Modeling and Exploring Historical Records to Facilitate Service Composition

Jian Wu, Liang Chen*, Yanan Xie, Lichuan Ji and Zhaohui Wu

College of Computer Science,
Zhejiang University, Hangzhou, China
Email:{wujian2000, cliang, xyn, jilichuan, wzh}@zju.edu.cn
*Corresponding author

Abstract: Along with a continuously growing number of Web services, how to locate appropriate Web services to complete the task of service composition is becoming more critical. Differing from most recent studies which mainly focus on functional and non-functional properties, we mine nuggets from the Historical Service-composition Dataset (HSD), which carries related users' past experiences. In this paper, a graph mining based recommendation approach is presented to facilitate the process of service composition. In particular, we first extend the graph mining approach *gSpan* to recognize Frequently Used Web Services (FUWS) with their connecting structures from HSD. Then, according to the records in HSD, which share the same FUWSs with user's partially composed service, a bunch of Web services with higher probability is recommended automatically as candidates. Finally, the *skyline* approach is adopted for optimal composite service selection with consideration of overall quality of services (QoS). Furthermore, experiments based on 1,530 real Web services demonstrate the effectiveness and efficiency of our approach.

Keywords: web service, recommendation, graph mining, QoS, skyline

Reference to this paper should be made as follows: Jian Wu, Liang Chen, Yanan Xie, Lichuan Ji and Zhaohui Wu (2013) 'Modeling and Exploring Historical Records to Facilitate Service Composition', *Int. J. Web and Grid Services*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Jian Wu received his B.S. and Ph.D. Degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently an associate professor at the College of Computer Science, Zhejiang University, and visiting professor at University of Illinois at Urbana-Champaign. His research interests include service computing and data mining. He is the recipient of the second grade prize of the National Science Progress Award. He is currently leading some research projects supported by National Natural Science Foundation of China and National High-tech R&D Program of China (863 Program).

Liang Chen received the B.S. degree in computer science from Zhejiang University, Hangzhou, China, in 2009. He is currently working toward the Ph.D. degree in the College of Computer Science, Zhejiang University. His publications have appeared in some well-known conference proceedings and international journals. He also served as a Reviewer for some international

conferences and journals (CIKM, PAKDD, TSMC, TSC, ICWS, ICSOC, ISSRE, etc). His research interests include service computing and data mining. He received the award of ”Excellent Intern” from Microsoft Research Asia in 2010.

Yanan Xie is currently an undergraduate student in Qiushi Science Class, Chukochen Honors College, Zhejiang University. His publications have appeared in some well-known international conference proceedings and journals (WWW, PAKDD, etc.). His research interests include service computing and recommender system.

Lichuan Ji received the B.S. degree in information and computer science from Huazhong Agricultural University, Wuhan, China, in 2012. He is currently working toward the Ph.D. degree in the College of Computer Science, Zhejiang University. He is a ACM student member and his research interests include recommender system, service computing and topic model.

Zhaohui Wu received the Ph.D. degree in computer science from Zhejiang University, Hangzhou, China, in 1993. From 1991 to 1993, he was a joint Ph.D. student in the area of knowledge representation and expert system with the German Research Center for Artificial Intelligence, Kaiserslautern, Germany. He is currently a Professor at the College of Computer Science, and the Director of the Institute of Computer System and Architecture, Zhejiang University. His research interests include intelligent transportation systems, distributed artificial intelligence, semantic grid, and ubiquitous embedded systems. He is the author of four books and more than 100 referred papers. Dr. Wu is a Standing Council Member of China Computer Federation (CCF), Beijing. Since June 2005, he has been the Vice Chairman of the CCF Pervasive Computing Committee. He is also on the editorial boards of several journals and has served as a Program Chair for various international conferences.

1 Introduction

A service-oriented computing (SOC) paradigm and its realization through standardized Web service technologies provide a promising solution for the seamless integration of single-function applications to create new large-granularity and value-added services. Web service composition attracts industry’s attention and is applied in many domains, e.g., workflow management, finances, e-Business, and e-Science. With the increase of diverse and dynamic demands of users, traditional composition schema and existing composite services cannot satisfy users’ demands.

Traditional service composition schema has three main drawbacks: 1) Some professional domain knowledge is essential to select simplex services for composition; 2) Knowledge about service description language is needed for composing services to satisfy users’ demands; 3) Manually selecting simplex services is a time-consuming job. Although the appearance of some service composition tools such as *Eclipse BPEL Designer* can overcome the second drawback, professional domain knowledge is still a roadblock for non-professional users to do Web service composition. Furthermore, the reliability of composite service generated by traditional service composition schema can not be guaranteed.

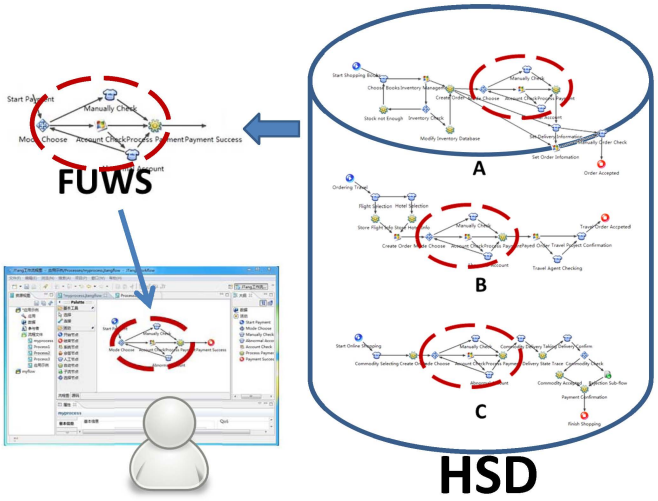


Figure 1 A Service Composition Scenario

Previous works are less aware of the potential value and the hidden knowledge of the service usage data for assisting users with their service composition task. Basically, Frequently Used Web Services (**FUWS**) have the following advantages: They carry users' previous experiences, have higher probability to fulfill users' requirements, and have been proved to be more reliable and robust. Furthermore, composite services that share the same **FUWS** may have similar composition process or functionality. By reusing such kind of verified knowledge, it will dramatically increase the correctness and rationality of composed services, and accelerate the rapid application development. Given the scenario as Fig. 1 shows, the user does the task of service composition by using a designer tool. By analyzing the historical composition records in **HSD**, some similar **FUWS**s could be recommended to the user as candidates for selection according to the user's partially composed service. Further, if the user's partially composed service share the same **FUWS** with composite services in **HSD**, they may have similar composition process in the forwarding step, which could also be recommended to the user to facilitate the composition task.

To handle the above scenario, we propose a graph mining based approach to model and explore the hidden knowledge from Historical Service-composition Dataset. First of all, three hypotheses are formulated as follows: 1) the experiences of domain experts can be implied in the structure of the composed Web services and their execution logs; 2) these Web services that have been previously composed or executed together tend to be reused in a similar way; 3) the more frequently used Web services, the better reliability and robustness.

More specifically, our approach includes the following: We model an executed service transaction or a bunch of Web services which compose together and register in service repository by using directed labeled graph. We adopt and extend the graph mining approach *gSpan* (Yan & Han (2002)) to identify **FUWS**s among those directed labeled graphs. Then, we explore and locate those Web services that share the same **FUWS**s with the user's partially composed services. After that, we propose a model

to estimate how likely the service share the same **FUWS**s will meet the user's needs. Based on the likelihood, we recommend some Web services as candidates for user's composition task, and then select the optimal ones with consideration of overall QoS by using *skyline* (S.Borzsonyi et al. (2001)) approach.

Technically, to address the issue of frequent subgraph mining (e.g., **FUWS** identification), we adopt and extend *gSpan* to discover frequent substructures efficiently without costly subgraph isomorphism test and costly candidate generation. To our best knowledge, this work is the first attempt to exploit such efficient frequent structured patterns mining techniques to identify **FUWS**s with their connecting structure, which could provide instant and smart suggestions to facilitate Web services composition.

In particular, the main contributions of this paper can be summarized as follows:

1. We formulate the problem of **FUWS** identification by modeling historical service composition records, and extend an existing algorithm *gSpan* to discover frequent substructures efficiently.
2. Based on **FUWS**, we propose a service recommendation to help complete the composition task with the consideration of both composition structure and QoS.
3. Experiments based on 1,530 real Web services demonstrate the effectiveness and efficiency of the proposed approach.

The rest of this paper is organized as follows. A discussion of the related work is presented in Section 2. The framework of graph mining based service recommendation for composition is described in Section 3. Section 4 introduces the data preparation stage and data processing stage, while Section 5 gives a detailed description about service recommendation stage. Experimental evaluation results are presented in Section 6. Finally, we conclude the paper and introduces our future work.

2 Related Work

The problem of recommending Web service for composition has been extensively studied in recent years. Most of the recent works are based on *QoS*. For example, Thio & Karunasekera (2005) propose a mechanism involving automated measurement of *QoS* attributes for Web service recommendation. Moraru et al. (2009) introduce a hybrid system which combines semantic technology and logic based reasoning with numerical calculus to classify, evaluate and then recommend *QoS* aware WSs. Li et al. (2010) propose a reasonable web service mechanism which allows service recommendation based on the multi-QoS constraints. Taniar & Goh (2007) propose a movement pattern mining approach to handle the scenario of mobile computing.

Another major direction of recommending Web services for composition is inspired by the idea of Collaborative Filtering (CF). Zheng et al. (2011) present a CF-based approach to predict the missing *QoS* values and recommend appropriate Web services for composition by taking advantages of past usage experiences. Chen et al. (2010) present a novel hybrid CF-based algorithm, *RegionKNN*, for large scale Web service recommendation with consideration location. Different from other approaches, *RegionKNN* will build an efficient region model based on the users' physical locations and historical *QoS* similarities. Besides, there are also other methods about CF-based service recommendation for composition. In our previous work Wu et al. (2012), we

propose a Bayes theorem based service recommendation approach to help facilitate the process of service composition.

Differing from above recent studies, we propose to facilitate the process of service composition by modeling and exploring the hidden knowledge from historical service composition records. In particular, graph mining based approaches are proposed to identify the **FUWS**s which are recommended to the user as candidates for composition.

Graph mining is an important research direction in data mining area. Basically, graph mining has been widely used for finding frequent common structures. For example, (Takajashi et al. (1987), Bayada et al. (1992), Dehaspe et al. (1998)) study the problem of finding frequent common substructures for chemical compounds. Holder et al. (1994) describe the SUBDUE system, which uses the minimum description length (MDL) principle to discover substructures that compress the database and represent structural concepts in the data. However, these methods cannot identify the complete set of frequent substructures.

Furthermore, Inokuchi et al. (2000) propose the first approach of mining the complete set of frequent graph patterns, named *AGM*. Then Kuramochi & Karypis (2001) present a computationally efficient algorithm, named *FSG*, for finding all frequent subgraphs in large graph databases. Both *AGM* and *FSG* are Apriori-based algorithms and adopt breadth-first search.

After that, Yan & Han (2002) propose a novel algorithm, *gSpan*, which adopts the depth-first search strategy to mine frequent connected subgraphs efficiently. They also develop an algorithm, *CloseGraph* (Yan & Han (2003)), for closed graph pattern mining. Then, Yan et al. (2004) make use of frequent substructure to index graphs. Subsequently, Wang et al. (2004) develop an effective index structure, *ADI*, to support mining various graph patterns over large databases that cannot be held into main memory.

Moreover, there are some studies on mining frequent nested subgraphs in a single graph including *HSIGRAM* and *VSIGRAM* Kuramochi & Karypis (2005). Meanwhile, some more efficient algorithms, such as, *CODENSE* (Hu et al. (2005)), *CROCHET* (Pei et al. (2005)), *CLAN* (Wang et al. (2006)) and *COCAIN* (Zeng et al. (2007)) were proposed to mine frequent coherent dense subgraphs.

3 Framework of Service Recommendation Based on Graph Mining

Web service recommendation can accelerate and facilitate the process of service composition. Suppose a user is composing some Web services, our objective is to understand the intent of the user based on the semi-finished (i.e., partially composed) composite Web service, so as to recommend appropriate Web services to the user to help them finish the composition. Moreover, to ensure the effectiveness of the recommendation, we try to enlarge the probability that users select the recommended extensions.

In this paper, we propose a novel graph mining based service recommendation framework to help facilitate service composition. Generally, there are three main stages in the proposed framework: *Data Preparation Stage*, *Data Processing Stage* and *Service Recommendation Stage*, as illustrated in Fig.2.

1. **Data Preparation Stage**: In this stage, Web services with the same or similar functionality are clustered and marked with the same class label. Then the

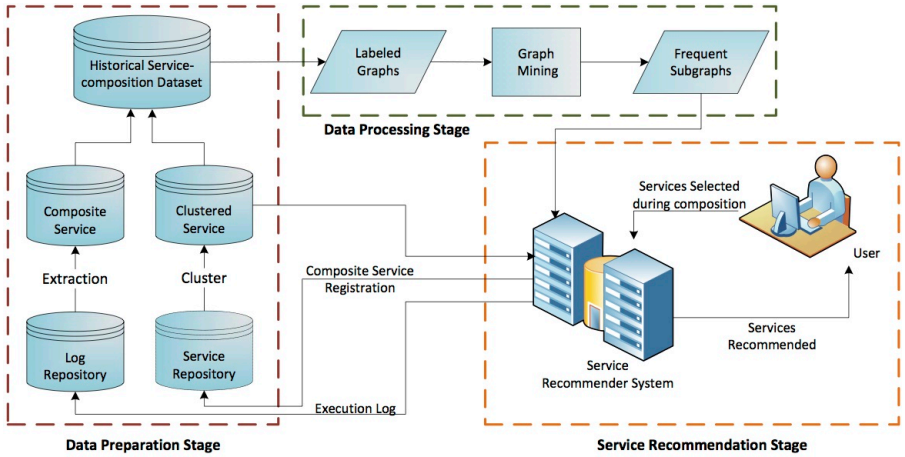


Figure 2 The process for service recommendation based on graph mining

executed composite Web services (i.e., historical composition records) are modeled into directed labeled graphs. Details will be given in Section 4.1.

2. **Data Processing Stage**: In this stage, we utilize graph mining algorithm to identify frequent subgraphs (i.e., **FUWS**), representing the key components of Web services that are often executed together. Details will be given in Section 4.2.

3. **Service Recommendation Stage**: We propose a model to estimate the probability that a directed labeled graph (e.g., a composition of Web services) will meet the user's needs, and then select the optimal composite services with consideration of overall QoS using skyline. Details will be given in Section 5.

4 Data Preparation and Processing Stages

Data preparation stage and data processing stage are the bases for service recommendation stage. In this section, we will give a detailed description of data preparation stage and data processing stage.

4.1 Data Preparation Stage

Data preparation stage can be mainly divided into two steps: 1) clustering Web services and 2) modeling historical composition records with directed labeled graphs.

Before identifying FUWSs, we need to cluster similar Web services, since the usage data are very sparse, and the clustering results can help to provide diverse candidates to users. There are many approaches for Web service clustering, like Elgazzar et al. (2010), Liang et al. (2009), Nayak & Lee (2007), Skoutas et al. (2010). In this paper, we use K-medoids Clustering algorithm to group Web services with the same function together. It should be noted that we utilize the modified version of the approach in Elgazzar et

al. (2010) to calculate service similarity while clustering, and the details are given in Section 6.1. The definition of clustered Web service is as follows.

Definition 1 Clustered Web Service. *After service clustering, a Web service can be formally defined by a 5-tuple, WS=(WS-ID, In, Out, QoS, Cluster-ID), where*

- *WS-ID is the unique identifier of the Web service;*
- *In/Out is the input/output interface;*
- *QoS is the quality of service;*
- *Cluster-ID is the unique identifier of the cluster which the Web service belongs to and all the Web services belonging to the same cluster have the same Cluster-ID.*

In most situations, a single Web service cannot meet users' needs, and service composition is inevitable. Once a composite Web service is invoked by a user, we can construct a corresponding graph for this composite Web service, where the links are built according to the execution order between component Web services. For the sake of simplicity, we focus on directed acyclic graphs in this paper. The formal definitions of a composite Web service and composite service graph are given as follows:

Definition 2 Composite Web Service. *A composite Web service can be represented by a 2-tuple $CS = (WSS, WSR)$, where*

- *WSS is the set of component Web services included in this composite Web service;*
- *$WSR \subseteq WSS \times WSS$ represents the relation between two Web services, i.e., $(WS_i, WS_j) \in WSR$ when WS_j is executed on the neck of WS_i.*

Definition 3 Directed Labeled Graph for Composite Service. *A directed labeled graph for composite service can be represented by a 4-tuple $G = (V, E, L_V, l_V)$, where*

- *V is the vertex set of the graph, where each vertex refers to a component Web service;*
- *$E \subseteq V \times V$ is the edge set of the graph, where each edge represents the execution order between two Web services;*
- *L_V is the label set for the vertex set, which is comprised by Cluster-IDs;*
- *l_V is a function mapping from a vertex to a label. Because each vertex represents a Web service and each Web service has a Cluster-ID, each vertex's label is just the corresponding Cluster-ID.*

For example, Fig. 3(a) gives a sample of composite Web service invoked by the user, and Fig. 3(b) shows *Cluster-ID* of each Web service in Fig. 3(a), then we can construct the corresponding composite service graph as Fig. 3(c) shows. In particular, *WS2* and *WS3* are in the same cluster *C2*.

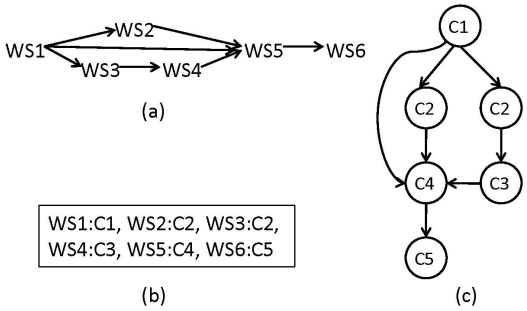


Figure 3 Construction of composite service graph

4.2 Data Processing Stage

The data processing stage is responsible for the acquisition of Web service clusters which are often executed together. To achieve this goal, we utilize graph mining algorithms to mine frequent subgraphs from composite service graphs.

4.2.1 Problem Definition

Before introducing the proposed method, we first define the subgraph isomorphism for composite service graphs.

Definition 4 Subgraph Isomorphism for Composite Service. *Given two composite service graphs* $G = (V, E, L_V, l_V)$ *and* $G' = (V', E', L'_V, l'_V)$. *A subgraph isomorphism from* G *to* G' *is an injective function* $f : V \to V'$, *such that*

1. *for any vertex* $u \in V$, $l_V(u) = l'_V(f(u))$, *i.e., the two corresponding Web services represented by* u *and* $f(u)$ *belong to the same cluster, and*
2. *for any edge* $(u, v) \in E$, $(f(u), f(v)) \in E'$.

If there exists a subgraph isomorphism from G to G', then G is a subgraph of G' and G' is a supergraph of G, denoted as $G \sqsubseteq G'$.

Based on the definition of subgraph isomorphism, we can give the definition of frequent subgraph mining as follows.

Definition 5 Frequent Subgraph Mining for Composite Service. *Given a composite service graph dataset* $D = \{G_1, G_2, \ldots, G_n\}$, *and a minimum support* min_sup, *assume* $sup(g)$ *denotes the support of* g *in* D, *then*

$$sup(g) = \sum_{G_i \in D} I(g, G_i), 1 \leq i \leq n, \textit{where}$$

$$I(g, G) = \begin{cases} 1, \textit{if } g \textit{ is a subgraph of } G; \\ 0, \textit{if } g \textit{ is not a subgraph of } G. \end{cases}$$

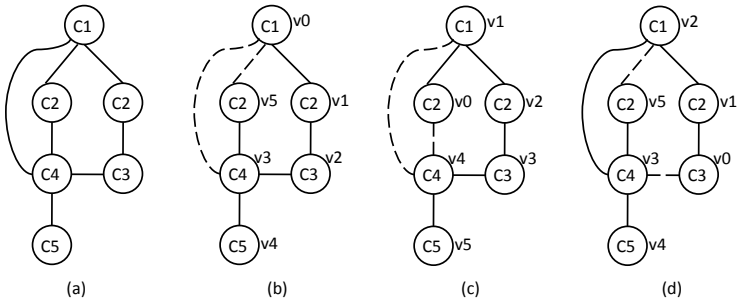


Figure 4 Depth-first search tree

A subgraph g is called a frequent subgraph pattern if $sup(g) \geq min\_sup$.

The objective of frequent Subgraph Mining for Composite Service is to find the complete set of subgraphs that are frequently used in the composite service graph dataset D.

4.2.2 Lexicographic Ordering

Yan & Han (2002) developed a lexicographic order for graphs to facilitate the graph mining process. Based on the lexicographic order, they proposed an efficient graph mining algorithm, *gSpan*, which is focused on undirected labeled simple graph.

In this paper, we extend and modify *gSpan* to make sure than it can be used for composite service graph mining. First, we transform a composite service graph G to the corresponding undirected labeled graph $\bar{G}$. Then we construct a Depth-First Search (DFS) tree by performing a depth-first search (Cormen et al. (2002)) in $\bar{G}$. The depth-first discovery of the vertices forms a linear order. Then, we use subscripts to label this linear order according to their discovery time (Cormen et al. (2002)). v_i is discovered before v_j iff $i < j$. v_0 is called the root and v_{n-1} is called the rightmost vertex, where n is the size of the vertex set. The straight path from the root v_0 to the rightmost vertex v_{n-1} is called the rightmost path. For example, Fig. 4(a) shows the corresponding undirected labeled graph of the composite service graph in Fig. 3(c). In Fig. 4(b)-4(d), three different subscriptings are generated for the graph in Fig. 4(a). The rightmost path is $(v_0, v_1, v_2, v_3, v_5)$ in Fig. 4(b), $(v_0, v_1, v_2, v_3, v_4, v_5)$ in Fig. 4(c), $(v_0, v_1, v_2, v_3, v_5)$ in Fig. 4(d). The rightmost vertex and the vertices on the rightmost path play an important role in growing patterns. $\bar{G}$ subscripted with a DFS tree T is denoted as $\bar{G}_T$.

Given $\bar{G}_T$, all edges in the DFS tree belong to forward edge (tree edge, Cormen et al. (2002)) set, and all edges not in the DFS tree belong to backward edge (back edge, Cormen et al. (2002)) set. For example, in Fig. 4(b)-4(d), solid lines represent forward edges, while dotted lines represent backward edges. If we only consider the subscripts of edges, the linear order $\prec_T$ for edges defined in (Yan & Han (2002)) can be described as follows: for all edges in $\bar{G}$ (assume $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$), (1) if $i_1 = i_2$ and $j_1 < j_2$, $e_1 \prec_T e_2$; (2) if $i_1 < j_1$ and $j_1 = i_2$, $e_1 \prec_T e_2$; and (3) if $e_1 \prec_T e_2$ and $e_2 \prec_T e_3$, $e_1 \prec_T e_3$. An edge sequence (e_i) constructed based on $\prec_T$ is called a DFS code, denoted as $code(\bar{G}, T)$. For example, for the DFS tree shown in Fig. 4(b), the corresponding DFS code is $((v_0, v_1), (v_1, v_2), (v_2, v_3), (v_3, v_0), (v_3, v_4), (v_3, v_5), (v_5, v_0))$.

Table 1 DFS codes of Fig. 3(c) based on Fig. 4(b)-4(d)

Edge no.	(b)α	(c)β	(d)γ
0	$(0, 1, +1, C1, C2)$	$(0, 1, -1, C2, C1)$	$(0, 1, -1, C3, C2)$
1	$(1, 2, +1, C2, C3)$	$(1, 2, +1, C1, C2)$	$(1, 2, -1, C2, C1)$
2	$(2, 3, +1, C3, C4)$	$(2, 3, +1, C2, C3)$	$(2, 3, +1, C1, C4)$
3	$(3, 0, -1, C4, C1)$	$(3, 4, +1, C3, C4)$	$(3, 0, -1, C4, C3)$
4	$(3, 4, +1, C4, C5)$	$(4, 0, -1, C4, C2)$	$(3, 4, +1, C4, C5)$
5	$(3, 5, -1, C4, C2)$	$(4, 1, -1, C4, C1)$	$(3, 5, -1, C4, C2)$
6	$(5, 0, -1, C2, C1)$	$(4, 5, +1, C4, C5)$	$(5, 0, -1, C2, C1)$

Then, we can get $code(G, T)$ by representing each edge in $code(\bar{G}, T)$ by a 5-tuple: (i, j, d, l_i, l_j), where l_i and l_j are the labels of v_i and v_j respectively, and d represents the direction of an edge in G. When $d = +1$, the direction is from $i(v_i)$ to $j(v_j)$. When $d = -1$, the direction is from $j(v_j)$ to $i(v_i)$. $code(G, T)$ is the DFS code of the composite service graph G with respect to T. Those DFS codes of the composite service graph in Fig. 3(c) based on the DFS trees in Fig. 4(b)-4(d) are listed in Table 1. Suppose there is a linear order in the label set (L_V) and $(d = +1) \prec_d (d = -1)$, we can build an order on the DFS codes of the same graph. For example, we have $\alpha < \beta < \gamma$ for the DFS codes listed in Table 1.

Given a composite service graph, there can be multiple DFS codes. To solve the uniqueness problem, the minimum DFS code (Yan & Han (2002)) is the canonical label of G.

Minimum DFS code has a nice property: given two graphs G and G', they are isomorphic if and only if $mincode(G) = mincode(G')$, where $mincode(G)$ and $mincode(G')$ are the minimum DFS codes of G and G' respectively. According to the above criteria the problem of mining frequent subgraphs is equivalent to the problem of mining their corresponding minimum DFS codes. Table 1 shows the minimum DFS code among all codes generated by the possible depth-first trees for the composite service graph in Fig. 3(c).

As a composite service graph is directed, we allow the extended edge to have two choices of d when doing a rightmost extension. For example, in Fig. 5, the graph shown in Fig. 5(a) has several potential children with one edge growth, which are shown in Fig. 5(b)-Fig. 5(g) (assume the darkened vertices constitute the rightmost path). In Fig. 5(b), Fig. 5(d) and Fig. 5(f), the extended edges have the choice of $d = +1$, while the extended edges in Fig. 5(c), Fig. 5(e) and Fig. 5(g) have the choice of $d = -1$.

This is the main idea of graph mining. Because each vertex's label is the corresponding Web service's *Cluster-ID*, each frequent subgraph represents these service clusters are often executed together.

5 Service Recommendation Stage

Service recommendation stage is the most important one in the framework. When a user is composing Web services, the objective of our recommendation is to extend the partially composed service by giving some service recommendations or suggestions. Our approach is proposed based on the results of graph mining (i.e., those frequent subgraphs obtained in data processing stage). For each frequent subgraph g, find the

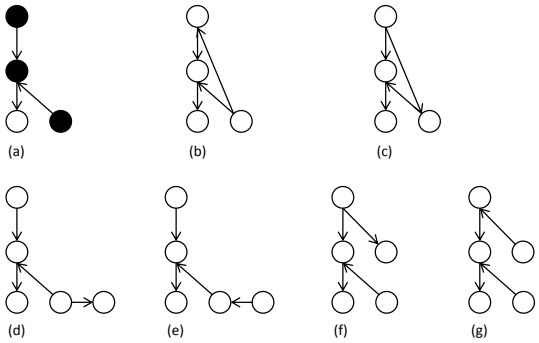


Figure 5 Right-most extension

frequent subgraph g' which is the supergraph of g, such that the probability for g' appearing together with g is larger. That is to say, when the frequent subgraph g appears in the user's partially composed service, we extend it to g', as the probability that the user need g' is quite high. Thus, for the user's partially composed service, we find the frequent subgraphs it contains and extend these frequent subgraphs to those corresponding supergraphs, such that the extensions of the partially composed service are realized. For each extension, we need to replace it with concrete Web services. As there will be many corresponding concrete Web services for each extension, we have to select and compose the optimal composite Web service according to both satisfaction probability and *QoS*.

Thus, this stage can be divided into two steps: 1) determine frequent subgraphs for recommendation; 2) select the optimal composite Web service. These two steps are described in the following subsections.

5.1 Determination of Recommended Frequent Subgraphs

According to the user's partially composed service, we can construct the corresponding directed labeled graph as described in Section 4.1. This directed labeled graph is used as an original graph G_o. We enumerate all subgraphs of G_o to find subgraphs which are in the result set of graph mining. To make it clear, the set comprised of these subgraphs is denoted as S_o, and the result set of graph mining is denoted as FS. Formally, $S_o = \{g | g \sqsubseteq G_0 \text{ and } g \in FS\}$.

For one graph g, if another graph g' can be obtained by adding one edge to g, then g' is the child of g. And the child of g' is the descendant of g. For each graph g in S_o, we enumerate all descendants of g to find those descendants which are also in FS. The set of those descendants with respect to g is denoted as $Des(g)$. Formally, $Des(g) = \{g' | g' \text{ is the descendant of } g \text{ and } g' \in FS\}$.

Then, we need to find the graph in $Des(g)$ with the highest probability that is extended from g. For each graph g' in $Des(g)$, the probability that g' appears when g appears can be denoted as $P(g'|g)$. According to the Bayesian formula, probability $P(g'|g)$ can be transformed as follows:

$$P(g'|g) = \frac{P(g|g') \times P(g')}{P(g)}, \tag{1}$$

where $P(g|g')$ represents the probability that g appears when g' appears. As g' is the descendant of g, g must appears when g' appears. Namely, the value of $P(g|g')$ is 1. Then, Eq. (1) can be transformed as follows:

$$P(g'|g) = \frac{P(g')}{P(g)} = \frac{sup(g')}{sup(g)} \tag{2}$$

According to Eq. (2), the probability $P(g'|g)$ is larger when g' has fewer edges, and we can not get those extensions with more edges. Thus, by introducing the edge count, Eq. (2) is transformed into Eq. (3) whose value is denoted as $Score(g', g)$.

$$Score(g', g) = \frac{(|E(g')| + 1)^{\alpha} \times sup(g')}{(|E(g)| + 1)^{\alpha} \times sup(g)} \tag{3}$$

Next, we need to find the graph in $Des(g)$ with the highest score, and denote it as g_b, which is a candidate for recommendation. The list of candidates for recommendation is denoted as CFS, where each graph g_{ib} is the graph in $Des(g_i)$ whose score is highest. All graphs in CFS will be ranked according to $Score(g_{ib}, g_i)$. The higher the score, the higher the ranking. If scores are equal, we will consider the extended edge count. The extended edge count for g_{ib} is calculated by Eq. (4).

$$N_e(g_{ib}, g_i) = |E(g_{ib})| - |E(g_i)| \tag{4}$$

The larger the extended edge count, the higher the ranking. If extended edge counts are equal, minimum DFS code is considered. The smaller the minimum DFS code, the higher the ranking. The number of candidates for recommendation may be large. We set a parameter k to limit the number of frequent subgraphs recommended for users. If the number of candidates is larger than k, we just return top k in CFS, otherwise, we return all the candidates. The list of recommended frequent subgraphs is denoted as RFS_k. For each graph in FS, its descendant with highest score can be calculated offline, so that we can speed up the online service recommendation stage.

To make it more convenient for the user, the final recommendation is based on the graphs which are acquired by joining each graph in RFS_k with G_o. And the list of these graphs is denoted as RG_k.

For example, Fig. 6(a) shows the user's partially composed service, and Fig. 6(b) shows *Cluster-ID* of each Web service in Fig. 6(a). After that, we can construct the corresponding original graph G_o, which is given in Fig. 6(c). Figure 6(d) shows graphs in S_o. For each graph in Fig. 6(d), its descendant with highest score are given in Fig. 6(e), and the corresponding scores and extended edge counts are given in Fig. 6(f) and Fig. 6(g), respectively. Then the list of recommended frequent subgraphs is given in Fig. 6(h), where each graph has been ranked already. Please note that the value of α and k in this example are 2 and 5, respectively. Figure 7 shows the graphs in RG_5, where the solid lines represent the edges in Fig. 6(c), and the dotted lines represent the edges in corresponding graph in Fig. 6(h).

5.2 Optimal Composite Service Selection

For a graph g in RG_k, the label of each vertex in g is *Cluster-ID*. For the vertices of g which are also in G_o, they are replaced by the corresponding Web services selected

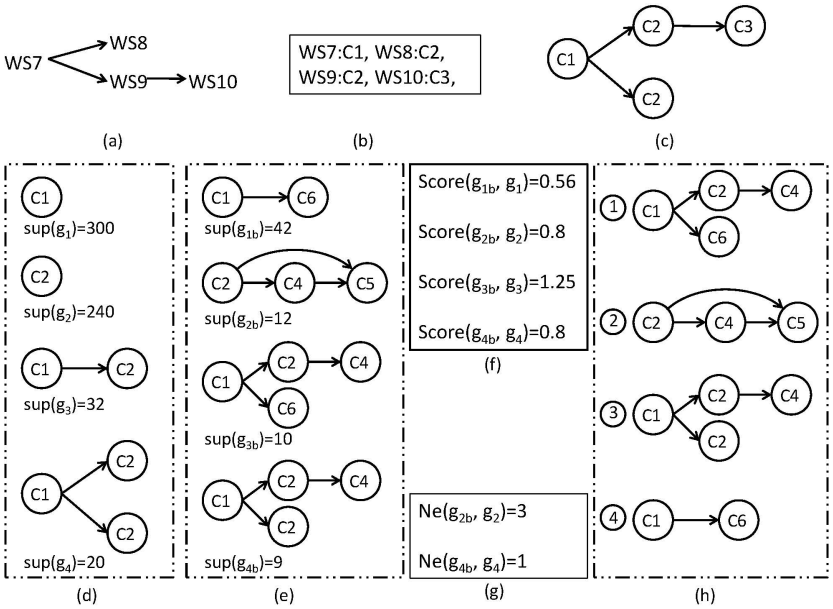


Figure 6 Determination of recommended frequent subgraphs

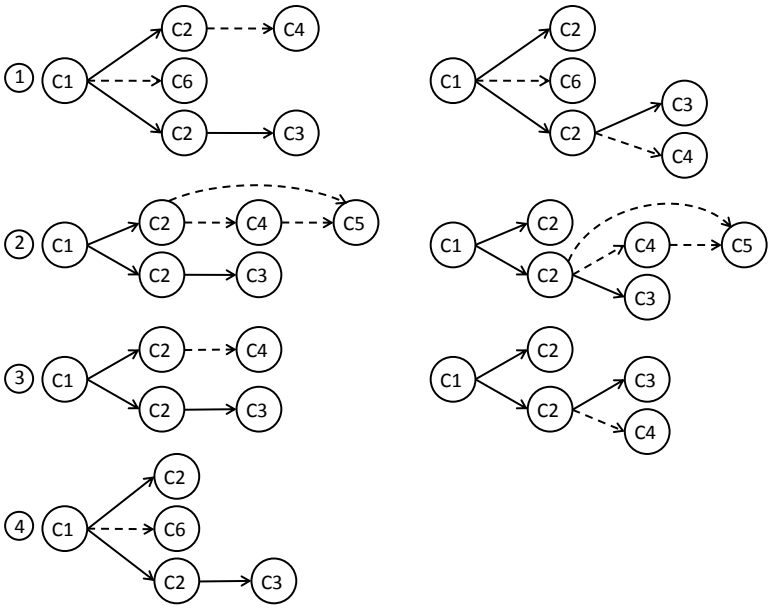


Figure 7 Graphs in RG_5

Table 2 QoS parameters

Parameter Name	Description	Units
Response Time (RT)	Time taken to send a request and receive a response	ms
Availability (AV)	Number of successful invocations/total invocations	%
Throughput (TP)	Total Number of invocations for a given period of time	#/s
Successability (SU)	Number of response / number of request messages	%
Reliability (RE)	Ratio of the number of error messages to total messages	%
Latency (LA)	Time taken for the server to process a given request	ms

by the user already. For other vertices of g, they can be replaced by any Web service in the corresponding cluster. Thus, we can get many composite Web services for g with different replacement. A criterion is needed to measure the quality of composite Web services, so that we can select the optimal composite Web service.

In this paper, we consider six dimensions of *QoS*, which are shown in Table 2. For all the Web services in the service repository, we utilize the function: $y = (x - \min(x))/(\max(x) - \min(x))$ to normalize each dimension of *QoS* into the range $[0, 1]$. Without explicit mentioning, *QoS* discussed below is normalized *QoS*.

For each composite Web service, we can calculate its *QoS* in six dimensions according to its structure and the *QoS* of component Web services. According to Definition 2, we can use $CS = (WSS, WSR)$ to denote one composite Web service. The formulas to calculate CS's *QoS* in six dimensions will be given in the following. Because branch relation in composite Web services is a common relation and its calculation is more complicated, we focus on branch relation in this paper. However, it is not hard to cover other relations in composite Web services.

To compute CS's *QoS* in six dimensions, for each component Web service WS in CS, the number of its direct predecessors is called its *indegree*, denoted as $deg^-(WS)$, and the number of its direct successors is called its *outdegree*, denoted as $deg^+(WS)$. Besides, we add two more Web services, WS_s and WS_f, which represent the initial service and final service, respectively. In addition to the relations in WSR, each Web service whose *indegree* is zero has a relation from WS_s, and each Web service whose *outdegree* is zero has a relation to WS_f. Thus, in CS, WS_s is the only Web service whose *indegree* is zero, and WS_f is the only Web service whose *outdegree* is zero. Be noted that, response time and latency of WS_s and WS_f are 0, and availability, throughput, successability and reliability of WS_s and WS_f are 1.

1) *Response Time for Composite Web Service*: For each component Web service WS in the composite Web service CS, its response time can be denoted as $RT(WS)$. The aggregated response time of WS, denoted as $Agg\_RT(WS)$, is the response time that an execution starting from WS_f will terminate at WS. For a Web service WS_i, $Agg\_RT(WS_i)$ can be defined recursively as follows.

If $deg^+(WS_i) \neq 0$, then $Agg\_RT(WS_i) =$

$$RT(WS_i) + \frac{1}{deg^+(WS_i)} \sum_{j,(WS_i,WS_j)\in WSR} Agg\_RT(WS_j).$$

Otherwise, $Agg\_RT(WS_i) = RT(WS_i)$. Namely, $Agg\_RT(WS_f) = RT(WS_f) = 0$.

Thus, the response time of the composite Web service CS is the aggregated response time of component Web services whose *indegree* is zero. Formally, we use $RT(CS)$ to represent the response time of CS, and it can be calculated as follows:

$$RT(CS) = Agg\_RT(WS_s).$$

2) *Availability for Composite Web Service*: For each component Web service WS in the composite Web service CS, its availability can be denoted as $AV(WS)$. The aggregated availability of WS, denoted as $Agg\_AV(WS)$, is the availability that an execution starting from WS_s will terminate at WS. For a Web service WS_j, $Agg\_AV(WS_j)$ can be defined recursively as follows.

If $deg^-(WS_j) \neq 0$, then $Agg\_AV(WS_j) =$

$$AV(WS_j) \times \sum_{i,(WS_i,WS_j)\in WSR} \frac{1}{deg^+(WS_i)} Agg\_AV(WS_i).$$

Otherwise, $Agg\_AV(WS_j) = AV(WS_j)$. Namely, $Agg\_AV(WS_s) = AV(WS_s) = 1$.

Thus, the availability of the composite Web service CS is the aggregated availability of component Web services whose *outdegree* is zero. Formally, we use $AV(CS)$ to represent the availability of CS, and it can be calculated as follows:

$$AV(CS) = Agg\_AV(WS_f).$$

3) *Throughput for Composite Web Service*: For each component Web service WS in the composite Web service CS, its throughput can be denoted as $TP(WS)$. The aggregated throughput of WS, denoted as $Agg\_TP(WS)$, is the throughput that an execution starting from WS_s will terminate at WS. For a Web service WS_j, $Agg\_TP(WS_j)$ can be defined recursively as follows.

If $deg^-(WS_j) \neq 0$, then

$Agg\_TP(WS_j) = \min(TP(WS_j),$

$$\sum_{i,(WS_i,WS_j)\in WSR} \frac{1}{deg^+(WS_i)} Agg\_TP(WS_i)).$$

Otherwise, $Agg\_TP(WS_j) = TP(WS_j)$. Namely, $Agg\_TP(WS_s) = TP(WS_s) = 1$.

Thus, the throughput of the composite Web service CS is the aggregated throughput of component Web services whose *outdegree* is zero. Formally, we use $TP(CS)$ to represent the throughput of CS, and it can be calculated as follows:

$$TP(CS) = Agg\_TP(WS_f).$$

4) *Successability for Composite Web Service*: For each component Web service WS in the composite Web service CS, its successability can be denoted as $SU(WS)$. The aggregated successability of WS, denoted as $Agg\_SU(WS)$, is the successability that an execution starting from WS_s will terminate at WS. For a Web service WS_j, the method for calculating $Agg\_SU(WS_j)$ is similar to $Agg\_AV(WS_j)$. It can be defined recursively as follows.

If $deg^-(WS_j) \neq 0$, then $Agg\_SU(WS_j) =$

$$SU(WS_j) \times \sum_{i,(WS_i,WS_j)\in WSR} \frac{1}{deg^+(WS_i)} Agg\_SU(WS_i).$$

Otherwise, $Agg\_SU(WS_j) = SU(WS_j)$. Namely, $Agg\_SU(WS_s) = SU(WS_s) = 1$.

Thus, the successability of the composite Web service CS is the aggregated successability of component Web services whose *outdegree* is zero. Formally, we use $SU(CS)$ to represent the successability of CS, and it can be calculated as follows:

$$SU(CS) = Agg\_SU(WS_f).$$

5) *Reliability for Composite Web Service*: For each component Web service WS in the composite Web service CS, its reliability can be denoted as $RE(WS)$. The aggregated reliability of WS, denoted as $Agg\_RE(WS)$, is the reliability that an execution starting from WS_s will terminate at WS. For a Web service WS_j, the method for calculating $Agg\_RE(WS_j)$ is similar to $Agg\_AV(WS_j)$. It can be defined recursively as follows.

If $deg^-(WS_j) \neq 0$, then $Agg\_RE(WS_j) =$

$$RE(WS_j) \times \sum_{i,(WS_i,WS_j)\in WSR} \frac{1}{deg^+(WS_i)} Agg\_RE(WS_i).$$

Otherwise, $Agg\_RE(WS_j) = RE(WS_j)$. Namely, $Agg\_RE(WS_s) = RE(WS_s) = 1$.

Thus, the reliability of the composite Web service CS is the aggregated reliability of component Web services whose *outdegree* is zero. Formally, we use $RE(CS)$ to represent the reliability of CS, and it can be calculated as follows:

$$RE(CS) = Agg\_RE(WS_f).$$

6) *Latency for Composite Web Service*: For each component Web service WS in the composite Web service CS, its latency can be denoted as $LA(WS)$. The aggregated latency of WS, denoted as $Agg\_LA(WS)$, is the latency that an execution starting from WS_f will terminate at WS. For a Web service WS_i, the method for calculating $Agg\_LA(WS_i)$ is similar to $Agg\_RT(WS_i)$. It can be defined recursively as follows.

If $deg^+(WS_i) \neq 0$, then $Agg\_LA(WS_i) =$

$$LA(WS_i) + \frac{1}{deg^+(WS_i)} \sum_{j,(WS_i,WS_j)\in WSR} Agg\_LA(WS_j).$$

Otherwise, $Agg\_LA(WS_i) = LA(WS_i)$. Namely, $Agg\_LA(WS_f) = LA(WS_f) = 0$.

Thus, the latency of the composite Web service CS is the aggregated latency of component Web services whose *indegree* is zero. Formally, we use $LA(CS)$ to represent the latency of CS, and it can be calculated as follows:

$$LA(CS) = Agg\_LA(WS_s).$$

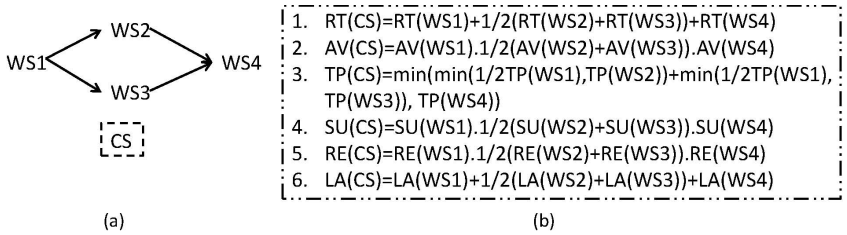


Figure 8 Calculation of composite Web service's QoS

For example, Fig. 8(a) gives a sample composite Web service CS. According to the formulas we defined above, we can calculate its *QoS* in six dimensions, as illustrated in Fig. 8(b).

After calculating the *QoS* of composite Web services, we select the composite Web service whose *QoS* is optimal for recommendation. In order to select an optimal composite Web service, we unity the six dimensions of *QoS* into one value ranging between $[0, 1]$, and the greater the value, the higher quality of the composite Web service. A weight vector $W = (w_1, w_2, w_3, w_4, w_5, w_6)$ is introduced to represent the importance of *QoS*'s each dimension. For response time and latency, the higher the value is, the lower the quality of the composite Web service is. For other dimensions of *QoS*, the higher the value is, the higher the quality of the composite Web service is. Thus, for a composite Web service CS, its *QoS* can be calculated as follows:

$$QoS(CS) = w_1 \cdot \frac{1}{RT(CS)+1} + w_2 \cdot AV(CS) + w_3 \cdot TP(CS)$$

$$+w_4 \cdot SU(CS) + w_5 \cdot RE(CS) + w_6 \cdot \frac{1}{LA(CS)+1},$$

where $\sum_{i=1}^{6} w_i = 1$ and $w_i \geq 0$ for $1 \leq i \leq 6$.

5.3 Boost with Skyline

For a graph g in RG_k, we assume that it has N vertices which are not in G_o (i.e., N clusters) and there are M Web services in each cluster. The total number of corresponding composite Web services is M^N. The computation cost of selecting an optimal composite Web service is $O(M^N)$. Such an approach is impractical for large N and M.

However, from those formalus in Section 5.2, we can find that *QoS* of the composite Web service is better only when component Web services' *QoS* is better. Thus, we can use *skyline* S.Borzsonyi et al. (2001) to boost the selection of optimal composite Web service. First, we utilize *skyline* to select Web services with higher *QoS* in each cluster. And these Web services are the skyline services of each cluster. Then, each vertex is just replaced with skyline services instead of any service in the corresponding cluster, so that we can reduce the computation cost. Please note that the process for computing skyline services of each cluster can be finished offline.

Table 3 QoS of each Web service in Fig. 6(a)

Web service	RT	AV	TP	SU	RE	LA
WS7	0.7	0.4	0.35	0.68	0.2	0.8
WS8	0.5	0.6	0.65	0.6	0.44	0.4
WS9	0.6	0.6	0.65	0.56	0.36	0.6
WS10	0.8	0.52	0.15	0.52	0.28	1

Table 4 QoS of skyline services in $C4 - C6$

Cluster	Skyline service	RT	AV	TP	SU	RE	LA
C4	WS11	0.3	0.8	0.7	0.56	0.6	0.2
	WS16	0.2	0.8	0.6	0.72	0.6	0
C5	WS12	0	0.72	0.65	0.76	0.64	0
	WS17	0.05	0.84	0.65	0.88	0.64	0.1
C6	WS13	0.05	0.8	0.55	0.68	0.84	0.1
	WS18	0.05	0.72	0.75	0.8	0.76	0.1

Table 5 QoS of optimal composite Web services (C-Service means composite service)

No.	C-Service	RT	AV	TP	SU	RE	LA	QoS
1	CS4	1.417	0.202	0.35	0.345	0.075	1.5	0.298
	CS8	1.25	0.229	0.35	0.396	0.091	1.333	0.323
2	CS12	1.725	0.153	0.325	0.253	0.033	1.85	0.247
	CS16	1.433	0.201	0.35	0.333	0.060	1.5	0.292
3	CS18	1.75	0.158	0.325	0.246	0.036	1.8	0.248
	CS20	1.5	0.200	0.35	0.322	0.060	1.55	0.287
4	CS22	1.35	0.218	0.35	0.383	0.087	1.5	0.311

For example, Table 3 shows *QoS* of each Web service in Fig. 6(a), and Table 4 shows *QoS* of skyline services in clusters $C4 - C6$. For each graph in Fig. 7, we replace each vertex with corresponding Web services to get those composite Web services, which are shown in Fig. 9. Then we can calculate *QoS* of each composite Web service in Fig. 9 and get optimal composite Web services. Table 5 shows *QoS* of optimal composite Web services.

6 Experimental Evaluation

6.1 Experiment Setting

We prepare a service repository containing 1,530 services for the experiments. All services are selected from the QWS dataset (`http://www.uoguelph.ca/~qmahmoud/qws/index.html`) (Al-Masri & Mahmoud (2007*a*,*b*)). As mentioned in Section 4.1, we adopt K-medoids Clustering algorithm and get 100 clusters at last. While clustering, we utilize the modified version of the method in (Elgazzar et al. (2010)) to calculate service similarity. The main modifications are those features considered and the method for calculating *Web service name* similarity. Particularly, we consider these features

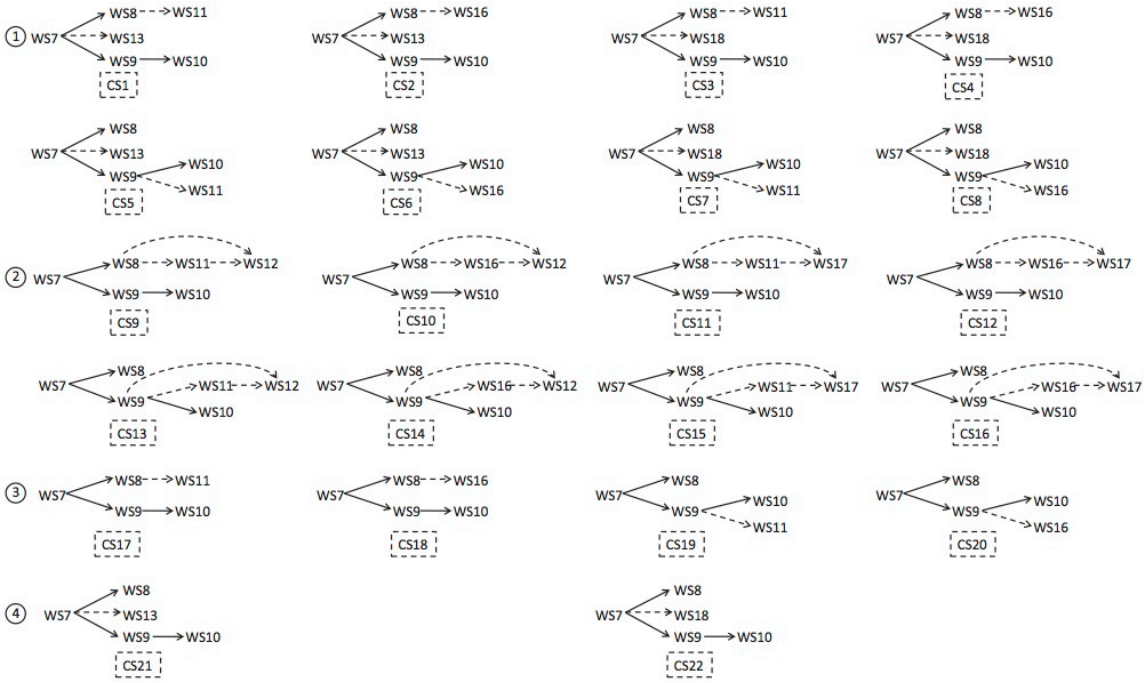


Figure 9 Candidates of composite Web service for recommendation

including *WSDL types*, *WSDL messages*, *WSDL ports* and *Web service name*, while the feature *WSDL contents* is not considered. This is because the implementation for the feature *WSDL contents* is very complex and this feature is less important. While calculating *Web service name* similarity, we do not use Normalized Google Distance (NGD) (Cilibrasi & Vitanyi (2007)) but use the Lin measure of WordNet similarity (Lin (1998), Pedersen et al. (2004)). However, the effectiveness of the Lin measure of WordNet similarity is not better than NGD, because for those words not in WordNet, there will be no similarity. The main reason we do not use NGD is that we have 1,530 services, and *Web service name* of these services has more than 1,000 words. NGD need search engine to search the number of any two words both appear, and there will be more than $1,000^2$ searches. Whereas so many times of auto-search (i.e., machine search, not manual search) are not allowed by current search engines, we utilize the Lin measure of WordNet similarity to calculate *Web service name* similarity. For each word not in WorNet, its similarities with other words are all set to 0.

Furthermore, we prepare a log repository containing 10,000 executed composite services, which are generated based on the service repository. So the size of the composite service graph dataset is 10,000. Edge count of each graph in the composite service graph dataset is uniformly distributed in the range [5,30].

Besides, we generate 1,000 composite services, and those corresponding 1,000 composite service graphs are used as test set, where edge count of each graph is uniformly distributed in the range [10,60]. Then we select half of each graph in test set as input set for recommendation, i.e., each graph in input set like G_o mentioned in Section 5.1. Thus, edge count of each graph in input set is uniformly distributed in the range [5,30].

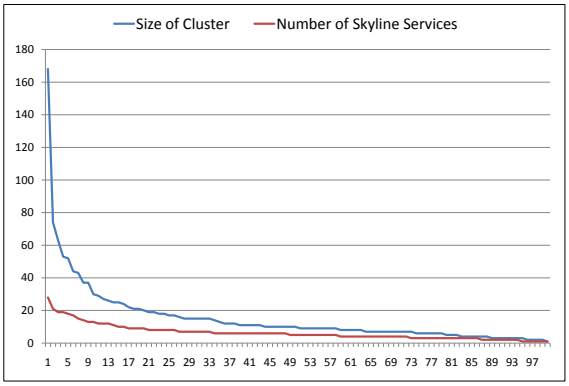


Figure 10 The distribution of clusters and skyline services

While evaluating the efficiency and effectiveness of our approach, random recommendation is used as baseline. Random recommendation with skyline means each vertex replaced with a random selected skyline service in the corresponding cluster. And, random recommendation without skyline means each vertex replaced with a random selected service in the corresponding cluster. In addition, each dimension of the weight vector W (mentioned in Section 5.2) is set to $\frac{1}{6}$ for simplicity in our experiments, i.e., $W = (\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$.

Our experiments run on a desktop PC with Intel Core 2 Duo E7400 2.80GHz CPU and 3G memory, and Windows 7 OS. The program is written in Java and runs on Sun JDK 6 Update 26. To minimize the experimental error, all evaluations are executed in a robust benchmark framework (`http://www.ibm.com/developerworks/java/library/jbenchmark2/`) for Java program.

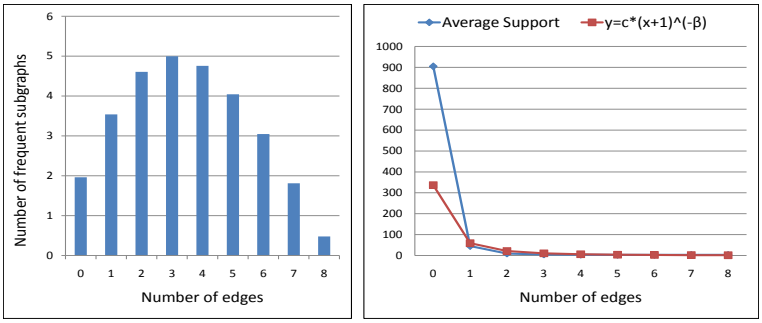
6.2 Distribution Evaluation

In this group of experiments, we evaluate skyline services and those frequent subgraphs mining from the composite service graph dataset.

Figure 10 shows the size of 100 clusters, i.e., the number of services each cluster has. It can be observed that only one cluster has more than 100 services. Five clusters have more than 50 services. 15 clusters have 20∼50 services. 50 clusters have less than 10 services. We calculate skyline services in each cluster, and the number of skyline services in each cluster is also given in Fig. 10. For larger clusters, calculation of skyline services is essential. It should be noted that the largest cluster with 168 services has 28 skyline services.

Then, we evaluate the frequent subgraphs mining from the composite service graph dataset. The minimum support is set to 3.

Figure 11(a) shows the distribution of frequent subgraphs on the number of edges. Please note that the Y-axis is in logarithmic scale. As the number of edges increases, the number of corresponding frequent subgraphs first increases exponentially. When the number of edges reaches 3, the number of corresponding frequent subgraphs reaches a peak. Then as the number of edges increases, the number of corresponding frequent subgraphs decreases exponentially. The result is consistent with our understanding.



(a) Distribution of frequent graphs (b) Distribution of average support

Figure 11 The distribution of frequent subgraphs

We calculate the average support of frequent subgraphs with respect to the number of edges. In order to get the suitable value of α in (3), we use function $y = c \cdot (x+1)^{-\beta}$ to fit the data. The reason we use this function is that we hope the result of (3) is around 1 and the impact of edge count on support can be eliminated. Figure 11(b) shows the results. We can observe that the average support of frequent subgraphs decreases exponentially as the number of edges increases. And we get the value of c is 336.853, and the value of β is 2.516. So when the minimum support is 3, the suitable value of α is 2.516.

6.3 Efficiency Evaluation

In this group of experiments, we evaluate the efficiency of our approach. Because data preparation stage and data processing stage can be completed offline, we just study the time of service recommendation stage.

As mentioned in Section 5, we first determine frequent subgraphs for recommendation. Then, we replace each vertex with the corresponding cluster's skyline services to find optimal composite Web service for each frequent subgraph. We test the average time when we utilize or not utilize skyline services to obtain optimal recommendation. Meanwhile, we also test the average time of random recommendation with or without skyline. The results are shown in Table 6. For optimal recommendation, the average time without skyline is 8 times more than the average time with skyline. Thus, utilizing skyline services is quite efficiency for optimal recommendation. Further, it can be observed that skyline does not affect the average time of random recommendation. The average time of random recommendation is much less than that of optimal recommendation, which is consistent with our understanding. Assume that there are N vertices and each vertex can be replaced with M Web services, the time complexity of random recommendation is $O(N)$ while optimal recommendation is $O(M^N)$.

6.4 Effect Evaluation

In this group of experiments, we evaluate the practical effectiveness of the proposed approach. First, we evaluate the extended edge count of recommendation, as illustrated in Fig. 12. In Fig. 12(a), the extended edge count decreases when the minimum support

Table 6 Average time of different approaches(ms)

Method	With Skyline	Without Skyline
Optimal Selection	**36.379**	308.291
Random Selection	0.661	0.661

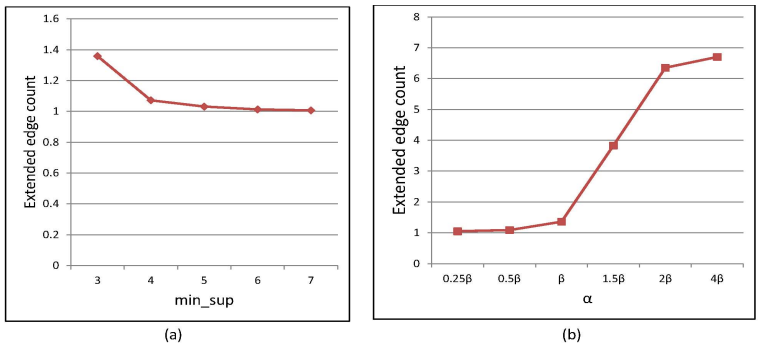


Figure 12 Effect evaluation on extended edge count

support increases. This is because edge count of frequent subgraphs decreases along with the increase of minimum support. It should be noted that for different minimum support, the suitable value of α is set according to method mentioned in Section 6.2. When the minimum support varies from 3 to 7, the suitable value of α is 2.516, 2.572, 2.676, 2.804 and 2.72 respectively. In Fig. 12(b), the extended edge count increases along with the increase of α. This is because when α is small, the result of (3) is mainly affected by support. The graphs with larger support often have fewer edges. It should be noted that the minimum support is set to 3 in Fig. 12(b).

Then, we evalaute the hit rate of recommendation which mainly considers functionality. That is to say, we concern the structure of the recommended composite Web services (i.e., these corresponding composite service graphs). For one graph G_l in the test set, we denote its corresponding graph in the input set as G_s. Thus, $E(G_s) = \frac{1}{2}E(G_l)$. For G_s, we denote its list of recommended graphs as $RG_{G_s,k}$, where k is the parameter limiting the number of frequent subgraphs recommended for users (mentioned in Section 5.1). If one graph in $RG_{G_s,k}$ is isomorphic to the corresponding subgraph in G_l, this recommendation is hit. For the entire test set and input set, there will be 1,000 recommendations. We can get the number of hit recommendations, and the hit rate is the corresponding percentage.

Figure 13(a) shows the change of hit rate with k when the minimum support varies from 3 to 7. For each minimum support, the hit rate increases along with the increase of k. When k reaches 10, the hit rate for each minimum support is at least 0.768. With the same k and different minimum support, the difference in hit rate is small. However, in general, with the same k, the hit rate decreases slightly as the minimum support increases. This is because the larger the minimum support, the less the number of frequent subgraphs. Thus, when k is 10 and the minimum support is 3, the hit rate reaches the highest, i.e., the effectiveness is best. It should be noted that for different minimum support, the suitable value of α is set according to method mentioned in Section 6.2.

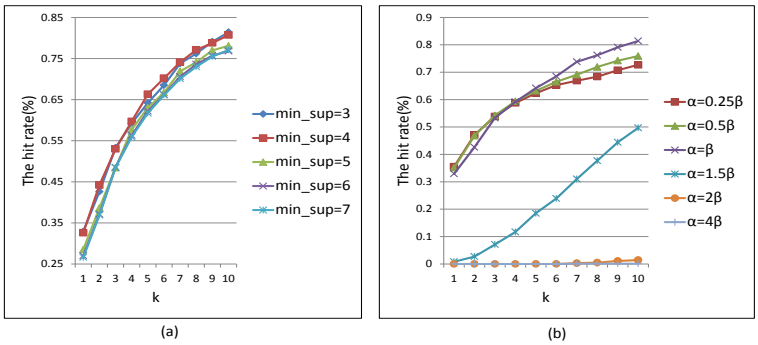


Figure 13 Effect evaluation on hit rate

Table 7 Effect evaluation of different approaches on QoS

Method	With Skyline	Without Skyline
Optimal Selection	**0.5333**	0.5333
Random Selection	0.5316	0.5313

We evaluate the impact of α on the hit rate, by setting the minimum support to 3. The results are shown in Fig. 13(b). For each value of α, the hit rate increases along with the increase of k. With the same k, when $\alpha \geq \beta$, the hit rate decreases rapidly along with the increase of α. Because the result of (3) is mainly affected by edge count, graphs for recommendation should have more edges. However, the larger the number of edges, the smaller the support. Thus, the hit rate is low. With the same k, when $\alpha \leq \beta$, the difference in hit rate is small for different α. Because the result of (3) is mainly affected by support. However, when $k \geq 5$ and $\alpha \leq \beta$, with the same k, the hit rate increases as α increases. Thus, when $k > 5$, the suitable value of α is β.

Furthermore, we evaluate these recommended composite Web services' average *QoS*. The results are given in Table 7. For optimal recommendation, these recommended composite Web services' average *QoS* of the approach with skyline is the same with the one without skyline. That is to say, optimal composite Web service is in the candidates of composite Web services replaced with skyline services, which means our approach is feasible and correct. For random recommendation, these recommended composite Web services' average *QoS* is better with skyline, because skyline services' *QoS* is not worse than any service in the same cluster. These recommended composite Web services' average *QoS* of optimal recommendation is better than that of random recommendation, which is consistent with our understanding.

7 Conclusion and Future Work

Hidden knowledge is implied in gigantic amount of service usage data and the structure of composite services. In this paper, we propose a graph mining based recommendation approach to model and explore the hidden knowledge to facilitate users' service composition task.

The technical contributions of this work are threefold: Extend the graph mining approach *gSpan* to mine **FUWS**s; Recommend appropriate Web services to facilitate the composition task based on identified **FUWS**s; Select optimal composite services with consideration of overall QoS using *skyline*. Further, we demonstrate the efficiency and effectiveness of the proposed approach by implementing it based on 1,530 real Web services.

There are many interesting research problems related to our work should be pursued further. For instance, how to extend our approach to provide personalize recommendation, to integrate more information such as Web services tags, to further enhance the effectiveness and efficiency.

8 Acknowledgement

This research was partially supported by the National Technology Support Program under the grant of 2011BAH16B04, the National Natural Science Foundation of China under the grant of No. 61173176, Science and Technology Program of Zhejiang Province under the grant of 2008C03007, National High-Tech Research and Development Plan of China under the Grant No. 2011AA010501.

References

Al-Masri, E. & Mahmoud, Q. H. (2007*a*), 'Discovering the best web service', *16th International Conference on World Wide Web Conference(WWW)* pp. 1257–1258.

Al-Masri, E. & Mahmoud, Q. H. (2007*b*), 'Qos-based discovery and ranking of web services', *IEEE 16th International Conference on Computer Communications and Networks (ICCCN)* pp. 529–534.

Bayada, D. M., Simpson, R. W. & Johnson, A. P. (1992), 'An algorithm for the multiple common subgraph problem', *Journal of Chemical Information and Computer Sciences* **32**, 680–685.

Chen, X., Liu, X., Huang, Z. & Sun, H. (2010), 'Regionknn: A scalable hybrid collaborative filtering algorithm for personalized web service recommendation', *2010 IEEE International Conference on Web Services (ICWS 2010)* pp. 9–16.

Cilibrasi, R. L. & Vitanyi, P. M. (2007), 'The google similarity distance', *IEEE Transactions on Knowledge and Data Engineering* **19**(3), 370–383.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2002), *Introduction to Algorithms*, second edn, MIT Press and McGraw-Hill.

Daly, O. & Taniar, D. (2004), 'Exception rules mining based on negative association rules', *Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2004),Lecture Notes in Computer Science* **3046**, 543–552.

Dehaspe, L., Toivonen, H. & King, R. D. (1998), 'Finding frequent substructures in chemical compounds', *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)* pp. 30–36.

Elgazzar, K., Hassan, A. E. & Martin, P. (2010), 'Clustering wsdl documents to bootstrap the discovery of web services', *2010 IEEE International Conference on Web Services (ICWS 2010)* pp. 147–154.

Holder, L. B., Cook, D. J. & Djoko, S. (1994), 'Substructure discovery in the subdue system', *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases* pp. 169–180.

Hu, H., Yan, X., Huang, Y., Han, J. & Zhou, X. J. (2005), 'Mining coherent dense subgraphs across massive biological network for functional discovery', *Bioinformatics* **21**, 213–221.

Inokuchi, A., Washio, T. & Motoda, H. (2000), 'An apriori-based algorithm for mining frequent substructures from graph data', *Proceedings of 2000 European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'00)* pp. 13–23.

Kuramochi, M. & Karypis, G. (2001), 'Frequent subgraph discovery', *Proceedings of 2001 IEEE International Conference on Data Mining (ICDM'01)* pp. 313–320.

Kuramochi, M. & Karypis, G. (2005), 'Finding frequent patterns in a large sparse graph', *Data Mining and Knowledge Discovery* **11**(3), 243–271.

Li, J. & Hou, Z. (2005), 'Research survey of web service composition (in chinese)', *Computer Application Research* **12**, 4–7.

Li, S., Chen, H. & Chen, X. (2010), 'A mechanism for web service selection and recommendation based on multi-qos constraints', *2010 6th World Congress on Services Services (Services 2010)* pp. 221–228.

Liang, Q., Li, P., Hung, P. C. K. & Wu, X. (2009), 'Clustering web services for automatic categorization', *2009 IEEE International Conference on Services Computing (SCC 2009)* pp. 380–387.

Lin, D. (1998), 'An information-theoretic definition of similarity', *Proceedings of the 15th International Conference on Machine Learning (ICML 1998)* pp. 296–304.

Moraru, A., Fortuna, C., Fortuna, B. & Slăvescu, R. R. (2009), 'A hybrid approach to qos-aware web service classification and recommendation', *IEEE 5th International Conference on Intelligent Computer Communication and Processing (ICCP 2009)* pp. 343–346.

Nayak, R. & Lee, B. (2007), 'Web service discovery with additional semantics and clustering', *2007 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2007)* pp. 555–558.

Pedersen, T., Patwardhan, S. & Michelizzi, J. (2004), 'Wordnet::similarity - measuring the relatedness of concepts', *Proceedings of HLT-NAACL–Demonstrations '04 Demonstration Papers at HLT-NAACL 2004* pp. 38–41.

Pei, J., Jiang, D. & Zhang, A. (2005), 'On mining cross-graph quasi-cliques', *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)* pp. 228–238.

S.Borzsonyi, D.Kossmann & K.Stocker (2001), 'The skyline operator', *International Conference on Data Engineering* pp. 421–430.

Skoutas, D., Sacharidis, D., Simitsis, A. & Sellis, T. (2010), 'Ranking and clustering web services using multicriteria dominance relationships', *IEEE Transactions on Services Computing* **3**(3), 163–177.

Takajashi, Y., Satoh, Y., Suzuki, H. & Saski, S. (1987), 'Recognition of largest common fragment among a variety of chemical structures', *Analytical Sciences* **3**, 23–38.

Taniar, D. & Goh, J. (2007), 'On mining movement pattern from mobile users', *International Journal of Distributed Sensor Networks* **3**(1), 69–86.

Taniar, D., Rahayu, W., Lee, V. C. S. & Daly, O. (2008), 'Exception rules in association rule mining', *Applied Mathematics and Computation* **205**(2), 735–750.

Thio, N. & Karunasekera, S. (2005), 'Automatic measurement of a qos metric for web service recommendation', *Proceedings of the 2005 Australian Software Engineering Conference (ASWEC'05)* pp. 202–211.

Wang, C., Wang, W., Pei, J., Zhu, Y. & Shi, B. (2004), 'Scalable mining of large disk-based graph databases', *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)* pp. 316–325.

Wang, J., Zeng, Z. & Zhou, L. (2006), 'Clan: An algorithm for mining closed cliques from large dense graph databases', *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)* pp. 73–82.

Wu, J., Chen, L., Jian, H. & Wu, Z. (2012), 'Composite service recommendation based on bayes theorem', *International Journal of Web Services Research* **9**(2), 69–93.

Yan, X. & Han, J. (2002), 'gspan: Graph-based substructure pattern mining', *Proceedings of 2002 IEEE International Conference on Data Mining (ICDM'02)* pp. 721–724.

Yan, X. & Han, J. (2003), 'Closegraph: Mining closed frequent graph patterns', *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)* pp. 286–295.

Yan, X., Yuz, P. S. & Han, J. (2004), 'Graph indexing: A frequent structure-based approach', *Proceedings of 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04)* pp. 335–346.

Zeng, Z., Wang, J., Zhou, L. & Karypis, G. (2007), 'Out-of-core coherent closed quasi-clique mining from large dense graph databases', *ACM Transactions on Database Systems*.

Zheng, Z., Ma, H., Lyu, M. R. & King, I. (2011), 'Qos-aware web service recommendation by collaborative filtering', *IEEE Transactions on Services Computing* **4**(2), 140–152.