Hindawi
Mathematical Problems in Engineering
Volume 2019, Article ID 1495676, 11 pages
https://doi.org/10.1155/2019/1495676

Research Article

Clustering Services Based on Community Detection in Service Networks

Shiyuan Zhou[1,2] **and Yinglin Wang**[1]

[1]*School of Information Management and Engineering, Shanghai University of Finance and Economics, Shanghai 200433, China*
[2]*Jiaxing University, Jiaxing 314001, China*

Correspondence should be addressed to Yinglin Wang; yinlwang@zoho.com.cn

Received 26 September 2019; Accepted 15 November 2019; Published 2 December 2019

Guest Editor: Chunlai Chai

Service-oriented computing has become a promising way to develop software by composing existing services on the Internet. However, with the increasing number of services on the Internet, how to match requirements and services becomes a difficult problem. Service clustering has been regarded as one of the effective ways to improve service matching. Related work shows that structure-related similarity metrics perform better than semantic-related similarity metrics in clustering services. Therefore, it is of great importance to propose much more useful structure-related similarity metrics to improve the performance of service clustering approaches. However, in the existing work, this kind of work is very rare. In this paper, we propose a SCAS (service clustering approach using structural metrics) to group services into different clusters. SCAS proposes a novel metric *A2S* (atomic service similarity) to characterize the atomic service similarity as a whole, which is a linear combination of *C2S* (composite-sharing similarity) and *A3S* (atomic-service-sharing similarity). Then, SCAS applies a guided community detection algorithm to group atomic services into clusters. Experimental results on a real-world data set show that our SCAS performs better than the existing approaches. Our *A2S* metric is promising in improving the performance of service clustering approaches.

1. Introduction

Web service is a new web application mode that has been widely distributed and invoked on the whole Internet [1, 2]. With the rapid development of SaaS (software-as-a-service) and SOA (service-oriented architecture) technologies, web services on the Internet are showing a trend of rapid growth [1, 2]. The traditional way to develop software has moved to service-oriented development. More and more software systems are developed by composing existing services on the Internet, and it has become a promising way to develop software. There are a large number of web services with different types on the Internet, which makes the matching of requirements and services become a complex process that needs to be solved through a process composed of many steps such as clustering, selection, and evaluation of services [3]. Thus, in the initial stage of service matching, how to cluster services has become an urgent problem for the selection of service set that can meet the requirements of users.

Service clustering has been widely regarded as one of the effective ways to improve service matching [1]. In the literature, many different clustering approaches have been proposed to group services into clusters. Among them, the most widely used approaches are based on the mining of features of WSDL documents [4–6]. These approaches first extract key features such as WSDL description, WSDL type, WSDL port, and web service name from the WSDL document. Then, based on these extracted features, they calculate the service similarity between services by using methods like cosine similarity, so as to organize services into clusters. However, due to the fact that WSDL documents usually contain few descriptions, these approaches usually fail to achieve satisfactory clustering results. Worse still, these approaches usually ignore the semantic association between services. Therefore, many other approaches based on the

LDA topic model and its extensions have been proposed [1, 2]. They first extract the hidden topics in web services, and then use low-dimensional topic vectors to encode the functional attributes of web services. Finally, they calculate the similarity between services so as to organize services into different clusters. The experimental results show that these topic model-based approaches can achieve good results. However, the WSDL documents for web services are usually short in length, containing very little semantic (or functional) information that can be used by LDA-based clustering approaches. Thus, the semantic-related similarity metrics seem to not perform better than structure-related similarity metrics in clustering web services [1]. Therefore, it is of great importance to propose much more useful structure-related similarity metrics to promote the performance of service clustering approaches.

The purpose of this work is to propose some novel structure-related similarity metrics so as to combine them together to quantify the similarity between services as a whole. It is expected that the novel similarity metrics can be used to improve the performance of service clustering approaches. In this paper, we propose a SCAS (service clustering approach using structural metrics) approach to group services into different clusters. First, SCAS uses an ASAN (atomic service affiliation network) to represent composite services, atomic services, and their relationships. Based on the ASAN, we propose our first structural metric, *C2S* (composite-sharing similarity) which is a similarity owing to the sharing of common composite services. Second, by projecting the ASAN onto the atomic services, we build an ASDN (atomic service dependency network) to denote atomic services and their relationships. Based on ASDN, we propose our second structural metric, *A3S* (atomic-service-sharing similarity) which is a similarity owing to the sharing of common atomic services. Third, we propose the final structural metric, *A2S* (atomic service similarity) to characterize the atomic service similarity as a whole, which is a linear combination of *C2S* and *A3S*. Finally, we propose a SSN (service similarity network) and a community detection algorithm, GUIDA (Guided community detection algorithm), to find the community structures in the SSN. The communities correspond to the atomic service clusters. GUIDA only takes the similarity between atomic services as its input and can determine the number of clusters for the atomic service set automatically. Our experimental study is carried out on a real-world service data set collected from a famous service directory ProgrammableWeb (PWeb) (https://www.programmableweb.com). The comparative studies show that our approach performs better than *C2S*-based approaches and semantic-based approaches.

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

(i) The introduction of a novel structural metric to characterize the service similarity, which combines two structure similarity metrics simultaneously, i.e., one is proposed to characterize the composite-sharing similarity and the other one is used to characterize atomic-service-sharing similarity. Our metric is very different from the existing structural similarity metrics which usually only character the composite-sharing similarity, ignoring the atomic-service-sharing similarity.

(ii) The application of a parameter-free community detection algorithm to detect the communities and group services into clusters. Our approach only takes the similarity between services as the input and can determine the number of clusters in the service set automatically. It is very different from the existing approaches which needs to set the number of clusters beforehand to cluster services.

(iii) A real-world data set is built to validate our approach empirically, which is very different from those approaches using a simulation way to validate their approaches.

The reminder of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents our SCAS approach, with the focus on service networks, similarity metrics, and the GUIDA algorithm used to group services into clusters. Section 4 presents the empirical validation of our SCAS approach. We draw conclusions in Section 5.

2. Related Work

As mentioned above, service clustering is one of the effective technologies to help the matching of requirements and services. Its main goal is to group services into different clusters according to the similarity of their functions. Generally, services in the same cluster are similar to each other while services in different clusters are different. Till now, lots of service clustering approaches have been proposed. According to the information that they used to characterize service similarity, they can be roughly divided into two categories: function-based service clustering approaches [1, 7–11] and nonfunction-based clustering approaches [12–16].

Function-based service clustering approaches employed the functional information of services (e.g., description documents, tags) to group services into clusters. Chen et al. [7, 8] proposed an improved service clustering approach which integrates WSDL documents and service tags to improve clustering accuracy. They think that tags represent the functional characteristics of services, which can be combined with WSDL documents to much more accurately determine the functional category of services. Shi et al. [9] first organized words in the service description of all services according to semantics using the Word2Vec, and then they proposed an improved clustering approach by considering the auxiliary function of the words which belong to the same cluster with the words in the service description document. Liu et al. [10] first trained a preliminary SVM classifier based on a small set of manually labeled samples. Then, they recommended potential labels for other services that are not manually labeled based on the SVM classifier. Finally, services are clustered based on these labeled samples. Cao et al. [11] extracted the hidden topic information of mashup

services based on the LDA topic model, and then clustered the mashup services based on the LDA topic model. Chen et al. [8] proposed an enhanced probabilistic topic model (WT-LDA), which can model Web service description documents and service tag information simultaneously so as to group services into clusters. Shi et al. [9] proposed a Mashup-API-Tag probabilistic topic model to model the service description, label information, and the composite relationship information among services, so as to improve the accuracy of topic information extraction. Pan and Chai [1] proposed a novel mashup service clustering approach based on a structural similarity and a genetic algorithm-based clustering algorithm. Their approach can group mashup services into clusters effectively and determine the number of clusters automatically.

Nonfunction-based clustering approaches employed the nonfunctional information of services (e.g., QoS and physical locations) to group services into clusters. Liu et al. [12] proposed a service clustering algorithm based on an ontology which contains the information of service name, performance, interface, and QoS attributes. Zhou et al. [13] used a genetic algorithm to group services into clusters by using the QoS information of services. To avoid converging to a local optimum, they introduced the concept of entropy to measure and improve the population diversity of their genetic algorithm. Chen et al. [14] proposed a service clustering algorithm based on the historical QoS data with similar physical characteristics to ensure that services in the same cluster have similar physical environment characteristics. These clustering algorithms only consider nonfunctional attributes, thus usually have a relatively small value of execution complexity. However, these algorithms usually do not have good scalability because nonfunctional attributes of services are often difficult to obtain and unstable.

Generally, the structure-related similarity metrics perform better than semantic-related similarity metrics in grouping services into clusters since the WSDL documents and service descriptions do not always carry a lot of function-related information. However, to the best of our knowledge, there is only one structure-related similarity metric, *C2S* (composite-sharing similarity) [3]. Therefore, it is of great importance to propose much more useful structure-related similarity metrics to promote the performance of service clustering approaches. In this work, we proposed an atomic-service-sharing similarity metric. Our metric is different from *C2S*. Our metric is based on the sharing relationship of atomic services while *C2S* is based on the sharing relationship of composites.

3. SCAS Approach

In this paper, SCAS approach is proposed to group services into different clusters. It is composed of four steps: (1) SCAS uses an ASAN to represent composite service, atomic services, and their relationships and proposes the *C2S* metric. (2) SCAS builds an ASDN to denote atomic services and their relationships and proposes the *A3S* metric. (3) SCAS proposes the *A2S* metric to characterize the atomic service similarity as a whole. (4) SCAS applies a community detection algorithm to find the service clusters in the set of atomic services. The workflow of the SCAS approach is shown in Figure 1.

3.1. Services Profile Crawling. To group services into clusters, the first work that we should do is to craw the profile of services to be clustered. Generally, the profile of services are usually registered in a registration center where people can share their own developed services with other people over the world. Furthermore, people can find the right services to meet their requirements by searching and browsing in the registration center.

To the best of our knowledge, there are many famous service registration centers such as ProgrammableWeb (PWeb) (https://www.programmableweb.com), myExperiment (https://www.myexperiment.org/home), and Biocatalogue (https://www.biocatalogue.org). To carry out the service clustering work, we will crawl the profile of services stored in these registration centers. The information includes service name, the lower level services each composite service used, and the category each service belongs to. The information will be stored in the local database as to reduce the noises and finally build the data set for SCAS to perform the service clustering task.

3.2. Service Network Construction. Complex network theory has been widely used in software engineering to quantify software structure [17, 18], identify key classes [19, 20], and measure software quality [21–23]. In this work, to quantify the service similarity as a whole, we also apply the complex network theory and build three service networks, i.e., ASAN (atomic service affiliation network), ASDN (atomic service dependency network), and SSN (service similarity network), which are defined as follows.

3.2.1. Atomic Service Affiliation Network

Definition 1. ASAN is an affiliation network that can be denoted formally as ASAN = (V_c, V_a, E), where V_c denotes the node set of composite service, V_a denotes the node set of atomic services, and E denotes the unweighted undirected edge set between every pair of composite service and atomic service if the composite service uses the atomic service, i.e., if $v_i \in V_c$ uses $v_j \in V_a$, then it follows that $\{v_i, v_j\} \in E$. ASAN is essentially a two-mode graph. As a two-mode graph, any edge can only exist between the different node sets V_c and V_a, i.e., $V_c \cap V_a = \varnothing$. To be specific, if $(v_i, v_j) \in E$, then it follows that $v_i \in V_c$ and $v_j \in V_a$. E will be associated with a $|V_c| \times |V_a|$ adjacency matrix ψ to encode the "use" relationships between a specific pair of composite service and atomic service. The entry of ψ, ψ_{ij}, is defined as

$$\psi_{ij} = \begin{cases} 1, & \{v_i, v_j\} \in E, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

We use a simple example shown in Figure 2 to show the idea to build the ASAN. The example is chosen from our

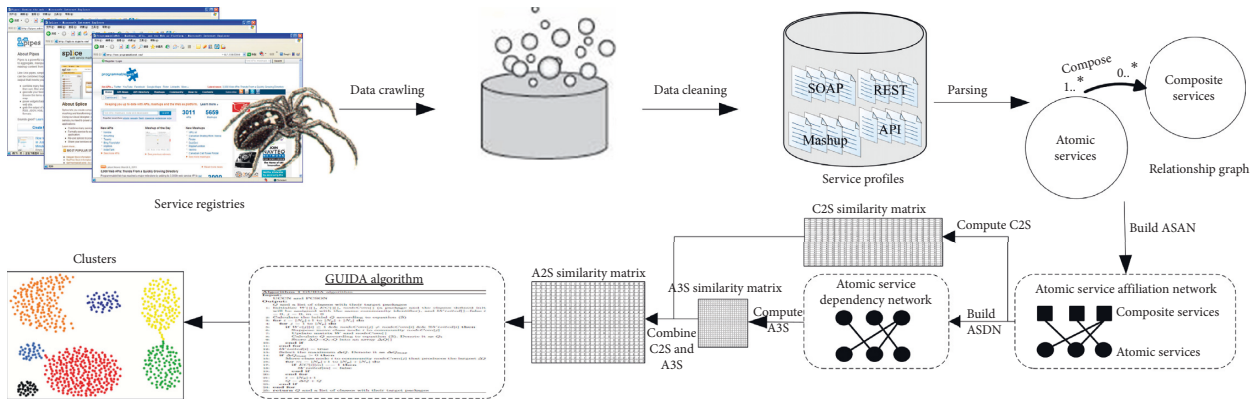


Figure 1: The workflow of SCAS. The workflow is adapted from that of [1, 2].

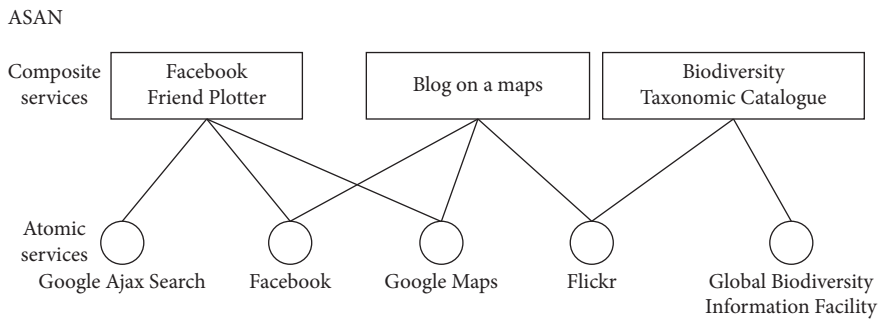


Figure 2: Illustration of ASAN.

data set used in Section 4 and is borrowed from [1, 2]. As shown in the example, the composite service (i.e., mashup "Facebook Friend Plotter (https://www.programmableweb.com/mashup/facebook-friend-plotter)") consists of three atomic services (i.e., APIs "Google Ajax Search (https://www.programmableweb.com/api/google-ajax-search)," "Facebook (https://www.programmableweb.com/api/facebook)," and "Google Maps (https://www.programmableweb.com/api/google-maps)"). Thus, in the corresponding ASAN, there exist three edges between composite service "Facebook Friend Plotter" and the other three atomic services "Google Ajax Search," "Facebook," and "Google Maps." By taking a similar way, we can establish other edges in the ASAN.

3.2.2. Atomic Service Dependency Network. ASAN describes the macro composition information between composite services and atomic services, and it also implicitly reflects the composition potential between the atomic services that exist in the same composite services. In other words, if two atomic services $v_j \in V_a$ and $v_k \in V_a$ are used together in ≥1 composite service(s), it follows that $v_j \in V_a$ and $v_k \in V_a$ have the probability to be composed together to build a composite service. Thus, in this work, we will build an ASDN (atomic service dependency network) to capture atomic services and their composition potential.

Definition 2. ASDN can be formally denoted as ASDN = (V_a, E_a), where V_a denotes the node set of atomic services (V_a is the same as that in ASAN) and E_a is the edge set between atomic services, signifying the composition potential between the atomic services, i.e., if $(v_j, v_k) \in E_a$, and then it follows that v_j and v_k have the probability to be composed together. Each edge will be assigned a weight to denote the number of composite services that the two atomic services commonly participate in. E_a will be associated with a $|V_a| \times |V_a|$ adjacency matrix ψ^a to encode the composition relations between a specific pair of atomic services. The entry of ψ^a, ψ^a_{ij}, is defined as

$$\psi^a_{ij} = \begin{cases} w, & \{v_i, v_j\} \in E_a, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

where w is the weight assigned to the edge $\{v_i, v_j\} \in E_a$, denoting the number of composite services that the two atomic services commonly participate in.

We also use a simple example shown in Figure 3 to show the idea to build the ASDN. ASDN is constructed from the example shown in Figure 2. As is shown in the example, since the composite service "Facebook Friend Plotter" uses the function of atomic services "Google Ajax Search" and "Google Maps," an edge between the node of "Google Ajax Search" and the node of "Google Maps" will be established in Figure 3. We establish all other edges in Figure 3 by taking a similar way.

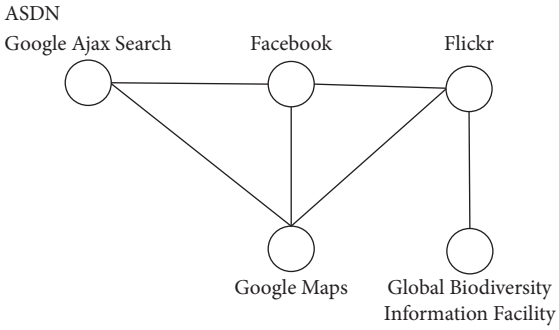


Figure 3: Illustration of ASDN.

In this work, ASDN is built by using the one-mode projection of ASAN on atomic services. Specifically, if two atomic services are used together in ≥1 composite service(s), then it follows that an edge will be established between the two atomic services in ASDN.

3.2.3. Service Similarity Network. We use SSN to denote services and their similarity relationships. Thus, SSN can be defined as follows:

Definition 3. SSN can be formally denoted as SSN = (V_a, E_s), where V_a denotes the node set of atomic services (V_a is the same as that in ASAN and ASDN) and E_s is the edge set between atomic services, signifying the similarity relationships between the atomic services, i.e., if $(v_j, v_k) \in E_s$, and then it follows that v_j and v_k are similar to each other. Each edge will be assigned a weight to denote the similarity between the two atomic services. E_s will be associated with a $|V_a| \times |V_a|$ adjacency matrix ψ^s to encode the similarity relations between a specific pair of atomic services. The entry of ψ^s (ψ^s_{ij}) is defined as

$$\psi^s_{ij} = \begin{cases} A2S(s,t), & \{s,t\} \in E_s, \\ 0, & \text{otherwise}, \end{cases} \tag{3}$$

where $A2S(s,t)$ is the similarity between atomic services s and t, which will be computed in Section 3.3.

3.3. Service Similarity Metrics. To group services into clusters, we usually need to quantify service similarity as a whole. In this work, service similarity is measured from the perspective of structure. As mentioned above, in this work, we quantify service similarity using two structure-related metrics, *C2S* (composite-sharing similarity) and *A3S* (atomic-service-sharing similarity). The service similarity as a whole is measured as the integration of *C2S* and *A3S*.

If the composite-sharing similarity between two atomic services s and t is denoted as $C2S(s,t)$, then it can be defined as

$$C2S(s,t) = \frac{|N_s \cap N_t|}{|N_s \cup N_t|}, \tag{4}$$

where N_s and N_t denote the number of composite services that atomic services s and t are used, respectively.

If the atomic-service-sharing similarity between two atomic services s and t is denoted as $A3S(s,t)$, then it can be defined as

$$A3S(u,v) = \begin{cases} \dfrac{\psi^a_{st}}{\sum_{k=1}^{n} \psi^a_{st}}, & \sum_{k=1}^{|V_a|} \psi^a_{kt} \neq 0, \\ 0, & \text{otherwise}. \end{cases} \tag{5}$$

Obviously, $A3S(s,t)$ is the ratio of ψ^a_{st} relative to the total number of weights on the edges that link to v_t; otherwise, $A3S(s,t)$ equals 0.

Furthermore, the atomic-service-sharing similarity between atomic services s and t is denoted as the maximum value of $A3S(s,t)$ and $A3S(t,s)$, i.e.,

$$A3S(s,t) = A3S(t,s) = \max(A3S(s,t), A3S(t,s)). \tag{6}$$

Then, the total similarity between atomic services s and t, $A2S(s,t)$, can be defined as

$$A2S(s,t) = \lambda \times C2S(s,t) + (1-\lambda) \times A3S(s,t), \tag{7}$$

where λ is the weight assigned to the component of equation (7).

In this work, λ is determined by CV (coefficient of variation) [24], which refers to a statistical metric that is used to measure the distribution of data points in a data series around the mean. CV is a helpful statistic in comparing the degree of variation from one data series to the other. CV is computed by deriving the ratio between the standard deviation and the mean. Thus, CV is defined as

$$\text{CV} = \frac{\text{std}}{\text{mean}}, \tag{8}$$

where std and mean are the standard deviation and mean of the sample, respectively.

In this work, we will first compute $C2S(s,t)$ (or $A3S(s,t)$) for all pairs of atomic services s and t ($s,t \in V_a$). Then, we compute the std and mean for the $C2S(s,t)$ (or $A3S(s,t)$) set so as to obtain the CV for *C2S* (or *A3S*). If we use CV_{c2s} (or CV_{a3s}) to denote the CV for *C2S* (or *A3S*), then it follows that

$$\lambda = \frac{\text{CV}_{c2s}}{\text{CV}_{c2s} + \text{CV}_{a3s}}. \tag{9}$$

3.4. Community Detection Algorithm

3.4.1. Modularity Q. In the literature, many different community detection algorithms have been proposed to organize nodes in a network into communities. Among them, a popular way used is to optimize a Q index [21, 25]. Q is short for modularity, which is used to quantify the quality of a specific partition of a network into communities. Q is based on the measurement of density of edges within communities compared with edges between

communities. In this work, we also applied Q to measure the quality of a specific clustering solution of atomic services. For SSN, a weighted undirected network, Q can be defined as

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j), \tag{10}$$

where m denotes the sum of the weight on all edges in the SSN, A_{ij} denotes the weight assigned to edge $\{i, j\}$ or $(\{j, i\})$, k_i and k_j are the sum of the weight on the edges linking to nodes i and j, respectively, c_i and c_j denote the communities into which nodes i and j are organized, respectively, and δ returns 1 when c_i equals c_j, 0, otherwise.

In the process of community detection in SSN, once an atomic service is moved from one community to another community, Q will be recalculated to decide whether to accept or reject this movement.

3.4.2. Community Detection Algorithm Flow. In this work, a Guided community detection algorithm (GUIDA) is introduced to perform the community detection task in SSN. GUIDA is originally proposed to refactor the class structure [22, 23]. The "guide" means GUIDA is different from the traditional community detection algorithms using a guidance way. Its "guide" reflects mainly in two aspects:

(i) Guide the initial division: composite services consist of a set of related atomic services. Thus, composite services can be seen as the initial communities of atomic services. There is no need to start the community detection process using a random way by assigning each atomic service to a random community. In this work, we will guide the initial division by grouping the atomic services in the same composite service to a same community. However, this will make some atomic services belong to different communities since some atomic services may participate in the composition of ≥1 composite service(s). Thus, in this work, we first assign atomic services participating in the composition of ≥1 composite service(s) to different communities. Then, for the atomic services participating in the composition of only one composite service, they will be assigned to the community of the atomic services that they are most similar to that in the SSN. Finally, for the remainder of the atomic services, we will assign them to a random community.

(ii) Guide the atomic-service-moving process: GUIDA groups atomic services into communities by a series of atomic-service-moving operations. In this work, the atomic-service-moving operation can only happen at atomic services linking to other atomic services in ASDN and belonging to different communities.

In Algorithm 1, we show the flow of GUIDA, where array depend $[\cdot][\cdot]$ stores ψ^{a} of ASDN; array sim $[\cdot][\cdot]$ stores ψ^{s} of SSN; deg $[\cdot]$ is an array storing the degree of atomic services in ASAN, e.g., atomic service i has degree deg $[i]$; community $[\cdot]$ is an array storing the community identifiers for all atomic services, e.g., atomic service i belongs to the community with identifier community $[i]$; bInit$[\cdot]$ is an array with the Boolean type denoting whether atomic service i has been initialized or not; and bVisited$[\cdot]$ is an array with the Boolean type denoting whether node i has been visited or not.

As is shown in Algorithm 1, the most dominant steps of GUIDA are step 32 to step 53 with the loop. Thus, the computational complexity of GUIDA is O $(|V_a|^2)$.

4. Empirical Study

In this section, we performed experiments to investigate the effectiveness of our SCAS approach. For the experiment environment, all the experiments were carried out on a PC at 2.6 GHz with 8 GB of RAM.

4.1. Research Questions. We performed experiments with the aim to address the following two research questions (RQs):

(i) RQ1: Does our SCAS approach perform better than other *C2S*-based approaches?

We integrate *C2S* and *A3S* together to quantify the similarity between atomic services and use a guided community detection algorithm to group atomic services into clusters. Thus, we wish to know whether our SCAS approach performs better than approaches that only use *C2S* to quantify the similarity between atomic services.

(ii) RQ2: Does our SCAS approach perform better than other semantic-based approaches?

There are many research works on clustering services (see Section 2). Thus, we wish to know whether our SCAS approach performs better than some of these approaches, especially those based on semantic similarities.

4.2. Objects of Study. As mentioned above, PWeb is a famous service repository and widely used as a benchmark data set for experiments of service clustering approaches [1]. PWeb provides the profile of APIs and mashups, including their names, descriptions, tags, and providers. Every mashup is composed of ≥1 API(s) which will be listed in the profile of the mashup. Mashups can be regarded as composite services and APIs can be regarded as atomic services. Thus, PWeb meets the requirement of our SCAS approach and can be used as our object of study.

We crawled the profile of mashups stored in the PWeb till December 14, 2011. Specifically, we crawled the name of mashups and the APIs that mashups used. Figure 4 illustrates the information that we crawled for a mashup.

Specifically, we crawled the name of the mashup (i.e., "Pulse Medic") and the APIs it used (i.e., "Google AdWords," "Healthfinder.gov," and "eduroam"). Finally,

Input:
ASAN, ASDN, and SSN
Output:
Q and all the communities
(1) Initialize depend [·][·], sim [·][·], community [·], bInit[·] = false, and bVisited[·] = false
(2) Calculate the degree of each atomic service in ASAN and store them in deg[ ]
(3) **for** $i = 1$ to $|V_a|$ **do**
(4) $cnt = 0$;
(5) **if** deg [$i > 1$ **then**]
(6) community [·] = cnt++;
(7) bInit [i] = true;
(8) **end if**
(9) **end for**
(10) **for** $i = 1$ to $|V_a|$ **do**
(11) **if** deg[$i == 1$ && !bInit [i] **then**]
(12) MAX = −1;
(13) max Index = −1;
(14) **for** $j = 1$ to $|V_a|$ **do**
(15) **if** sim[i] [j] > MAX **then**
(16) MAX = sim[i] [j];
(17) max Index = j;
(18) **end if**
(19) **end for**
(20) **if** MAX != −1 || $maxIndex$!= −1 **then**
(21) community [·] = community [·];
(22) bInit [i] = true;
(23) cnt++;
(24) **end if**
(25) **end if**
(26) **if** $cnt == |V_a|$ **then**
(27) break;
(28) **end if**
(29) $i = 1$;
(30) **end for**
(31) Calculate the initial Q according to equation (10)
(32) **for** $i = 1$ to $|V_a|$ **do**
(33) **for** $j = 1$ to $|V_a|$ && j! = i **do**
(34) **if** depend[j [i] $\geq$ 1 && community[j] $\neq$ community[i] && !bVisited[i] **then**
(35) Suppose we move atomic service i to community community[j]
(36) Update community[·]
(37) Calculate Q according to equation (10). Denote it as Q_t
(38) Store $\Delta Q = Q_t - Q$ into an array ΔQ[ ]
(39) **end if**
(40) **end for**
(41) bVisited [i] = true
(42) Select the maximum ΔQ. Denote it as ΔQ_{max}
(43) **if** $\Delta Q_{max} > 0$ **then**
(44) Move atomic service i to community community[j] that produces the largest ΔQ
(45) **for** $m = 1$ to $|V_a|$ **do**
(46) **if** depend[i [m] == 1 **then**]
(47) bVisited[m] = false
(48) **end if**
(49) **end for**
(50) $i = 1$
(51) $Q = \Delta Q + Q$
(52) **end if**
(53) **end for**
(54) **return** Q and all the communities

Algorithm 1: GUIDA algorithm.

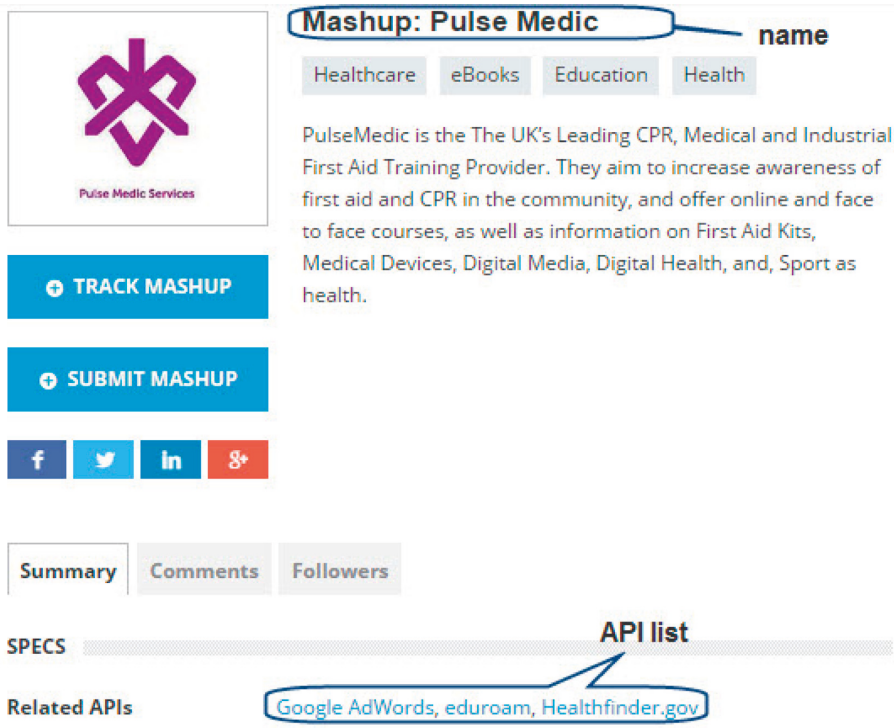


Figure 4: The information that we crawled for a mashup.

our data set for experiments contains 6,362 mashups and 982 APIs.

4.3. Baseline Approaches. The purpose of this work is to improve the performance of service clustering approaches that are based on structure-related metrics. Thus, our SCAS will be compared with the approach using *C2S*, which is named as CSCA (*C2S*-based service clustering approach). CSCA uses *C2S* to quantify the similarity between atomic services and uses GUIDA to group atomic services into clusters.

We cannot make a thorough comparison with the approaches that we have reviewed in Section 2. The main reason is that we cannot implement their approaches since their work does not report the details of their approaches. But, we try our best to compare our approach with the approaches that we reviewed in Section 2. In this work, we compare our approach with the following two approaches in the related work, which are originally proposed to cluster Web services.

(i) TCluster: it uses the information of tags to quantify the semantic similarity between APIs by using LDA and further applies k-means to organize APIs into clusters.

(ii) DTCluster: DTCluster is very similar to TCluster. The only difference is DTCluster also utilizes the information of the description.

4.4. Experiment Process and Results. We follow the main procedures shown in Figure 1 to parse the data set, build the ASAN, ASDN, and SSN, compute *C2S*, *A3S*, and *A2S*, and apply GUIDA group APIs into clusters.

Figures 5(a) and 5(b) show the ASAN and ASDN we constructed from PWeb, respectively. The position of the nodes in ASAN and ASDN is all computed using the circular algorithm in Pajek (Pajek: http://vlado.fmf.uni-lj.si/pub/networks/pajek/).

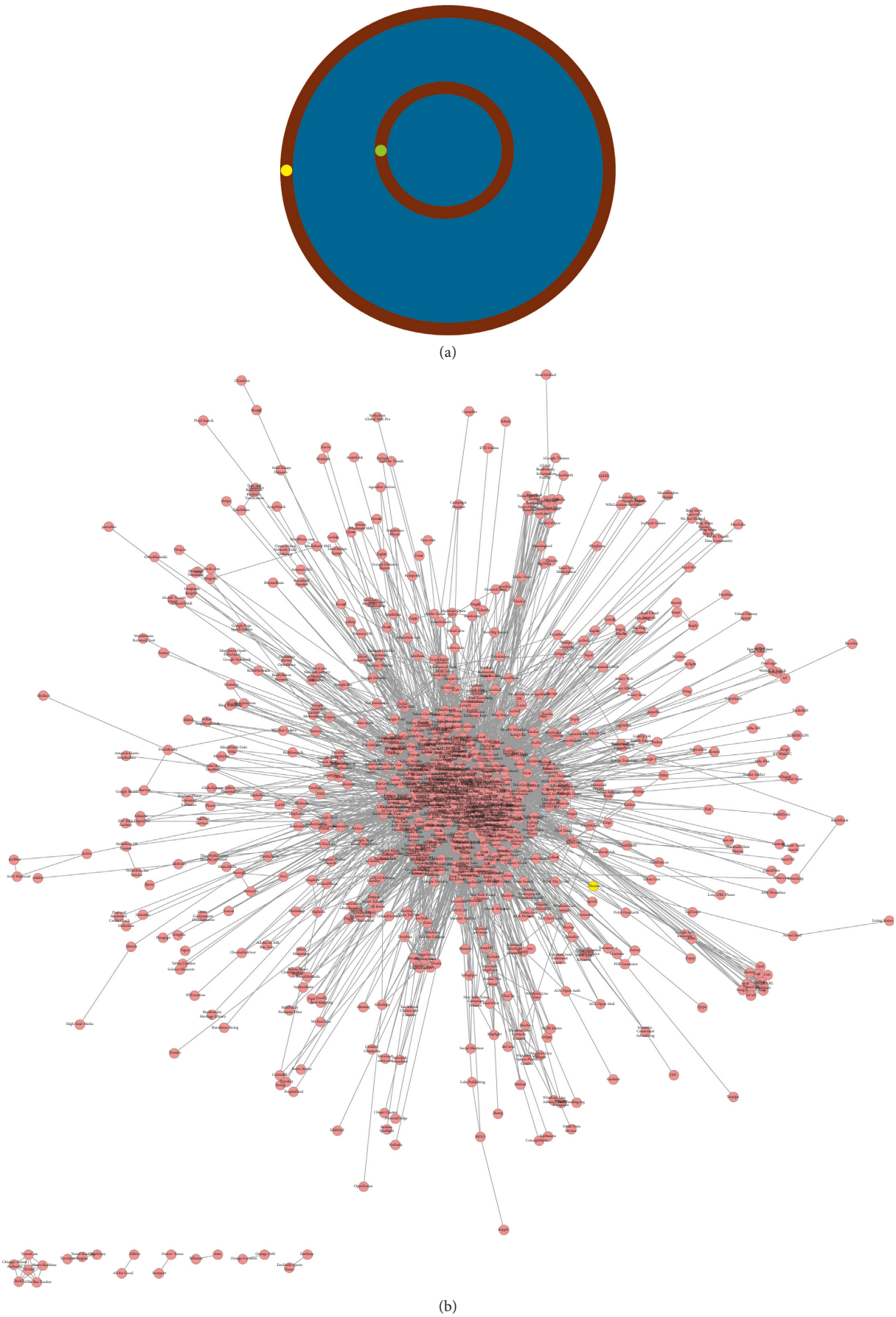
Based on ASAN and ASDN, we compute the *C2S* and *A3S* similarity metrics between APIs so as to compute the *A2S* to quantify API similarity as a whole. Then, we can use GUIDA to group APIs into clusters. In the experiment, GUIDA returns the communities in the SSN when it terminates at $Q = 0.521102$.

4.5. Analysis of the Results. In this section, we analyze the obtained service clustering results with the aim to answer the research questions that we have presented in Section 4.1. To compare SCAS with CSCA, we should measure the quality of their clustering solutions.

In this work, we apply a set of criteria that are widely used in information retrieval, i.e., Precision, Recall, and F-Measure (i.e., F_1 and F_5) [1]. To compute the value of these criteria, we use the classification system in PWeb as the standard clustering results, i.e., APIs in the same category are viewed as APIs in the same cluster. These criteria can be defined as follows:

(i) *True Positive (TP)*. A *TP* decision assigns two similar atomic services to the same cluster

(ii) *False Positive (FP)*. A *FP* decision assigns two dissimilar atomic services to the same cluster



(a)

(b)

Figure 5: (a) ASAN and (b) ASDN constructed from PWeb.

(iii) *False Negative (FN)*. A *FN* decision assigns two similar atomic services to different clusters

(iv) *True Negative (TN)*. A *TN* decision assigns two dissimilar atomic services to different clusters

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP + FP}, \\ \text{Recall} &= \frac{TP}{TP + FN}, \\ F_1 &= \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}, \\ F_5 &= \frac{26 * \text{Precision} * \text{Recall}}{25 * \text{Precision} + \text{Recall}}. \end{aligned} \tag{11}$$

Generally, a larger value of Precision, Recall, and *F*-Measure indicates a better service clustering solution [17–19].

4.5.1. Does Our SCAS Approach Perform Better than Other C2S-Based Approaches? SCAS combined *C2S* and *A3S* to quantify the similarity between atomic services while CSCA applied *C2S*. The only difference between SCAS and CSCA is the similarity metrics that they used. Thus, by comparing the results obtained by SCAS with those of CSCA, we can know whether SCAS performs better than CSCA.

Figure 6 shows the performance comparisons of the two approaches, SCAS and CSCA, when applied to the data set. Obviously, SCAS outperforms CSCA with respect to *Precision*, *Recall*, F_1, and F_5. It indicates that the combination of *C2S* and *A3S* can improve the performance of a service clustering approach.

4.5.2. Does Our SCAS Approach Perform Better than Other Semantic-Based Approaches? As mentioned in Section 2, there are many research works on clustering services. In this section, we performed experiments to check whether our SCAS approach performs better than some of these approaches which are based on semantic similarities, i.e., TCluster and DTCluster. As mentioned above, the only difference between SCAS and TCluster (or DTCluster) is the former applied structure-based similarity metrics while the latter applied semantic-based similarity metrics. Thus, by comparing the results obtained by SCAS with those of TCluster (or DTCluster), we can know whether structure-based similarities are better than semantic-based similarities in service clustering.

Figure 7 shows the performance comparisons of the three approaches, SCAS, TCluster, and DTCluster, when applied to the data set. Obviously, SCAS outperforms TCluster and DTCluster with respect to Precision, Recall, F_1, and F_5. It indicates that structure-based similarity metrics are better than semantic-based similarities in service clustering.

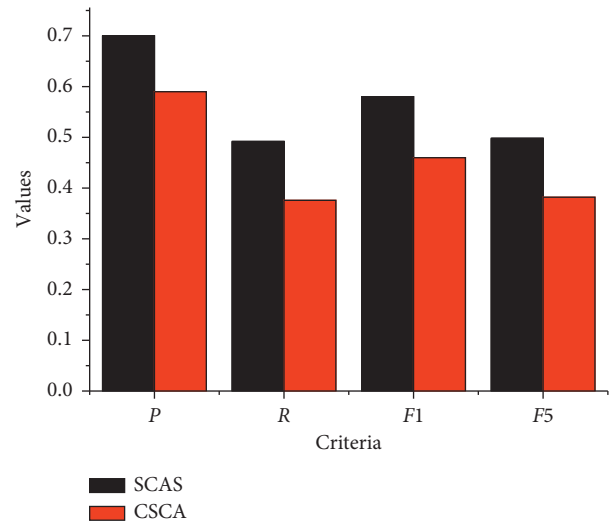


Figure 6: Performance comparison of SCAS and CSCA. P, *Precision*; R, *Recall*; *F*1 and *F*5 denote F_1 and F_5, respectively.

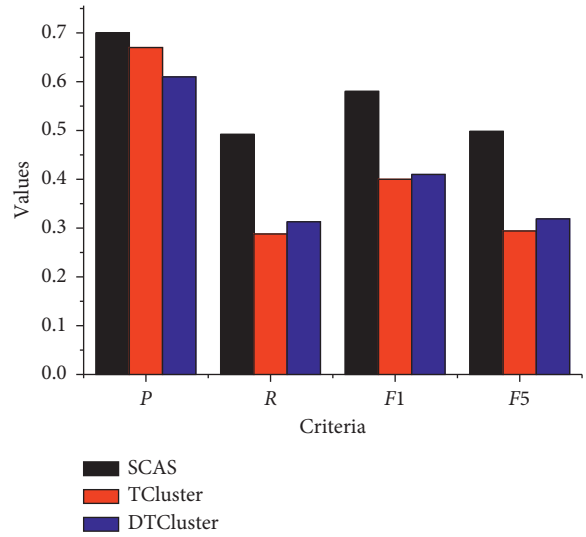


Figure 7: Performance comparison of SCAS and TCluster (or DTCluster). P, Precision; *R*, Recall; *F*1 and *F*5 denote F_1 and F_5, respectively.

5. Conclusions

In this work, we proposed a SCAS approach to group services into different clusters. It proposed an improved structure-related metric, *A2S*, to quantify service similarity and proposed a guided community detection algorithm to organize services into clusters. Comparative studies with other related approaches on a real-world data set show that SCAS performs better than some of the existing approaches.

Data Availability

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

Conflicts of Interest

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

Acknowledgments

This work was supported by the National Natural Science Foundation of China (no. 61375053).

References

[1] W. Pan and C. Chai, "Structure-aware Mashup service clustering for cloud-based Internet of Things using genetic algorithm based clustering algorithm," *Future Generation Computer Systems*, vol. 87, pp. 267–277, 2018.

[2] W. Pan, J. Dong, K. Liu, and J. Wang, "Topology and topic-aware service clustering," *International Journal of Web Services Research*, vol. 15, no. 3, pp. 18–37, 2018.

[3] B. Jiang, L. Y. Ye, W. F. Pan, and J. L. Wang, "Service clustering based on the functional semantics of requirements," *Chinese Journal of Computer*, vol. 41, no. 6, pp. 1255–1266, 2018.

[4] W. Liu and W. Wong, "Web service clustering using text mining techniques," *International Journal of Agent-Oriented Software Engineering*, vol. 3, no. 1, pp. 6–26, 2009.

[5] L.-J. Zhang and J. Zhang, "Service oriented solution modeling and variation propagation analysis based on architectural building blocks," *International Journal of Web Services Research*, vol. 10, no. 4, pp. 39–61, 2013.

[6] K. Elgazzar, A. E. Hassan, and P. Martine, "Clustering WSDL documents to bootstrap the discovery of Web services," in *Proceedings of the 8th IEEE International Conference on Web Services (ICWS 2010)*, pp. 147–154, Miami, FL, USA, July 2010.

[7] L. Chen, L. K. Hu, Z. B. Zheng et al., "WTcluster: Utilizing tags for Web services clustering," in *Proceedings of the 9th International Conference on Service-Oriented Computing (ICSOC 2011)*, pp. 204–218, Paphos, Cyprus, December 2011.

[8] L. Chen, Y. Wang, Q. Yu, Z. Zheng, and J. Wu, "WT-LDA: User tagging augmented LDA for Web service clustering," in *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*, pp. 162–218, Berlin, Germany, December 2013.

[9] M. Shi, J. X. Liu, D. Zhou, M. D. Tang, and B. Q. Cao, "WE-LDA: A word embeddings augmented LDA model for Web services clustering," in *Proceedings of the 15th IEEE International Conference on Web Services (ICWS 2017)*, pp. 9–16, Honolulu, HI, USA, June 2017.

[10] X. Liu, S. Agarwal, C. Ding, and Q. Yu, "An LDA-SVM active learning framework for Web service classification," in *Proceedings of the 14th IEEE International Conference on Web Services (ICWS 2016)*, pp. 49–56, San Francisco, CA, USA, July 2016.

[11] B. Q. Cao, "Mashup service clustering based on an integration of service content and network via exploiting a two-level topic model," in *Proceedings of the 14th IEEE International Conference on Web Services (ICWS 2016)*, pp. 212–219, San Francisco, USA, July 2016.

[12] J. X. Liu, K. Q. He, J. Wang, and D. Ning, "A clustering method for web service discovery," in *Proceedings of the International Conference on Service Computing (SCC 2011)*, pp. 729-730, Washington, DC, USA, July 2011.

[13] J. Zhou and S. Li, "Semantic web service discovery approach using service clustering," in *Proceedings of the International Conference on Information Engineering and Computer Science (ICIECS 2009)*, pp. 1–5, Wuhan, China, December 2009.

[14] F. Chen, S. Yuan, and B. Mu, "User-QoS-based Web service clustering for QoS prediction," in *Proceedings of the 13rd IEEE International Conference on Web Services (ICWS 2015)*, pp. 583–590, New York, USA, July 2015.

[15] N. Zhang, J. Wang, and Y. T. Ma, "Mining domain knowledge on service goals from textual service descriptions," *IEEE Transactions on Services Computing*, 2017.

[16] R. B., Xiong, J. Wang, N. Zhang, and Y. T. Ma, "Deep hybrid collaborative filtering for Web service recommendation," *Expert Systems with Applications*, vol. 110, pp. 191–205, 2018.

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

[18] W. Pan, B. Hu, J. Dong, K. Liu, and B. Jiang, "Structural properties of multilayer software networks: a case study in Tomcat," *Advances in Complex Systems*, vol. 21, no. 2, Article ID 1850004, 2018.

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

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

[21] Y. Xiang, W. Pan, H. Jiang, Y. Zhu, and H. Li, "Measuring software modularity based on software networks," *Entropy*, vol. 21, no. 4, p. 344, 2019.

[22] W. Pan and C. Chai, "Measuring software stability based on complex networks in software," *Cluster Computing*, vol. 22, no. S2, pp. 2589–2598, 2019.

[23] W. F. Pan, H. B. Jiang, H. Ming, C. L. Chai, B. Chen, and H. Li, "Characterizing software stability via change propagation simulation," *Complexity*, vol. 2019, Article ID 9414162, 17 pages, 2019.

[24] J. T. McClave, P. G. Benson, and T. Sincich, *Statistics for Business and Economics*, Prentice Hall, Upper Saddle River, NJ, USA, 2008.

[25] M. E. J. Newman, "Fast algorithm for detecting community structure in networks," *Physical Review*, vol. 69, no. 6, Article ID 066133, 2004.