A QoS-Aware Hyper-graph Based Method of Semantic Service Composition

Cui Lizhen, Xu Meng

A QoS-Aware Hyper-Graph Based Method of Semantic Service Composition

Cui Lizhen, Xu Meng

School of Computer Science and Technology, Shandong University, Jinan, China
Shandong Provincial Key Laboratory of Software Engineering
clz@sdu.edu.cn

Abstract—Semantic service composition is an important issue in web service based systems. It is about how to build a new, reusable and value-added service by combining existing semantic web services. In this paper we propose a new hyper-graph based method supporting semantic service composition, which model the service composition problem as a hyper-graph. Then the hyper-graph based service composition algorithm has been implemented and evaluated. The evaluation results have shown that the method can effectively reduce the composition time and has good robustness, especially in the case of large number of web services and concepts.

Index Terms—QoS-aware, Hyper-graph, Service Composition

1 Introduction

Service-oriented computing (SOC) has been widely accepted as the next generation programming paradigm. The web services are loosely coupled reusable and self-describing software components that encapsulate discrete functionality and are distributed and programmatically accessible over standard internet protocols [1]. It is an alternative and promising approach to development web-based application or system rapidly to combine the existing Web Services. But with the number of Web Service on the internet over the last few years increasing sharply, it is difficult and time consuming to combine web services manually [2]. Therefore, semantic web is introduced into Web Service technologies to support dynamic and automated tasks such as discovery, composition and execution. In recent years, how to composite services automatically have drawn lots of research attentions [3-9].

In this paper we proposed a QoS(Quality of Service)-aware hyper-graph based method for semantic Web Service Composition. We model the service composition problem as two hyper-graphs. And by pruning the vertexes and the link between two hyper-graphs, the composite service is built effectively.

The main contributions of this paper are as follows:

(1) Show how the problem of service composition can be modeled as a hyper-graph transversal problem.

(2)Tackle the problem of automatic service composition and QoS-aware service selection problem together with an algorithm that both efficient and 100% precise.

The rest of the paper is organized as follows: Section 2 reviews related work and motivates this work. Section 3 shows how to model the service composition problem as a hyper-graph transversal problem. Section 4 discusses the detail of the approach. Section 5 evaluates the performance of the proposed QoS-aware hyper-graph based method of semantic service composition in this paper, and finally Section 6 concludes.

2 Related Work

Since it is very difficult and time consuming to find composite service manually from huge amount of services, automatic service composition is proposed to enable automatic building composition for a given request. Some approaches, such as [5, 6], tackle Web service composition problem via AI planning. On the other hand, some other researches [4, 10] regards it as a graph search problem and solves it by shortest path, A*, etc. algorithms.

To guarantee local or global QoS requirements of service composition, the QoS-aware service composition has attracted a lot of attention in recent years. Different from automatic service composition, it assumes that a work plan is predefined and has a set of "abstract" tasks and the objective is to select service for each task from its candidate services to meet local or global QoS constraints. Zeng [12] uses two service selection approaches based on Multiple Criteria Decision Making. In [13], the authors model the problem as a multi-choice 0-1 knapsack problem or a multi-constraint optimal path problem, and then compute optimal result according to the objective function. In addition, a genetic algorithm is proposed to address the problem in [3]. Although it is more inefficient than Integer Programming, it can deal with non-linear constraints.

In one words, all the methods above for automatic service composition or QoS-aware service composition only consider one aspect of QoS-aware automatic service composition, which is to search work plans for a given request or to guarantee global QoS requirements. Furthermore, Some researchers have proposed approaches to address the QoS-aware automatic service composition problem, such as [14-16]. Peter [11] proposes a fast approach by some index tables stored in database to fit the competition CEC'09[11]. But it cannot provide 100% precision and its efficiency are not good enough when the scale of services is big.

In [17], Zhu modeled the problem of computing cost constrained collections of Web services as a constrained hyper-graph transversal problem. But it only can compute a set of services to fulfill some functions. Out method can find a composite service which can satisfy a user request.

3 Definition and Problem Model

We now provide the necessary definition about hyper-graphs and the semantic service composition. Then a formal description of the problem will be given.

Definition 1 A hyper-graph H is a pair H = <E, V> where V is a finite set of vertices V = { v_1, v_2,…,v_m } and E is a family of (hyper) edges, E ={ e_1, e_2,…, e_t } where each e_i is a subset of V, $1 \le i \le t$. We assume $V = \{ \cup e_i \mid e_i \in E, 1 \le i \le t \}$.

Definition 2 The candidate service is a service which can be used in the service composition process. The candidate service set is a finite set of services $S=\{s_i|s_i$ is a cadidate service$\}$. And $s_i=<p(s_i)_{in}, p(s_i)_{out}, QoS>$, where $p(s_i)_{in}$ is the input parameters set of s_i, $p(s_i)_{out}$ is the output parameters set of s_i, QoS is the QoS property of s_i.

In this paper, we only consider the response time as the QoS property of web service. And the computing method can be find in [18].

Definition 3 The composition request is a request of user which can be represent as $R= \{P_{in},P_{out}\}$, P_{in} is the set of input parameters and P_{out} is the set of output parameters.

We now describe the reduction of the semantic service composition problem to the problem involving hyper-graphs. We will model the problem to two hyper-graphs. One is called output parameters hyper-graph, which is represented by H_o. And the other is called input parameters hyper-graph, which is represented by H_i. Each Web Service in S is modeled as a vertex both of H_o and H_i. Furthermore, a virtual service (vertex) is added in H_o and H_i respectively. The virtual service which is added in H_o is called input_service $I_s=\{\varphi,R.P_{in}, 0\}$ whilst the virtual service which is added in H_i is called output_service $O_s=\{R.P_{out}, \varphi, 0\}$. Then we will add edges in H_o and H_i. Each edge in $H_o(H_i)$ corresponds to a set of vertices (web services) which have a particular output(input) parameter. The transversals of the two hyper-graphs now correspond to two sets of services that cover all the services requested by the user.

Example 1 Let us consider a example. $S=\{s_1, s_2, s_3, s_4, s_5, s_6\}$. The input and output parameters of all services are listed in Table 1. The request of user is R={{a,b,e}, {f,g}}.

TABLE 1. CANDIDATE SERVICES

Service Name	input	output	QoS
s1	a, b	c, d	10
s2	c	g	20
s3	d, e	f	15
s4	c	g	25
s5	c, d	f, g	30
s6	f, g	h	20
s7	h, i	j	15

The request of user is R={{a,b,e},{f,g}}.

According to the R, we add two virtual services $I_s=\{\{\phi,\{a,b,e\}\}$ and $O_s=\{\{f,g\}, \phi\}$. Then the H_o and H_i are drawn in Fig. 1.

$$H_o = \begin{matrix} e1: I_s^0 \\ e2: I_s^0 \\ e3: I_s^0 \\ e4: s_1^{10} \\ e5: s_1^{10} \\ e6: s_2^{30} \quad s_4^{35} \quad s_5^{40} \\ e7: s_3^{25} \quad s_5^{40} \\ e8: s_6^{60} \end{matrix} \qquad H_i = \begin{matrix} e1: s_1^{10} \\ e2: s_1^{10} \\ e3: s_3^{15} \\ e4: s_2^{30} \quad s_4^{35} \quad s_5^{40} \\ e5: s_3^{25} \quad s_5^{40} \\ e6: s_6^{60} \quad O_s^{30} \\ e7: s_6^{55} \quad O_s^{25} \\ e8: s_7^{75} \end{matrix}$$

Fig. 1. Hi and Ho

4 Hyper-Graph Based Service Composition Algorithm

In this section, we will describe the hyper-graph based service composition algorithm. We will use the example shown in Fig. 1. Response time of every service is shown using superscripts. Fig. 2 illustrates the basic outline of the algorithm.

First, the two hyper-graphs are generated according to the request and candidate service set. During the process of generating hyper-graphs, the complete time is calculated for every service. Then a pruning strategy is applied to delete the useless service and edge. At last, a solution generation algorithm is used to create the composite service.

Input: User Request R, Candidate Service Set S
Output: Composite Service CS

1. $H_o = \phi$, $H_i = \phi$
2. HG_Generate(R, S, H_o, H_i)
3. Prune(H_o, H_i)
4. CS=Solution_Generate(H_o, H_i)
5. Return CS

Fig. 2. Hyper-Graph Based Service Composition Algorithm

4.1 Generating the Hyper-Graphs

Fig. 3 shows how to generate the hyper-graphs. First, two virtual services, I_s and O_s, is created. The output of I_s is the input of request $R.P_{in}$ and the input of O_s is the output of request $R.P_{out}$, while the input of I_s and the output of O_s isφ (line 0). After the creation of the virtual services, the I_s is added into H_o and its complete time is computed. And an edge is added into H_o for every output parameter of I_s. Moreover, all parameters of I_s.output is put into a queue Q (line 2-6). If the queue Q is not null, it suggests that there are parameters which are not processed. So we get the parameter and find the services whose input parameters include it. Then the service is added into H_i and its complete time is calculated. At that time, it is decided whether the edge

need to be added into H_i (line 9-17). If all of the input parameters of the service is in H_o, the service is added into H_o. Repeat that until the queue is null. After doing all this, we get two hyper-graph H_i and H_o. Fig. 1 shows the H_i and H_o which is generated by the algorithm.

```
        HG_Generate
      Input: User Request R, Candidate Service Set S
      Output: Hyper-Graphs Ho, Hi
1.    create virtual service Is={φ,R.Pin, 0}, Os={R.Pout, φ, 0}
2.    set Is.complete_time=0
3.    for each p∈Is.output do
4.      add ep to Ho
5.      add Is to Ho.ep
6.      EnQueue(Q,p)
7.    end for
8.    while Q!=NULL
9.      p= DeQueue(Q)
10.     while ∃ service s which input includes p
11.       if s.complete_time<s.response_time+min(Ho.ep.s.complete_time)
12.         s.complete_time=s.response_time+min(Ho.ep.s.complete_time)
13.       end if
14.       if ep not in Hi
15.         add ep to Hi
16.       end if
17.       add s to Hi.ep
18.       if all p ∈s.input exist ep in Ho
19.         for each p ∈s.output do
20.           add ep to Ho
21.           add s to Ho.ep
22.           EnQueue(Q,p)
23.         end for
24.       end if
25.     end while
26.   end while
```

Fig. 3. Hyper-Graph Generating Algorithm

4.2 Pruning Strategy

After the hyper-graphs is generated, they should be pruned to remove the useless vertexes and edges. Fig. 4 shows the pruning strategy. By this way, the search space and the time of composition time can be reduced. Before the pruning strategy is applied, the redundant vertexes of H_i should be deleted and the edge order of the H_i should be adjusted first. We arrange the edges which contain the service O_s at last (line 1-2). By doing this, the pruning strategy can be applied from the last to first of H_i. The edges of H_i which contain O_s should be reserved and all vertexes except O_s in the edge should be deleted. On the other hand, the edges of H_i which do not contain O_s will only reserve the vertex which has the minimal complete time and all other

vertexes will be deleted. After the edge of H_o is pruned, the edge of H_i which has the same subscript will be pruned. Then we will get two pruned hyper-graphs after H_o and H_i are pruned. Fig. 5 shows the pruned hyper-graphs of Fig. 1.

Prune
Input: Hyper-Graphs H_o, H_i
Output: Hyper-Graphs H_o, H_i after pruned

1. delete all vertexes from H_i which not in H_o
2. sort the edges $H_i.e$ to make all edges $e \supseteq O_s$ have max subscript
3. for i=number of edges in H_i to 1 do
4. if $H_i.e_i = \phi$
5. delete $H_i.e_i$
6. delete $H_o.e_i$
7. continue
8. end if
9. if $H_i.e_i \supseteq O_s$
10. $s_i=O_s$
11. else
12. s_i=min_complete_time($H_i.e_i$)
13. end if
14. delete all vertexes except s_i which in $H_i.e_i$ from $H_i.e_i$
15. t= s_i.complete_time- s_i.response_time
16. s_o=s whose complete_time=t and $s \in H_o.e_i$
17. delete all vertexes except s_o which in $H_o.e_i$ from $H_o.e_i$
18. delete all vertexes except s_o which in $H_o.e_i$ from H_i
19. end for

Fig. 4. Pruning Algorithm

$$H_o = \begin{matrix} e1: I_s^0 \\ e2: I_s^0 \\ e3: I_s^0 \\ e4: s_1^{10} \\ e5: s_1^{10} \\ e6: s_2^{30} \\ e7: s_3^{25} \end{matrix} \quad H_i = \begin{matrix} e1: s_1^{10} \\ e2: s_1^{10} \\ e3: s_3^{15} \\ e4: s_2^{30} \\ e5: s_3^{25} \\ e6: O_s^{30} \\ e7: O_s^{25} \end{matrix}$$

Fig. 5. Pruned Ho and Hi

4.3 Solution Generation

Solution Generation is used to generate the composite service of the request. After applying the pruning strategy to H_o and H_i, we get two pruned hyper-graph. The every edge of pruned H_o and H_i must have and only have one vertex. Furthermore, the number of edges of H_o and H_i is equal. So we can get the i^{th} edge of H_o and the i^{th}

edge of H_i, and then link the vertexes of two services in the edges. This is an edge in composite service. Combining all edges which we get, the composite service is generated. Fig. 6 shows the solution generating algorithm and Fig. 7 gives the composite service of Example 1.

```
Solution_Generate
Input: Hyper-Graphs Ho, Hi
Output: Composite Service CS
1.  Graph CS=φ, temp=φ
2.  for i=1 to number of edges in Ho do
3.    get so from Ho.ei
4.    get si from Hi.ei
5.    add so to temp
6.    add si to temp
7.    add edge <so, si> to temp
8.    CS=combine(CS, temp)
9.    temp=φ
10. end for
11. return CS
```

Fig. 6. Solution Generating Algorithm

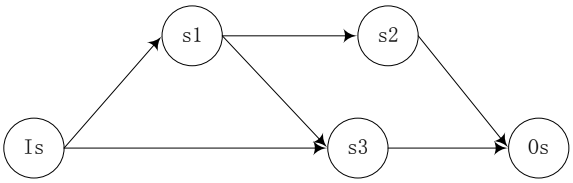


Fig. 7. Composite Service

5 Experiments and Results

To evaluate the performance of our method, we have developed some experiments. And our evaluation consists of two parts. First, we compare SUTB [4] with our method by the precision and composition time. Then we carry out some experiments on the test sets which are generated by the WSC 2009 Testset Generator to evaluate the method furthermore. This generator has three input parameters: concept number, service number and solution depth. In order to prove the scalability and efficiency of our algorithm in different and even extreme conditions, we generate five groups of test sets which have solution depth 10, 15, 20, 25 and 30 respectively. And each group contains three subgroup of different ratio of concept number and service number. We generate service number of test sets from 4000 to 20000 to make sure that our method has good scalability. Furthermore, we conduct experiments on each test set several times to get the average values. All experiments were performed on a PC platform with a Intel® Core™2 Duo 2.33GHz CPU, Windows XP, and 2 GB RAM and all algorithms were implemented in Java.

5.1 Compare with SUTB

We use the 5 Testsets of WSC'09[19] to evaluate the precision and composition time of SUTB and HGSC(Hyper-Graph based Service Composition). The composition time is shown in Fig. 8. The composition time of SUTB is increased sharply from Testset01 toTestset05. But it is not true for HGSC. The composition time of HGSC has little changed. On the other hand, from Table II, the precision of HGSC is 100% while SUTB is only 20%. So we can conclude that HGSC has a better scalability and precision.

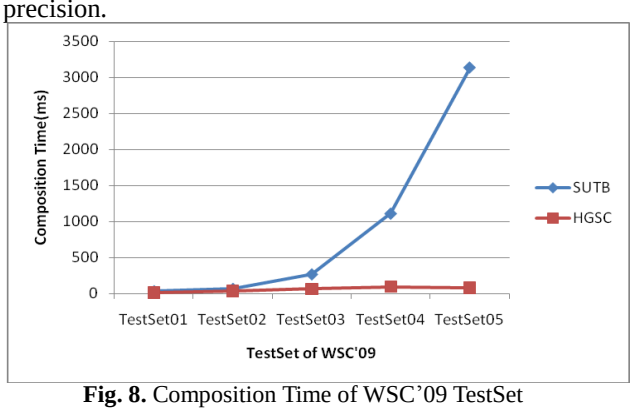


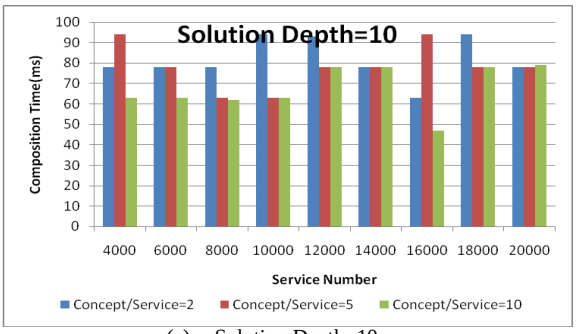
Fig. 8. Composition Time of WSC'09 TestSet

TABLE 2. PRICISION OF SUTB AND HGSC

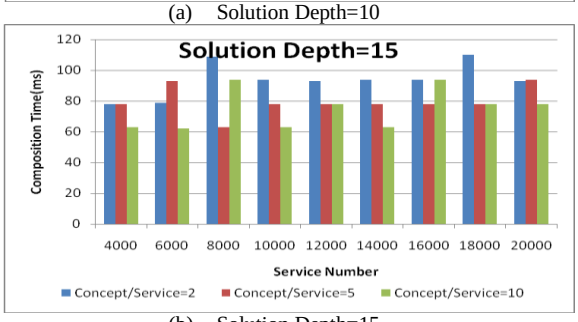
TestSets	SUTB	HGSC	Optimal QoS
TestSets01	780	500	500
TestSets02	2100	1690	1690
TestSets03	760	760	760
TestSets04	2070	1470	1470
TestSets05	4500	4070	4070

5.2 Experiments with Other Testsets

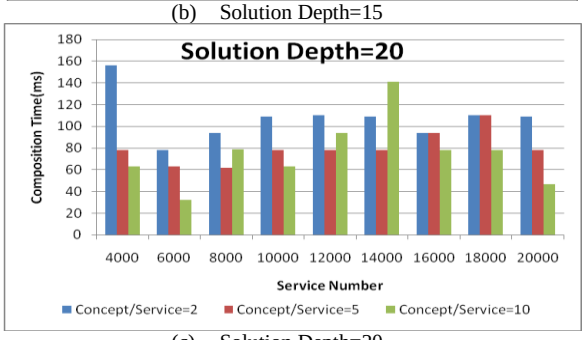
We use WSC 2009 Testset Generator to generate more testsets and evaluate HGSC furthermore. Fig. 9 shows the experiment results with different service number (4000-20000), concept number (8000-200000) and solution depth (10-30). From Fig. 9, we can find that the maximum composition time of HGSC is less than 180ms. In addition, with the change of service number, concept number and solution depth, the composition time varied little. So the HGSC has good scalability and can be applied in large number of service and concept.



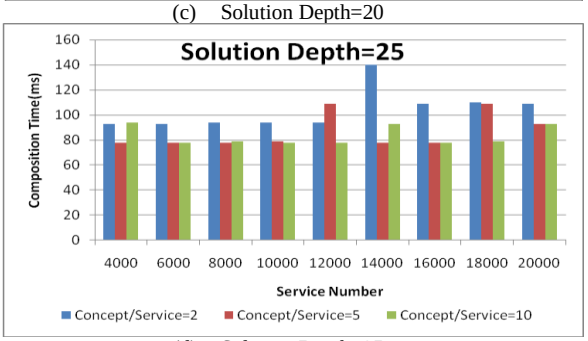
(a) Solution Depth=10



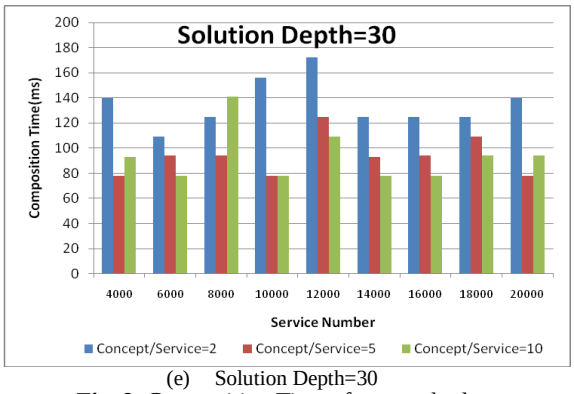
(b) Solution Depth=15



(c) Solution Depth=20



(d) Solution Depth=25



(e) Solution Depth=30

Fig. 9. Composition Time of our method

6 Conclusion and Future Work

To handle QoS-aware service composition automatically and efficiently, a QoS-aware hyper-graph based method is proposed in this paper. We show how to model the service composition problem as a hyper-graph and give the algorithms of generating hyper-graph, pruning strategy and solution generating. At last, to address the scalability, precision and performance of our method, some experiments are conducted and results show that our method has good scalability, efficiency and 100% precision.

In future work, we plan to develop the method in a distributed way.

Acknowledgment

This work has been supported by the National Natural Science Foundation of China under Grant No. 61003253; the Natural Science Foundation of Shandong Province of China under Grant No. ZR2010FM031,ZR2010FQ010.

References

[1] Tsur S, Abiteboul S, Agrawal R, Dayal U, Klein J, Weikum G, “Are Web services the next revolution in e-commerce?” In: Proceedings of the VLDB conference, Rome, September 2001, pp 614–617

[2] Freddy L ́ecu, “A Formal Model for Semantic Web Service Composition” In: ISWC 2006, 2006.

[3] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In GECCO, pages 1069–1075, 2005.

[4] S. V. Hashemian and F. Mavaddat. A graph-based framework for composition of stateless web services. In ECOWS, pages 75–86, 2006.

[5] S. McIlraith and T. Son. Adapting golog for composition of semantic web services. In KR2002, pages 482–493, Toulouse, France, April 22-25 2002.

[6] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. Htn planning for web service composition using shop2. Web Semantics: Science, Services and Agents on the World Wide Web, 1(4):377–396, October 2004.

[7] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web, 1(1):6, 2007.
[8] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. IEEE Transactions on Software Engineering, 30(5):311–327, May 2004.
[9] A. Zhou, S. Huang, and X. Wang. Bits: A binary tree based web service composition system. Int. J. Web Service Res., 4(1):40–58, 2007.
[10] N. Milanovic and M. Malek. Search strategies for automatic web service composition. Int. J. Web Service Res., 3(2):1–32, 2006.
[11] P. Bartalos and M. Bielikova. Semantic web service composition framework based on parallel processing. In CEC'09, Vienna, Austria, 2009.
[12] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. IEEE Transactions on Software Engineering, 30(5):311–327, May 2004.
[13] T. Yu, Y. Zhang, and K.-J. Lin. Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web, 1(1):6, 2007.
[14] J. Liu, N. Gu, Y. Zong, Z. Ding, S. Zhang, and Q. Zhang. Web services automatic composition based on qos. In ICEBE '05, pages 607–610, 2005.
[15] X. Wang, S. Huang, and A. Zhou. Qos-aware composite services retrieval. J. Comput. Sci. Technol., 21(4):547–558, 2006.
[16] M. Naseri and A. Towhidi. Qos-aware automatic composition of web services using ai planners. In ICIW '07, page 29, 2007.
[17] Zhou Zhu, James Bailey, "Fast Discovery of Interesting Collections of Web Services," wi, pp.152-160, 2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI'06), 2006.
[18] Danilo Ardagna, Barbara Pernici, "Adaptive Service Composition in Flexible Processes," IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 369-384, June 2007.
[19] Srividya Kona, Ajay Bansal, M. Brian Blake, Steffen Bleul, Thomas Weise, "WSC-2009: A Quality of Service-Oriented Web Services Challenge," cec, pp.487-490, 2009 IEEE Conference on Commerce and Enterprise Computing, 2009