm 1. In this algorithm, the function `getSimMashups` ( $m, \lambda$ ) is used to obtain existing Mashups whose similarity with  $m$  is higher than  $\lambda$ . For a candidate service  $cs$ , the function `getMRel` ( $cs, mu$ ) returns the Mashup relevancy of  $cs$  in a Mashup  $mu$ . The function `add` ( $R, cs, MRel(cs)$ ) is used to add  $cs$  and its relevancy into  $R$ . If  $R$  contains  $cs$ , its Mashup relevancy is added with the existing value in  $R$ . Finally, the function `getTop` ( $R, N$ ) returns  $N$  services that own the top Mashup relevancy in  $R$ . If the number of services in  $R$  is less than  $N$ ,  $R$  is returned directly.

---

**Algorithm 1** `getWSFromMashups`( $m, M, \lambda, N$ )

---

**Input:** The constructing Mashup ( $m$ ), services in  $m$  ( $m.S$ ), the set of existing Mashups ( $M$ ), Mashup similarity threshold ( $\lambda$ ), the number of recommended services ( $N$ )

**Output:** The set of services recommended ( $R$ )

1. **R = candiWSs =  $\Phi$**
  2. `simMashups = getSimMashups(m,  $\lambda$ )`//obtain Mashups whose similarity with  $m$  is higher than  $\lambda$
  3. **for** each  $mu$  in `simMashups`
  4. `candiWSs = mu.S - m.S`
  5. **for** each  $cs$  in `candiWSs`
  6. `MRel(cs) = getMRel(cs, mu)`//get the Mashup relevancy of  $cs$  in the Mashup  $mu$
  7. `add (R, cs, MRel(cs))`
  8. **Endfor**
  9. **Endfor**
  10. `return getTop(R, N)`//get  $N$  services that own the top Mashup relevancy in  $R$
- 

## 5. Web Service Parameter Correlation-Based Service Recommendation

### 5.1. Service Parameter Correlations and Service Hyperlink

While constructing a sustainable service Mashup, services selected by a user in the next step typically have parameter correlations with those already used services in the current Mashup. Therefore, services having parameter correlations with those already used can also be regarded as candidates. The parameter correlations of services in a service Mashup contain not only the correlation between the former service's output parameter and the latter's input parameter, but also the correlation between the input or output parameter of two services. According to the work [36,37], if the former and latter services are denoted as  $s$  and  $t$ , respectively, there are three types of parameter correlations: (1) OI correlation (denoted as  $pc_{OI}$ ), indicates that the value of an output parameter of  $s$  can be assigned to an input parameter of  $t$ . (2) II correlation (denoted as  $pc_{II}$ ), indicates that an input parameter of  $s$  and  $t$  can receive the same value as the input. (3) OO correlation (denoted as  $pc_{OO}$ ), indicates that an output parameter of  $s$  and  $t$  are semantically equivalent concepts. Figure 3 shows some examples. There is an OI correlation between the service 1 and 2, an II correlation between the service 1 and 3, and an OO correlation between the service 3 and 4. From these examples, we can see that the OI correlation is unidirectional, while the II and OO correlation are bidirectional [36]. Using the concept of service hyperlinks [13], we describe the relation between two services in terms of parameter correlations of Web services.

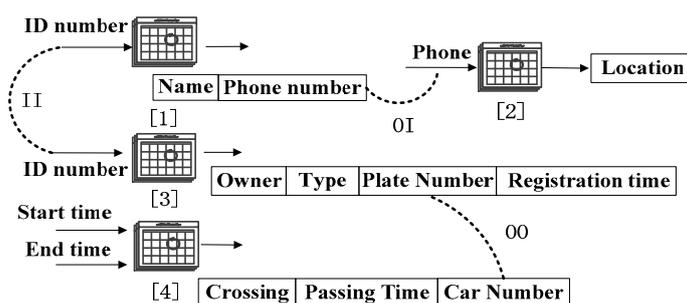


Figure 3. Examples of the service parameter links.

**Definition 1. Service HyperLink (SH).**  $SH = \langle LinkID, Source, Target, PCS \rangle$ , where *LinkID* is the id of the service hyperlink, *Source* is the source service, *Target* is the target service,  $PCS = \{pc_1, pc_2, pc_n\}$  is the parameter correlations in *SH*, and  $type(pc_i) = pc_{OI} | pc_{II} | pc_{OO}$ .

In this paper, if an *SH*'s source service is *s*, and the target service is *t*, *SH* is denoted as  $s \rightarrow t$ . A service hyperlink may contain the above three types of parameter correlations between the source and the target service. Through the parameter correlations between *s* and *t*, the hyperlink between them can be obtained as follows. (1) If there are one or more *II* or *OO* parameter correlations between *s* and *t*, there exist two hyperlinks containing all the correlations, which are denoted as  $s \rightarrow t$  and  $t \rightarrow s$ , respectively. (2) If there are one or more *OI* parameter correlations between *s* and *t*, there exists a hyperlink containing all these correlations, which are denoted as  $s \rightarrow t$ . (3) If a hyperlink can be obtained according to more than one correlation, one hyperlink is retained, and all these links are merged into this hyperlink [36].

The key for obtaining service hyperlinks is to obtain parameter correlations among services. The links can be obtained manually by the service providers, or set up automatically by a program. In addition, correlations can be obtained through extracting from existing service compositions or Mashups [33]. Limited by space, the details are not presented in this paper.

If we take a service as a vertex, and a hyperlink as a directed edge among two services, we can build a service network.

**Definition 2. Service Network (SN).**  $SN = \langle S, LINKS \rangle$ , where  $S = \{s_1, s_2 \dots s_n\}$  is the services in *SN*, and  $LINKS = \{link_1, link_2 \dots link_m\}$  is the service hyperlinks among the services in *SN*.

### 5.2. Service Network-Based Service Recommendation

While constructing a sustainable service Mashup, services that have parameters correlations with the used services can be directly obtained by *SN*. The services, whose distance with used services in *SN* is one, i.e., services that have hyperlinks with used services, can be regarded as the candidates. If the number of directly linked services is too small, we can look for indirectly linked services in *SN* using the Width-First Traversal and select from services whose distances with the used services in the constructing Mashup are two, three, and so on.

However, the key issue is how to select from multiple services according to *SN*. First, some services have too many directly linked services by service hyperlinks. For example, if a service has an attribute identity number in its output, it may have *OI* parameter correlations with multiple services. Another example, if a service has an input parameter that can be assigned to multi-parameter values, such as keyword, it may have *II* parameter correlations with many other services. In this case, we need to select services from the linked services. Second, if *N* services need to be recommended to a user, but the number of services that are directly correlated is smaller than *N*, we need to select from services whose distance with those used is two. Similarly, if the number of obtained services is still smaller than *N*, we need to select from services whose distance with the used services is three, four, and so on. To sum up, the main challenge is how to select from services in the same layer of *SN*.

To solve this problem, we propose an approach for measuring the correlation degree between a candidate service and a used service based on the structure of  $SN$ . In the Web hyperlink analysis, the random walk model [34] is used to compute the probability that one Web page walks to another Web page. Similarly, the correlation degree of a service with another service can be obtained using the probability that the former service walks to the latter one along service hyperlinks in  $SN$ .

Based on the graph structure, Haveliwala [35] proposes a recommendation algorithm to get the probability that one vertex randomly walks to another. Using this algorithm, we can regard the probability that the service  $u$  walks to  $v$  randomly in  $SN$  as the correlation degree of  $v$  and  $u$ , which is denoted as  $Rel(v, u)$ .  $Rel(v, u)$  can be computed by Equation (5), in which  $d$  is the probability that a user selects the next service along a service hyperlink, and  $out(v')$  is the service set that hyperlinks of  $v'$  point to. Similar to [24],  $d$  is set to 0.85.

$$Rel(v, u) = \begin{cases} d \sum_{v' \in in(v)} \frac{Rel(v', u)}{|out(v')|} & (v \neq u) \\ 1 - d + d \sum_{v' \in in(v)} \frac{Rel(v', u)}{|out(v')|} & (v = u) \end{cases} \quad (5)$$

Correlation degree values among services are computed iteratively based on  $SN$  until the value of each pair of services is converged. Since this algorithm has a higher time complexity [35], it can be executed offline. In addition, if  $SN$  changes, the correlation degree value of each pair of services needs to be updated regularly.

After all correlation degrees between each pair of services are obtained, the average correlation degree between a candidate service and those in the constructing service Mashup can be calculated. If the constructing Mashup is denoted as  $c$ , and services used in  $c$  is  $s_1, s_2, \dots, s_n$ , the average correlation degree  $Rel(s, c)$  between candidate service  $s$  and  $c$  can be measured by Equation (6).

$$Rel(s, c) = \frac{\sum_{i=1}^n Rel(s, s_i)}{n} \quad (6)$$

The algorithm `getWSFromWSN` ( $m, SN, N$ ) to recommend  $N$  services for a constructing Mashup based on  $SN$  is shown in Algorithm 2. In this algorithm, `getWSFromSH` ( $m.S, SN, distance$ ) returns services whose distances with services in  $m.S$  are equal to the value of `distance`. `getTopRelatives` ( $m.S, SN, N$ ) is used to select  $N$  services according to the average correlation degrees with services in  $m.S$ .

---

**Algorithm 2** `getWSFromWSN` ( $m, SN, N$ )

---

**Input:** The constructing Mashup ( $m$ ), services in  $m$  ( $m.S$ ), the service network ( $SN$ ), the number of recommended services ( $N$ )

**Output:** The set of services recommended ( $R$ )

1. **R** =  $\Phi$
  2. **for** (`distance` = 1;  $N > 0$ ; `distance`++)
  3. `candiServices` = `getWSFromSH`( $m.S, SN, distance$ )//returns services whose distances with services in  $m.S$  are equal to the value of `distance`
  4. **If** `candiServices` = `NULL`
  5. **return**  $R$
  6. **Else**
  7. **If** (`size`(`candiServices`)  $\leq N$ )
  8. **add** ( $R, candiServices$ )
  9.  $N = N - size(candiServices)$
  10. **Else**
  11. **add** ( $R, getTopRelatives(m.S, SN, N)$ )
  12. **return**  $R$
  13. **EndIf**
  14. **EndIf**
  15. **Endfor**
-

## 6. Instant Service Recommendation for Sustainable Service Mashup

According to the overall framework of the proposed approach in Section 3, Algorithm 3 presents the complete process to recommend  $N$  services for a constructing service Mashup using the above two approaches.

---

**Algorithm 3** recommendWS ( $m, M, SN, \lambda, N$ )

---

**Input:** The constructing Mashup ( $m$ ), the set of existing Mashups ( $M$ ), the service network ( $SN$ ), Mashup similarity threshold ( $\lambda$ ), the number of recommended services ( $N$ )

**Output:** The set of  $N$  services recommended ( $R$ )

1.  $R = \text{getWSFromMashups}(m, M, \lambda, N)$
  2. **If** ( $\text{size}(R) = N$ )
  3.     return  $R$
  4. **Else**
  5.      $n = N - \text{size}(R)$
  6.      $\text{add}(R, \text{getWSFromWSN}(m, SN, n))$
  7. **EndIf**
- 

In the constructing process of a sustainable service Mashup, services are recommended again by Algorithm 3 if services in the constructing Mashup are changed. In this way, the instant recommendation is implemented.

Algorithm 3 first uses `getWSFromMashups` to recommend  $N$  services. When the number of services returned by `getWSFromMashups` is less than  $N$ , the remaining services are recommended by `getWSFromWSN`. Therefore, the time complexity of Algorithm 3 can be obtained by the complexity of `getWSFromMashups` and `getWSFromWSN`. The time complexity of `getWSFromMashups` is mainly related to the number of Mashups (denoted as  $|M|$ ) in the existing Mashup library and the average number of services contained in each Mashup (denoted as  $|S|$ ). As a result, its complexity is  $O(|M|*|S|)$ . As for `getWSFromWSN`, when the correlation between services in  $SN$  is calculated offline, its time complexity is only related to the number of recommended services  $N$ , so its complexity is  $O(N)$ . As  $N$  is usually much less than  $|M|*|S|$ , the time complexity of Algorithm 3 is  $O(|M|*|S|)$ .

We take the sustainable service Mashup in Figure 1 as an example to introduce the process of Web services recommendation in Algorithm 3. Suppose that  $N$  Web services need to be recommended each time by Algorithm 3 when the user selects a Web service. After the user selects the service `getLostPersons` to identify the victim, the algorithm finds a group of Mashups from the existing Mashup library that have a similarity value greater than the threshold through the used service `getLostPersons`, as well as its input parameters assigned by the user and retained output parameters. Thus, the relevancy values of each service except `getLostPersons` in these similar Mashups are calculated by Equation (4). If the number of obtained services is greater than  $N$ , the top  $N$  services are regarded as recommended services. On the contrary, if the number of services obtained is less than  $N$  (assuming  $M$ ), the  $N-M$  services with the highest correlation degree values with `getLostPersons` can be obtained from  $SN$  by Equation (6), and they are regarded as another part of the recommended services. In special cases, if the number of candidate services based on similar Mashups and  $SN$  is still less than  $N$ , only the services obtained according to Algorithm 3 are recommended. Repeating the process, this algorithm automatically recommends  $N$  services to the user after the user selects a new service to the current Mashup. In this way, the instant service recommendation for a sustainable service Mashup is provided.

The proposed approach can cope with the problems of the existing two kinds of instant service recommendation approaches in Table 1. First, this approach makes use of existing Mashups to find candidate services through similar Mashups. The idea is similar to the service recommendation approaches based on composition history listed in Table 1. The difference is that the approach in this paper takes the same services used and the usage of the same services into account to measure the similarity of Mashups. Thus, the implicit requirements of users can be described more accurately.

In addition, the complexity of searching similar Mashups is lower than those based on Mashup graph matching [25,26], making this approach more suitable for the instant recommendation of sustainable service Mashups. Although service recommendation based on composition history is limited by the scale of existing Mashups, the proposed approach can find other services further based on *SN* when services cannot be obtained as required by existing similar Mashups. Because the service hyperlinks can be obtained in many ways [33,36], it is not limited by the scale of existing service compositions or Mashups. As a result, our approach can solve the problem that the approaches based on composition history exist. Second, for the approaches that only consider service parameter correlations, too many correlations may be built because of different semantic granularity of service parameters, leading to poor effectiveness in recommending services. As is shown in the two examples of Section 5.2, the service that provides search function has the input parameter keyword may have hyperlinks with a large number of other services in *SN* because the input and output parameters of most services can be correlated with its input. However, most of these linked services may have nothing to do with the actual requirements of the user when the service is used in a sustainable service Mashup. As a result, the effectiveness of recommendation may be lowered. The proposed approach first leverages similar Mashups to discover the user's implicit requirements based on the used services. Thus, services can be recommended more accurately compared with the approaches that only consider the directly linked services by parameter correlations of services.

## 7. Experimental Evaluation

In this section, we present the experiments to evaluate the effectiveness of the proposed instant service recommendation approach.

### 7.1. Data Set and Evaluation Metrics

We get a group of available Web services and Mashups from ProgrammableWeb.com, as well as existing services and Mashups in our service composition platform. We get 1180 Web services (including REST and SOAP services) and 638 Mashups in total. By our service composition platform, these services can be encapsulated into the uniform services and composed to build these Mashups. Service hyperlinks can be obtained through the hyperlink modeling approach proposed in [33]. Meanwhile, we divide the obtained Mashups into two groups. One is the test data set *T*, containing 100 Mashups, in which each Mashup contains at least three services and fit the demands of a sustainable service Mashup. Another group that contains 500 Mashups is randomly selected from the remaining Mashups, and it is regarded as the Mashups library. We select the test data set and Mashup library randomly three times to conduct the experiments and take the average values of three times as the final result.

The approach proposed in this paper belongs to TopN recommendation. Therefore, the precision and recall are used as the evaluation metrics. Furthermore, coverage is used as another metric to verify the effectiveness of the approach in recommending infrequently used services and newly added services. Specifically, for each Mashup in the test data set, *N* services (the set is denoted as  $S_R$ ) are recommended according to the 1/3 front services by the proposed algorithm. The service set made up of the back 2/3 services in this Mashup is denoted as  $S_U$ . The precision, recall, and coverage can be measured by Equation (7). First,  $S_R$  is all the recommended services using Algorithm 3, and  $S_U$  is all the services that should be recommended. Therefore,  $S_R \cap S_U$  indicates the correctly recommended services. Thus, we can get the Precision and Recall in Equation (7). Second, in the formula of coverage, the numerator is the number of total services recommended for the 100 Mashups in the test data set *T*, and the denominator is the number of all the services (denoted as *S*). Through dividing the number of all recommended services by the number of all services, the coverage of the recommendation approach can be obtained.

$$Precision = \frac{S_R \cap S_U}{S_U} \quad Recall = \frac{S_R \cap S_U}{S_R} \quad Coverage = \frac{|\cup_{m \in T} S_R|}{|S|} \quad (7)$$

In order to evaluate the effectiveness of our approach, we first analyze some features of our algorithm. Specifically, we examine the effect of a few parameters on the algorithm, including the number of existing Mashups, the scale of  $SN$ , the number of services to be recommended, and the Mashup similarity threshold  $\lambda$ . Second, we compare the algorithm with other related approaches to observe the effect on the precision and recall. Finally, we observe the coverage value of the proposed approach and compare it with related approaches to analyze the effect on newly added and less frequently used services.

### 7.2. Evaluation Approach

First, we observe the influence of the number of existing Mashups and the scale of  $SN$  on the recommendation algorithm. We divide the existing Mashups library into five groups which contain 50, 100, 200, 300, and 500 Mashups, respectively. Meanwhile, we adjust service hyperlinks according to the number of existing Mashups. For example, for the case of 200 Mashups, we randomly delete the service hyperlinks in  $SN$ , and keep 2/5 (200/500) hyperlinks. In this way, the remaining  $SN$  is regarded as the corresponding network. Through five groups of different Mashups libraries and  $SN$ s, we obtain the precision, recall, and coverage of the algorithm.

Second, we compare our approach with the following related approaches. (1) The approach only uses parameter correlations to recommend services. In this approach, correlations or hyperlinks among services are built according to the input and output parameters of the services. Then, services having correlations or hyperlinks with those used services are recommended [26]. In our experiments, we get services linked by the hyperlinks of the front 1/3 services used in the constructing Mashup, and select  $N$  services randomly as the recommendation. (2) The approach counts the co-existence of services in existing Mashups and gets the similarity of services. In this approach, services similar to the used services are recommended [38]. In our experiments, we use these two approaches to recommend services and compare their results with ours. Because these three approaches are all affected by the number of existing Mashups and the scale of  $SN$ , we divide existing Mashups into five groups and deal with the  $SN$  as above. We observe the precision, recall, and coverage of these three approaches in these five cases. In this experiment, the Mashup similarity threshold  $\lambda$  is 0.4, and the number of recommended services is 5. In addition, we compare the running time of these three approaches to evaluate the efficiency and availability of these approaches.

Third, we observe the influence of the number of recommended services ( $N$ ) and the Mashup similarity threshold ( $\lambda$ ). We set fixed Mashups and the scale of  $SN$  to observe the influence of these two parameters on the algorithm's precision and recall.

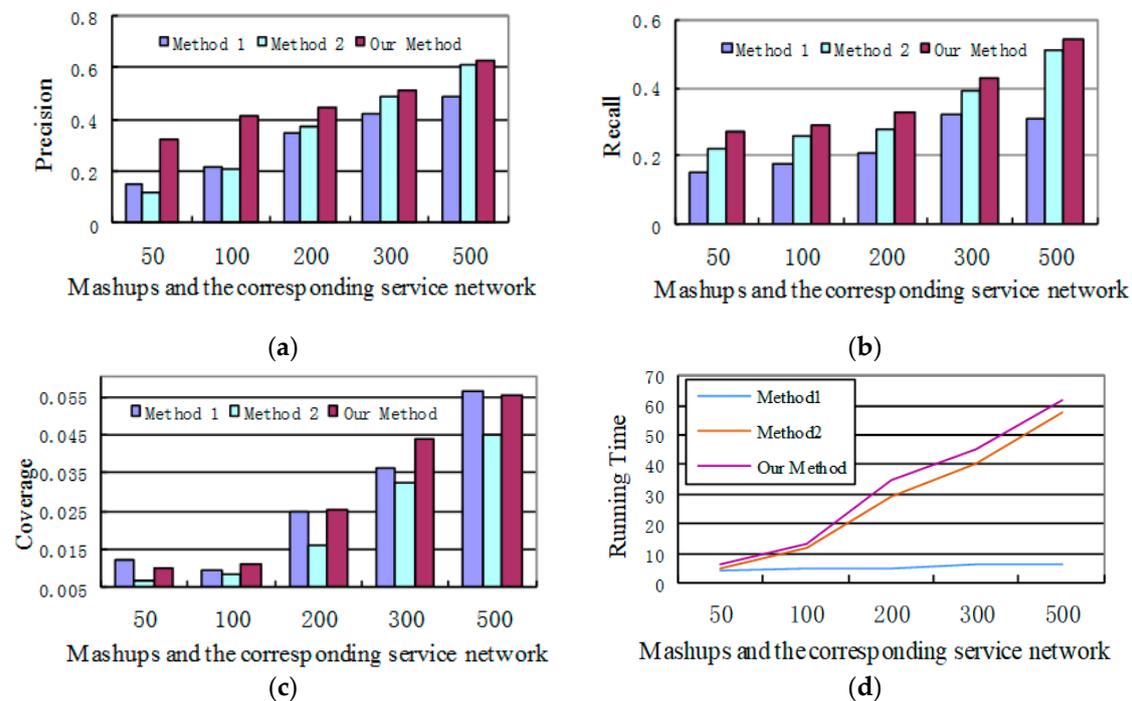
### 7.3. Results

Figure 4 shows the experimental results, based on which we can get the following conclusions.

First, the proposed approach is influenced by the number of existing Mashups and the scale of  $SN$ . If the scale of the existing Mashups,  $SN$ , and the number of recommended services are all small, the precision, recall, and coverage are not high. As the scale of existing Mashups and  $SN$  increases, the approach can find more similar Mashups and more services with higher correlation degree. As a result, the values of the above three metrics are all increased.

Second, compared with the first approach, the precision and recall of our approach are higher. In addition, the coverage of the first approach is comparable to our approach. If the scale of  $SN$  is small, services are selected randomly in this approach. Thus, the less frequently used services can be recommended. Therefore, the number of recommended services is comparable to our approach, making its coverage comparable to our approach. However, recommended services are not selected from the point of user requirement in the first approach. The services having parameter correlations with a service in the constructing Mashup may belong to multiple domains, and some of them may have little relation with the requirement of the constructing Mashup. In our approach, both services

with more comprehensive parameter correlations and those in the similar Mashups are all considered. Therefore, our approach has higher precision and recall than the first approach.



**Figure 4.** The experimental results. (a) Precision, (b) recall, (c) coverage, (d) complexity.

Compared with the second approach, the precision, recall, and coverage of our approach are all higher when the number of existing Mashups is small. Meanwhile, if the existing Mashups reach a certain scale, three metrics of the second approach are comparable to our approach. This is because the specified number of services cannot be recommended only depending on the existing Mashups using the second approach under the condition that the scale of the existing Mashups and SN is small. However, candidate services can be selected through the SN when services cannot be recommended only using existing Mashups in our approach. Specifically, for services that are not popular or less frequently used in the constructing Mashup, services can also be recommended based on SN. Hence, the coverage of our approach is higher. Through the experiment, we find that if existing Mashups reach a certain scale, services can be recommended only using existing Mashups in our approach. In this case, SN is not needed for service recommendation. That is, services that are recommended through existing Mashups typically have parameter correlations.

In addition, in terms of the time complexity, the complexity of the proposed approach is higher than the other two methods according to Figure 4d. This is because the proposed approach leverages both service parameter correlations and existing similar Mashups. However, the experimental results show that compared with method 1 and method 2, the proposed algorithm can return results in less than 100 ms under different Mashups libraries and scales of SN. Therefore, the requirements of instant service recommendation in sustainable service Mashups can be met using our approach.

To observe the influence of  $\lambda$ , three services are recommended for each Mashup in the test data set. The numbers of existing Mashups are set to 200 and 500, respectively, and SN is processed as above. The value of  $\lambda$  ranges from 0 to 1 with 0.1 increment each time. Figure 5 shows the values of the precision as  $\lambda$  changes under the conditions that the numbers of existing Mashups are 200 and 500, respectively. Based on the obtained results, we can see that the precision is the best when  $\lambda = 0.4$ .

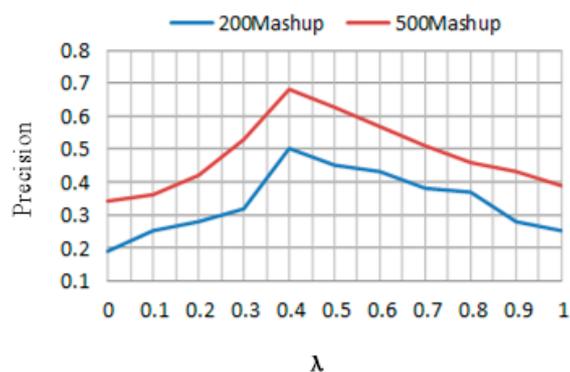


Figure 5. The influence of  $\lambda$  on the recommendation.

In order to observe how the number of recommended services ( $N$ ) influences the effectiveness of the recommendation, we set different values for  $N$  to get the precision and recall. The experiment is conducted under the conditions that the number of existing Mashups is 500, and  $SN$  is in the corresponding scale. Because the proposed approach does not need to recommend many services,  $N$  is set to 1, 3, 5, 7, and 9, respectively. The obtained results are shown in Figure 6, based on which we can see that the precision and recall are the highest when  $N = 5$  and the number of existing Mashups is 500.

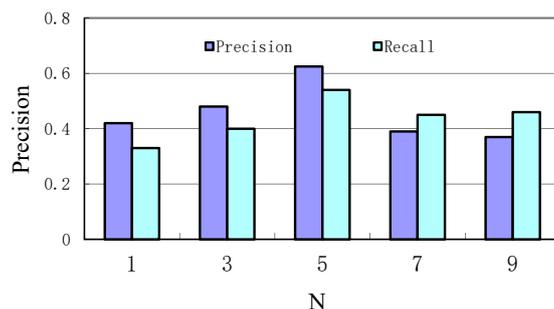


Figure 6. The influence of  $N$  on the recommendation.

Based on the experimental results, we can draw the final conclusions. For sustainable service Mashups, the effectiveness of our approach is improved as the scale of existing Mashups and  $SN$  increases. If the scale of existing Mashups and  $SN$  reaches a certain scale, the precision, recall, and coverage of our approach are higher than existing approaches.

To sum up, we can see from the experimental results that when the number of existing composition history or Mashups reaches a certain scale, the proposed approach has the similar precision and recall with the existing service recommendation approaches based on composition history. Differently, the proposed approach has better coverage. While the number of existing compositions or Mashups is small, the precision and recall of the proposed approach are better than existing composition history-based service recommendation approaches because it can find other suitable services through  $SN$ . Compared with approaches that only consider the parameter correlation, our approach has better precision and recall. In addition, the time complexity of our approach is suitable for instant service recommendation in the building process of a sustainable service Mashup.

## 8. Conclusions

In this paper, we propose an instant Web service recommendation approach for sustainable service Mashups that considers services used in similar Mashups and parameter correlations among the input and output parameters of Web services. First, services are selected from existing Mashups that are similar to the constructing Mashup. The similarity between two Mashups is computed based

on the same services used and the way they are used. Second, services are selected through the service network that is built based on the services and their hyperlinks from the perspective of actual operations of Web service composition. We propose to compute the service correlation degree based on parameter correlations. According to the experimental results, the shortcomings of existing approaches in Web services instant recommendation for sustainable service Mashups are properly tackled by combining two recommendation approaches. Meanwhile, the proposed approach has better precision, recall, and coverage on the whole, and can recommend Web services instantly to a user during the construction of a sustainable service Mashup. In conclusion, the proposed approach is more suitable for instant service recommendation of sustainable service Mashups.

With the wider application of Web services, more and more service compositions or Mashups will be accumulated on the Internet, and more parameter correlations between services will be found, which help to gradually form service big data. How to combine the machine learning techniques, especially the deep learning techniques, to achieve more accurate and efficient service recommendation, is an important issue of the future work. In addition, the execution of composed services can be regarded as a business process, advanced process modeling and analysis techniques, e.g., [39–41], can be applied for service behavior analysis.

**Author Contributions:** Conceptualization, F.Z.; methodology, F.Z. and C.L.; validation, B.C.; formal analysis, F.Z. and B.C.; data curation, B.C.; writing—original draft preparation, F.Z.; writing—review and editing, F.Z., B.C.; project administration, F.Z. and C.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the Education Ministry Humanities and Social Science Research Youth Fund Project of China (“User-Steering Multi-Source Education Data Integration Approach Research in Big Data Environment” with grant number 19YJCZH240), 2019 Qingdao social science planning research project (grant number QDSKL1901123), the NSFC (grant numbers 61902222 and 31671588), by Sci. & Tech. Development Fund of Shandong Province of China (grant numbers ZR2017MF027), Taishan Scholars Program of Shandong Province (tsqn201909109), Engineering and Technology R&D Center of IIOT in Colleges of Shandong Province (QingDao Technical College, grant number KF2019002), and SDUST Research Fund (grant number 2015TDJH102).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

- Hayyolalam, V.; Kazem, A.A.P. A systematic literature review on QoS-aware service composition and selection in cloud environment. *J. Netw. Comput. Appl.* **2018**, *110*, 52–74. [CrossRef]
- Vakili, A.; Navimipour, N.J. Comprehensive and systematic review of the service composition mechanisms in the cloud environments. *J. Netw. Comput. Appl.* **2017**, *81*, 24–36. [CrossRef]
- Sun, M.Y.; Zhou, Z.B.; Wang, J.P.; Du, C.; Gaaloul, W. Energy-efficient IoT service composition for concurrent timed applications. *Future Gener. Comput. Syst.* **2019**, *100*, 1017–1030. [CrossRef]
- Asghari, P.; Rahmani, A.M.; Javadi, H.H.S. Service composition approaches in IoT: A systematic review. *J. Netw. Comput. Appl.* **2018**, *120*, 61–77. [CrossRef]
- Barika, M.; Garg, S.; Zomaya, A.Y.; Wang, L.Z.; Moorsel, A.V.; Ranjan, R. Orchestrating big data analysis workflows in the cloud: Research challenges, survey, and future directions. *ACM Comput. Surv.* **2019**, *52*, 1–41. [CrossRef]
- Gudgin, M.; Hadley, M.; Mendelsohn, N.; Moreau, J.J.; Nielsen, H.F.; Karmarkar, A.; Lafon, Y. Simple Object access Protocol (SOAP) Version 1.2. W3C Recommendation. 2007. Available online: <http://www.w3.org/TR/soap12/> (accessed on 9 March 2020).
- Fielding, R. *Architectural Styles and the Design of Network-Based Software Architectures*; University of California: Oakland, CA, USA, 2000.
- Pautasso, C.; Zimmermann, O.; Amundsen, M.; Lewis, J.; Josuttis, N. Microservices in Practice, Part 2: Service Integration and Sustainability. *IEEE Softw.* **2017**, *34*, 97–104. [CrossRef]
- Valderas, P.; Torres, V.; Pelechano, V. A Microservice Composition Approach based on the Choreography of BPMN fragments. *Inf. Softw. Technol.* **2020**, *127*, 106370. [CrossRef]
- Lemos, A.L.; Daniel, F.; Benatallah, B. Web service composition: A survey of techniques and tools. *ACM Comput. Surv.* **2015**, *48*, 3. [CrossRef]

11. Garriga, M.; Mateos, C.; Flores, A.; Cechich, A.; Zunino, A. RESTful service composition at a glance: A survey. *J. Netw. Comput. Appl.* **2016**, *60*, 32–53. [[CrossRef](#)]
12. Deng, S.G.; Wu, H.Y.; Taheri, J.; Zomaya, A.Y.; Wu, Z.H. Cost Performance Driven Service Mashup: A Developer Perspective. *IEEE. Trans. Parallel Distrib. Syst.* **2016**, *27*, 2234–2247. [[CrossRef](#)]
13. Wang, G.; Zhang, S.L.; Han, Y.B.; Zhang, Z.M. A Dataflow-Pattern-Based Recommendation Framework for Data Service Mashup. *IEEE. Trans. Serv. Comput.* **2015**, *8*, 889–902. [[CrossRef](#)]
14. Hang, F.F.; Zhao, L.P. Supporting End-User Service Composition: A Systematic Review of Current Activities and Tools. In Proceedings of the 2015 IEEE International Conference on Web Services, New York, NY, USA, 27 June–2 July 2015; pp. 479–486.
15. Bai, L.; Wei, J.; Huang, X.; Ye, D.; Huang, T. An Exploratory Service Composition Approach for Mobile Application. *J. Softw.* **2015**, *26*, 2191–2211.
16. Han, Y.B.; Wang, H.C.; Wang, J.W.; Yan, S.Y.; Zhang, C. An End-User-Oriented Approach to Exploratory Service Composition. *J. Res. Dev.* **2006**, *43*, 1895–1903.
17. Yang, J.; Zhu, X.J.; Zhou, X.Z.; Liu, Y. Personalized Web Service Recommendation Based on Heterogeneous Social Network. *Acta Electron. Sin.* **2020**, *48*, 341–349.
18. Liu, J.; Tang, M.; Zheng, Z.; Liu, X.; Lyu, S. Location-Aware and Personalized Collaborative Filtering for Web Service Recommendation. *IEEE. Trans. Serv. Comput.* **2016**, *9*, 686–699. [[CrossRef](#)]
19. Chen, S.H.; Fan, Y.S.; Tan, W.; Zhang, J.; Bai, B.; Gao, Z.F. Time-Aware Collaborative Poisson Factorization for Service Recommendation. In Proceedings of the IEEE International Conference on Web Services (ICWS), San Francisco, CA, USA, 27 June–2 July 2016; pp. 196–203.
20. Zhang, Y.W.; Ai, X.; He, Q.; Zhang, X.Y.; Dou, W.C.; Chen, F.F.; Chen, L.; Yang, Y. Personalized Quality Centric Service Recommendation. In *International Conference on Service-Oriented Computing*; Springer: Cham, Switzerland, 2017; pp. 528–544.
21. Chen, L.; Wu, J.; Jian, H.Y.; Deng, H.B.; Wu, Z.H. Instant Recommendation for Web Services Composition. *IEEE. Trans. Serv. Comput.* **2013**, *7*, 586–598. [[CrossRef](#)]
22. Huang, K.M.; Fan, Y.S.; Tan, W.; Li, X. Service Recommendation in an Evolving Ecosystem: A Link Prediction Approach. In Proceedings of the 20th International Conference on Web Services, Santa Clara, CA, USA, 28 June–3 July 2013; pp. 507–514.
23. Ni, Y.Y.; Fan, Y.S.; Huang, K.M.; Bi, J. Negative-Connection-Aware Tag-based Association Mining and Service Recommendation. In Proceedings of the 12th International Conference of Service-Oriented Computing, Paris, France, 3–6 November 2014; pp. 419–428.
24. Kang, G.S.; Liu, J.X.; Tang, M.D.; Liu, X.Q.; Cao, B.Q.; Xu, Y. AWSR: Active Web Service Recommendation based on Usage History. In Proceedings of the 19th International Conference on Web Services, Beijing, China, 24–29 June 2012; pp. 186–193.
25. Elmeleegy, H.; Ivan, A.; Akkiraju, R.; Goodwin, R. Mashup Advisor: A Recommendation Tool for Mashup Development. In Proceedings of the 15th IEEE International Conference on Web Services, Beijing, China, 12–15 October 2008; pp. 337–344.
26. Abiteboul, S.; Greenshpan, O.; Milo, T.; Polyzotis, N. Matchup: Autocompletion for Mashups. In Proceedings of the 25th International Conference on Data Engineering, Shanghai, China, 29 March–2 April 2009; pp. 1479–1482.
27. Yan, S.Y.; Han, Y.B.; Wang, J.; Liu, C. A User-Steering Exploratory Service Composition Approach. In Proceedings of the Second International Conference on Services Computing, Honolulu, HI, USA, 8–11 July 2008; pp. 309–316.
28. Zhao, C.T.; Ma, C.; Zhang, J.; Zhang, J. HyperService: Linking and Exploring Services on the Web. In Proceedings of the 17th International Conference on Web Services, Miami, FL, USA, 5–10 July 2010; pp. 17–24.
29. Wang, G.L.; Yang, S.H.; Han, Y.B. Mashroom: End-User Mashup Programming Using Nested Tables. In Proceedings of the 18th International Conference on World Wide Web, Madrid, Spain, 20–24 April 2009; pp. 861–870.
30. Liu, C.; Wang, J.W.; Zhu, M.L.; Han, Y.B. Discovery of Service HyperLinks with User Feedbacks for Situational Service Mashup. *Int. J. Database Theory Appl.* **2015**, *8*, 71–80. [[CrossRef](#)]
31. Zhu, M.L.; Liu, C. Discovery and Reuse of Service Hyperlinks for Efficient Service Mashup. In Proceedings of the Eighth IEEE World Congress on Services, Anchorage, AK, USA, 27 June–2 July 2014; pp. 147–154.

32. Goldberg, D.; Nichols, D.A.; Oki, B.M.; Terry, D.B. Using Collaborative Filtering to Weave an Information Tapestry. *Commun. ACM* **1992**, *35*, 61–70. [[CrossRef](#)]
33. Zhang, F.; Wen, Y.; Wei, Y.S. Data Correlation of Data Services Oriented Service Hyperlink and Modeling Research. *J. Chin. Comput. Syst.* **2017**, *38*, 328–333.
34. Bianchini, M.; Gori, M.; Scarselli, F. Inside PageRank. *ACM Trans. Internet Technol.* **2015**, *5*, 92–128. [[CrossRef](#)]
35. Haveliwala, T.H. Topic-Sensitive PageRank. In Proceedings of the 11th International Conference on World Wide Web, Honolulu, HI, USA, 7–11 May 2002; pp. 494–502.
36. Zhang, F.; Zeng, Q.; Duan, H.; Liu, C. Composition Context-Based Web Services Similarity Measure. *IEEE. Access* **2019**, *7*, 65195–65206. [[CrossRef](#)]
37. Gu, Z.F.; Li, J.Z.; Hu, J.Q.; Xu, B.; Wang, K.H. Modeling and Application of Data Links among Web Services. *Chin. J. Comput.* **2008**, *31*, 1309–1318. [[CrossRef](#)]
38. Pan, W.F.; Li, B.; Shao, B.; He, P. Service Classification and Recommendation based on Software Network. *Chin. J. Comput.* **2011**, *34*, 2355–2369. [[CrossRef](#)]
39. Liu, C.; Zeng, Q.; Cheng, L.; Duan, H.; Zhou, M.; Cheng, J. Privacy-preserving Behavioral Correctness Verification of Cross-organizational Workflow with Task Synchronization Patterns. *IEEE Trans. Autom. Sci. Eng.* **2020**, *99*, 1–12. [[CrossRef](#)]
40. Duan, H.; Liu, C.; Zeng, Q.; Zhou, M. Refinement-based Hierarchical Modeling and Correctness Verification of Cross-organization Collaborative Emergency Response Process. *IEEE Trans. Syst. Man Cybern. Syst.* **2020**, *50*, 2548–2559.
41. Zeng, Q.; Liu, C.; Duan, H.; Zhou, M. Resource Conflict Checking and Resolution Controller Design for Cross-organization Emergency Response. *IEEE Trans. Syst. Man Cybern. Syst.* **2020**, *50*, 3685–3700. [[CrossRef](#)]

**Publisher’s Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).