A Method of Improving QoS: Explorations of the Possibility of Function Combinations in Service Compositions
品質改良のアプローチ：機能のぶれを許すサービス合成

by

Ziyuan Wang
王子源

A Senior Thesis
卒業論文

Submitted to
the Department of Information Science
the Faculty of Science, the University of Tokyo
on February 9, 2016
in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science

Thesis Supervisor: Shinichi Honiden　本位田真一
Professor of Information Science

ABSTRACT

There is a growing need for web service providers to develop customised and flexible web services as quick as they can. One way to satisfy this demand is to utilise service compositions, which provide a method of consolidating several services to a richer service. Because of the uncertainty in the feasibility of the functional and QoS requirements, it is not guaranteed that service compositions succeed with suitable solutions.

One way to satisfy the given requirements is to control the given two kinds of requirements. There have been studies on methods of optimising the QoS with fixed functional requirements. However, the search space in that case is limited, which would possibly not include service compositions with better QoS whose functions are slightly different from the required one. Therefore, I focus on the possibility of improving the QoS by compromising on functions or exploring the possibility of function combinations. This enables better advice of service compositions to users. In this article, I propose a method of efficiently searching different possibilities to support users to balance functional requirements and QoS requirements.

論文要旨

近年，ウェブサービスプロバイダーにとって，ユーザーの好みに合わせた，柔軟なウェブサービスを短時間で作る必要性が増してきている．そのために，よく使われている手法として，複数のサービスを一つにまとめてより高機能なサービスを作るサービス合成と呼ばれる手法がある．サービス合成において，入力として受け取る機能や品質に関する要求を必ず満たせるとは限らないという問題がある．

機能と品質の両方を達成するために既存研究で提案されている手法として，その片方を固定して保証し，もう片方を最適化するというものがある．しかし，この手法では，機能を固定するために，探索空間が限られているため，要求された機能に妥協を許した場合に品質が大きく高まるような合成を見逃す可能性がある．よって，本論文では，ユーザーが高品質の合成を見つけることがより容易になるよう，機能を妥協し，要求された機能と少しのぶれを許すことで探索空間を増やすことによる QoS の最適化の可能性について議論した．本論文では，ユーザーが機能要求と品質要求のバランスを取れるように効率的に様々な合成の可能性を探索する手法を提案した．

Acknowledgements

I would first like to thank my thesis advisor Prof. Honiden and Prof. Ishikawa of the National Institute of Informatics. The door to Prof. Honiden and Prof. Ishikawa office was always open whenever I ran into a trouble spot or had a question about my research or writing. They consistently allowed this paper to be my own work, but steered me in the right the direction whenever they thought I needed it.

I also appreciate Ms. Feng, Ms. Fang and Mr. Cui, help in improving the structure of the paper.

Contents

Chapter 1

INTREODUCTION

With the development and extensive use of the Internet, web services, as the component deployed on it, achieves good amount of encapsulation, loose coupling, and cross-platform. More and more developers publish their own services to the personal site or platforms that are similar to amazon expecting paid or unpaid usage. Thus, web services has been a focus of attention[5]. The main purpose of service-based system is to obtain the connection and integration among the various applications and systems on different platform by constructing a universal, platform and language independent technical layer, where it not only has business software which can respond to complicated situation but also the component that has a single function. However the single web services suffers from a limitation of services such that it has to composite the existing web services to satisfy the needs of users. How to generate the web services composition according to the user input becomes the key issue here.As an effective solution, service compositions are widely used in industry and research in order to create a functional richer loosely coupled service quickly. Generally, service composition has two categories. One is called planning algorithm which there is no workflow template so that it focuses on whether it is able to get a possible workflow, (i.e. a possible suggestion) but ignore the performance of non-functional part(i.e. the QoS). The other is selection algorithm which takes a fixed workflow template as the input and values the optimal QOS. In this paper, we mainly focused on the latter one which is capable to obtain more precious results based on the selection of web services issues.

1.1 Background

Because that the services which involved in the composition generally possess many candidate services with different QoS parameter (for example, execution time, service fee, availability, reliability, security, and reputation and so on[9]) and these candidate services have similar but not the same functional properties as well as different non-functional properties so that it is very challenging to effectively and dynamically composite existing web services to generate a new completive service which satisfy different functional and QoS needs of users. The result of choices of services not only concerns whether the composition of services can be succeed but also has crucial effect on the quality of the service compositions. Thus, using QoS to choose services becomes the key technology to achieve practicability of dynamic Web service compositions. At present, researchers already have done some work on dynamically Web services composition based on QoS. However, most of them only discussed about the case of fixed function requirements as input.

1.2 Motivation

When you provide a fixed functional requirement as input to fulfil your needs (fixed functions and optimal QoS), there is no way to visually determine whether the QoS of the result workflow satisfied your needs (not your functional requirement) is optimal or not. It is possible that because your requirements are too strict so that the workflow with lower performance on functions (but still can fulfil your objective or needs) are not in your search space. Besides, may be the functional requirement you provided is not comprehensive so that the system cannot shows the workflow which actually fulfil your requirements excessively in the search space. And this results in losing the workflow which has better performance on the function as well as the QoS than your own workflow. In other words, your need and the functional requirement you choose may leave a gap. This is unconscious and not able to validate immediately. There are two kinds of gaps:

First gap, choosing too strong function requirements. Assume you are a web service provider who want to composite a service which use s_1 or s_2 in Figure1 as first service of workflow. s_1 is a engine can search any car with higher price(low QoS), and s_2 is a engine can only search Kei car with low price(high QoS). Your service purpose is to help user to rental a car for shopping, which car and Kei car show the same performance on this purpose.If you insist " can search any car " as input part of functional requirements, s_2 would not be searched so you will not find the best solution.

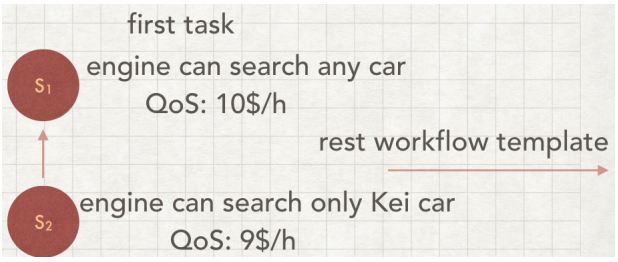

*Figure*1 : *example of service selection*

Second gap, excluding better function from function requirements. It is commonly known that QoS required for multi functional services is lower than that of single functional services (especially reflected in price and reaction time, etc.), or single functional services would lost their users. However, there are many counter-examples. For instance, Amazon is always cheaper than individual online retailers. Meanwhile, products 'quality is ensured by strict supervision. In other words, Amazon possess multi functions and higher QoS.

1.3 Contributions

Chapter 2 introduces the definition of background knowledge of service composition; chapter 3 presents the related work of service composition; chapter 4 declares extended composition problem; chapter 5 gives the algorithm that is able to effectively and swiftly solve the extended composition problem; chapter 6 evaluates the advantage of extended definition introduced in chapter 4 as well as the performance of the optimised algorithm provided in chapter 5; chapter 7 concludes this paper.

1.3.1 EXTENDED COMPOSITION PROBLEM

I defined the extended service composition which expands the search space appropriately to respond to the functional requirement flexible in order to find the services which achieve better QoS and satisfy the user needs.In this paper, I defined 8 kinds of basic variation and 3 kinds of variation degree. I changed the output of the previous systems and defines the Scheme as the output. This provides more choices for the user to enhance their experience.

1.3.2 ALGORITHM FOR EXTENDED COMPOSITION PROBLEM

First, I applied skyline algorithm which is based on the pdfs algorithm to improve the performance of search space and search time comparing to the full search. Besides, in order to solve the extended composition problem, I provided a naive variation algorithm and further enhanced it to achieve the developed variation algorithm which optimised the search space and search time in further step. I also defined methods to evaluate the scheme as well as the step numbers to represent the search space.

1.3.3 Evaluation

In chapter experiment, this paper discuss the significance of the extended problem and the proposed algorithm's capability of improving calculation.

Chapter 2

Preliminary

This section defines some key terms of formal research of the service composition that I continue to use in this paper.

2.1 Service

A service S is a reusable system that provides functionalities which are documented in a service description. This description defines a 5-tuple *IOPEQ = (S.I, S.O, S.P, S.E, S.Q)* where *S.I, S.O* are abbreviated from the required inputs and outputs of the S, *S.P, S.E* are the preconditions and effects which denote the necessary condition of utilising S and the consequence after running S, and *S.Q* is the set of the QoS attributes of the S[6].

In this paper, *S.P, S.E* are ignored therefore a service S is modelled only with 3-tuple *IOQ = (S.I, S.O, S.Q)*.

2.2 QoS

QoS is abbreviated from quality-of-service, that is, quality apart from the functionality the service can provides, such as the price, the execution time and the reliability of the service. The S.Q of a service S is consist of a number of QoS attributes which is normalized between 0 and 1, with 0 being the worst and 1 being the best.

2.3 Service Compliance

If there exists an output $o \in S.O$ of a service S is compatible with an input of $i \in S'.I$ of another service S', we say there is a service link between S and S', written as S $\rightarrow$ S'. That is, the type of o is the same or a subtype of i.

In this paper, a service is constrained to has only one input and one output, therefore S $\rightarrow$ S' iff:

$$o = S.O \ , \ i = S'.I \ , \ o \in i$$

2.4 I/O Plug-in match

Two functionality similar services, in this paper that denotes they are in the same task, may have two relations, defined as follow[4].

An Input Plug-in match $\sqsubseteq$ is a S $\times$ S relation that holds iff:

$$\text{S} \sqsubseteq \text{S'} \Leftrightarrow S.I \subset S'.I$$

In this paper, S $\sqsubseteq$ S' is referred to as S' is input-stronger than S, and S is input-weaker than S'.

An Output Plug-in match $\subset$ is a S $\times$ S relation that holds iff:

$$S \subseteq S' \Leftrightarrow S.I \subset S'.I$$

In this paper, S $\subseteq$ S' is referred to as S' is output-stronger than S, and S is output-weaker than S'.

2.5 Workflow

A workflow is a sequence of two or more linked services. The functions of a workflow depend on function combinations that consist of the input parameter of the workflow's first service and the output parameter of the workflow's last service. A workflow template contains service tasks instead of actual services. A task is characterised as an abstract functionality which can be replaced by an actual service. There are generally two ways to assign services to a task, one is to compare the functions of the services with the functional requirements of the task. The other is to collect services depend on documents such as service description, which are usually distributed with the service by provider. Finally, each task is assigned with a set of services which meet the functional requirements of the task.

Selection algorithms receive a workflow template with fixed functional requirement and a dozens of services, and select for each task of one or more services which guarantee the obtained workflow's QoS are optimised, then return the obtained workflow as an output.

Figure2 shows an example workflow template with service tasks and its corresponding services.

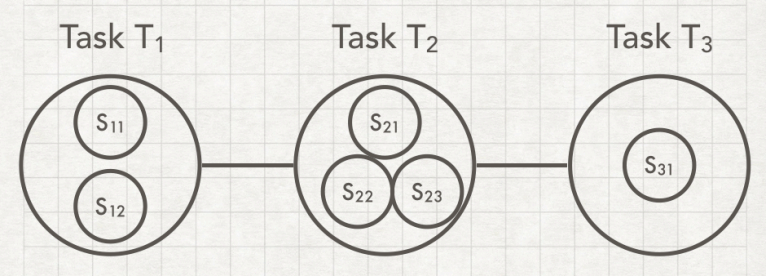

Figure2 : Workflow template with service tasks and its corresponding services

2.6 Functional requirements

Functional requirements can be expressed as below: To a workflow template W, where FT = the first task of W, IT = the last task of W, its functional requirements f can be:

$$f = (i, o), where\ i \subset W.FT.I, o \subset W.IT.O$$

In my experiment, for avoiding meaninglessly setting abstract i and o every time, I equivalently converted it into the definition as shown below:

f = (start services, end services)

where start services = a set of W.FT.services, end services = a set of W.IT.services

2.7 QoS optimization

In this paper, I use the method as shown below to judge the quality of a workflow.

First, QoS vector Q of the workflow is calculated, which computed on the basis of the types of QoS attributes and the control flow of the workflow. Second, value σ is calculated, which is the components of the obtained QoS vector Q are aggregated into a single value σ by applying a weighted sum. At last, service selection algorithm are used to find the workflow which satisfy the constraints with the greatest σ.

2.8 Service composition

service composition is a system to solve composition problem. [7] define the existing composition problem definition as a system whose input is a workflow template, a number of services, function requirements, output is the QoS best workflow and its QoS. However, this definition did not take variation into account.

Chapter 3

Related work

Web service selection based on QoS is a combination optimization question, which is belonged to NP-hard problems. Traditional typical web service selection algorithms based on QoS are mainly exhaustion algorithm and greedy algorithm. Since the exhaustion algorithm to search optimized answer is of very high complexity and the greedy algorithm to search local optimized answer may not retrieve global optimized answer, it is necessary to find effective approximate algorithm to search for approximate solution. Most used web service selection algorithm based on QoS are mainly selective algorithms based on heuristic algorithm.

3.1 Exhaustion Algorithm

The idea of exhaustion algorithm is to list out all possible service combination program and compare their QoS non-functional attributes and find a best optimized combination service to satisfy the need of customers. The advantages to use combination optimization method based on exhaustion algorithm is simple, intuitive and easy to understand. It is also accurate and comprehensive, able to find all solutions. However, its disadvantage is the computation amount being large, less expansibility. Assume a service combination program has n kinds of abstract service, and every kind of abstract service has m service candidates, then the whole executable program number is mn, which means the time complexity of this algorithm is O(m^n). Thus it is obvious that, as the length of service combination and the number of single abstract service candidates growing, the number of executable path candidates blows exponentially. This algorithm costs time and memory to achieve all possible solutions, so is only suitable to small scale service selection situations which has high demand to combination service QoS. It is powerless in front of large scale service combination with low demand for QoS, unable to satisfy complex demand of service selection.

3.2 Greedy Algorithm

The idea of greedy algorithm is to every part of solution step of step, greedily selecting web service with best QoS in all service candidates for every abstract service, not regarding the influence to the QoS of the whole combination service. The precondition is achieving globally optimized solution by progressive local optimized solutions. But for service combination problem, local optimization may not promise achieving globally optimization, generally a barely good solution would be achieved, so the suitable situations for this algorithm is limited. Greedy algorithm is intuitive, the time complexity is O(mn), far lower than that

of exhaustion algorithm. Reference [3] points out that greedy algorithm only executes local search, which will be effective in execution time, but one main disadvantage of this algorithm is that it cannot optimize service execution plan as a whole, and can only promise that single service can satisfy constraints given by the consumer, but not the whole execution plan can satisfy these constraints. Reference [2] comes up with greedy selection algorithm, selecting service with best density. This density represents the whole quality of QoS parameters of this service. However, this algorithm can not satisfy global constraints of combination service either.

3.3 Genetic Algorithm

Genetic Algorithm describes the optimization process of problems as the process of survival of the fittest chromosome. Using iterative evolution of population, implementing operations such as selection, intersection and variation to population, to generate individuals of next generation, gradually evolve toward optimized population, eventually achieving chromosomes the fit the environment the most, which is the most optimized solution or approximate optimized solution to the problem. In fact genetic algorithm is an optimization process using iteration of populations. [10] using this algorithm to solve web service selection problem based on QoS, mainly depends on specific situation of service combination to determine the way to represent chromosome, calculation of fitness function and operations of intersection and variation. Every chromosome represents a specific service combination programs that may come from different abstract service selected, consists of several genes. Every gene corresponds to an abstract service inside the service combination, the ith gene 's value is the index of ith selected abstract service in its corresponding sets of service candidates. The determination of fitness function is calculated based on QoS of every specific service. The intersection operation is interchange two chromosomes(which are two different combination program) using some kind of intersection method, to change some gene values in these two chromosome, to generate two new chromosomes. And variation operation is to randomly change some specific genes in chromosome, which means using another available specific service to substitute corresponding specific service.

Chapter 4

Approach

I introduced the core study of this paper, definition of Extended Composition Problem in the next section.

In the second section, I summerized concrete algorithm solving the problem mentioned above, and its optimized algorithm. Initially, instead of full search, skyline algorithm (introduced in subsection 4.2.1) could reduce run times effectively, thus facilitating feasibility of the whole system evidently. Furthermore, naive algorithm (introduced in subsection 4.2.2) is capable of calculating several basic variations in variation degree respectively in response to customers' choices, on the basis of skyline algorithm. Those calculation results are compared to generate Scheme, thus recommending the most proper results to customers. Next, development algorithm (introduced in subsection 4.2.3) is a modified version of naive algorithm. In this algorithm, basic variations are divided into several groups by different input requirements for Scheme generation. Thus compared with naive algorithm, it could reduce calculation much effectively.

4.1 Extended Composition Problem

4.1.1 Concept of Variation

To a workflow template W, where FT is the first task of W, LT is the last task of W, i is a set of W.FT.services, o is a set of W.LT.services, W's functional requirements f is (i, o), I put forward the concept of Variation:

$$Variation = (i', o'), where\, i' \subset P(Strong(i), Weak(i), i), o' \subset P(Strong(o), Weak(o), o)$$

In addition, P represents powerset. In the next, I am going to introduce the definition of Strong and Weak.

To a service s in the FT(first task) of a workflow template W, Strong(s) is defined as below:

$$Strong(s) = \{s' \in FT, s \sqsubseteq s'\}$$

To a service s in the LT(last task) of a workflow template W, Strong(s) is defined as below:

$$Strong(s) = \{s' \in LT, s \subseteq s'\}$$

To a service s in the FT(first task) of a workflow template W, Weak(s) is defined as below:

$$Weak(s) = \{s' \in FT, s' \sqsubseteq s\}$$

To a service s in the LT(last task) of a workflow template W, Weak(s) is defined as below:

$$Weak(s) = \{s' \in LT, s' \subseteq s\}$$

Function elevation is expected unless QoS were not reduced. If compromise of function were necessary for enhancement of QoS, there must be a minimum for function reduction. In this paper, I took a one-degree weaker model, where function compromise is controlled within an acceptable level into account.

To a set of services S ⊆ FT(first task) of a workflow template W, Strong(S) is defined as below:

$$Strong(S) = \cup(Strong(s \in S))$$

To a set of services S ⊆ LT(last task) of a workflow template W, Strong(S) is defined as below:

$$Strong(S) = \cup(Strong(s \in S))$$

To a set of services S ⊆ FT(first task) of a workflow template W, Weak(S) is defined as below:

$$Weak(S) = \cup(Weak(s \in S))$$

To a set of services S ⊆ LT(first task) of a workflow template W, Weak(S) is defined as below:

$$Weak(S) = \cup(Weak(s \in S))$$

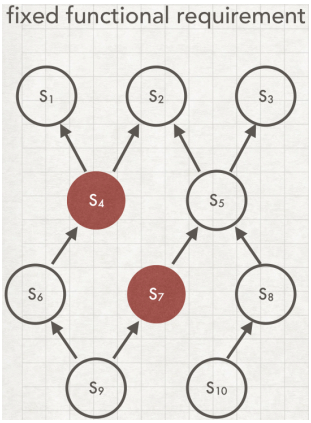

Figure3

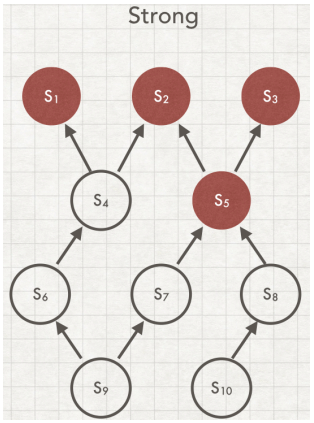

Figure4

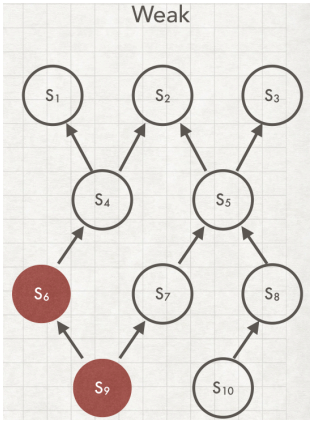

Figure5

To a functional requirement FQ = (i, o), where i = a set of W.FT.services, o = a set of W.LT.services. Figure3,4,5 shows a example of a set s = $\{s_4, s_7\}$, Strong(s) and Weak(s).

4.1.2 Three degrees of Variation

According to previous definition of Strong and Weak, 8 basic variations is acquainted as building blocks for 3 degrees of higher variations.

There are 8 kind of basic Variation:

$$FQ_{(s,s)}, FQ_{(s,f)}, FQ_{(s,w)}, FQ_{(f,s)}, FQ_{(f,w)}, FQ_{(w,s)}, FQ_{(w,f)}, FQ_{(w,w)}$$

s appear in foreparts of tuple represent Strong(i), those in later parts represent Strong(o). w and f could be interpreted in similar ways, with f representing fix, and fix(i) is i, fix(o) is o.

Moreover, I defined 3 degrees of variations for users to select. Users demanding strict functional requirements stronger than their variation are recommended to choose $FQ_{compatible}$. While $FQ_{trade-off}$ is proposed to users demanding balance

between function and QoS. $FQ_{compromise}$ is recommended to users who can tolerate with slight function compromise. The mathematical definitions of 3 degrees of variations are demonstrated here:

$$FQ_{compatible} = FQ \cup FQ_{(s,s)} \cup FQ_{(s,f)} \cup FQ_{(f,s)}$$

where compatible means functional compatible

$$FQ_{trade-off} = FQ \cup FQ_{(s,s)} \cup FQ_{(s,f)} \cup FQ_{(f,s)} \cup FQ_{(w,s)} \cup FQ_{(s,w)}$$

where trade-off means functional trade-off

$$FQ_{compromise} = FQ \cup FQ_{(s,s)} \cup FQ_{(s,f)} \cup FQ_{(f,s)} \cup FQ_{(w,s)} \cup FQ_{(s,w)} \cup FQ_{(f,w)} \cup FQ_{(w,f)} \cup FQ_{(w,w)}$$

where compromise means functional compromise

4.1.3 extended composition problem definition

Hence we could combine the new definition particularised in former section with definition of service composition to acquaint a new definition of service composition.

```
1 input: a workflow template, a number of services, function requirement, the
      degree of Variation
2 procedure: According to the degree of Variation, calculate new function
      requirement, then the new system take each basic Variation in it as input
3 output: each basic Variation 's QoS best workflow and its QoS, then generate a
      Scheme as selection advice to user.
```

First, each basic Variation 's QoS best workflow and its QoS as each node are calculated. Then they will be applied function gen_scheme to generate a Scheme.

Scheme: some local QoS best workflows which belong to different basic Variation

```
1 Input : local best workflows of all basic Variations
2 Output : a Scheme
3
4 def gen_scheme(local best workflows of all basic Variations):
5         refer to graph in Figure6 and Scheme algorithm below
6         for each node:
7             if this node 's QoS is greater than every node it can get to:
8                 add it to Scheme
```

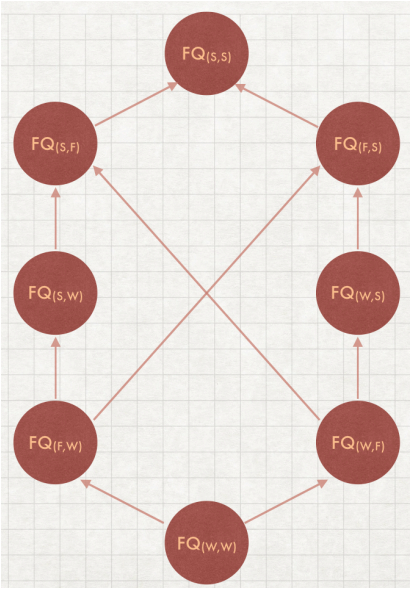

Figure6 : Relationship between each basic Variation

4.2 Algorithm for Extended Composition Problem

I describe concrete solution of former problems in this section.

I took full search based on dfs to search for workflow into consideration first. Full search based on dfs is a simple and directly perceived model with high accuracy and comprehensive capacity for calculation of all possible results. But it has high computational complexity and low scalability, thus considered impracticable for this paper. Therefore, before calculating and verifying extended composition problem, optimisation of full search for lower search space and search time should be taken into account. Then I proposed skyline algorithm, which is capable of curtailing minor details to remain indeed vital information, which refers to services possess either strong points in function or QoS.

Next, I proposed a solution for extended composition problem. That is to calculate each basic variation in degree variations respectively first, and then substitute those results to Scheme for Scheme's naive variation algorithm. However, it actually re-calculated quiet a few space and could be further optimised. Therefore, I proposed developed variation algorithm as a possible solution. That is to group basic variations according to their input requirements first, and then substitute each group to calculation to reduce re-calculated space.

4.2.1 Skyline

I compared full search and skyline algorithm to solve composition problem. I calculated next set of each service, which represents accessible service sets of target service, through i o relationship. Full search service using dfs algorithm is an algorithm departing from all start-services, testing every possible routes. If the last service acquainted by workflow belongs to end service, its QoS will be calculated and compared with current best QoS.

Defects of full search are obvious referring to Figure7. Suppose a certain s_0 in task k with 3 accessible services, s_1, s_2, s_3 in the next task. While each of s_1, s_2, s_3 also has 3 accessible services in the next task, $\{s_4, s_5, s_6\}$, $\{s_5, s_7\}$, $\{s_6, s_7\}$. The ranking of services in task k+1 is defined as $s_2 < s_1 < s_3$. If full search was utilised from task k to k+2, $3 \times 4 = 12$ routes have to be screened, with obvious unnecessary routes calculated. As mentioned in [1], they perform a skyline query on the services in each class to distinguish between those services that are potential candidates for the composition, and those that can not possibly be part of the final solution. The latter can effectively be pruned to reduce the search space. But for they are too complicated for high order QoS vector. In this paper QoS constraint is defaulted, thus QoS value is figured out by adding QoS Vector of each service. Here I simplified and improved the algorithm proposed in previous paper.

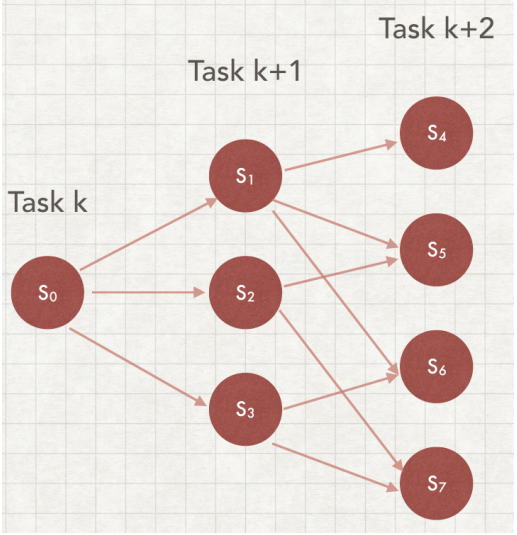

Figure7 : Full search algorithm

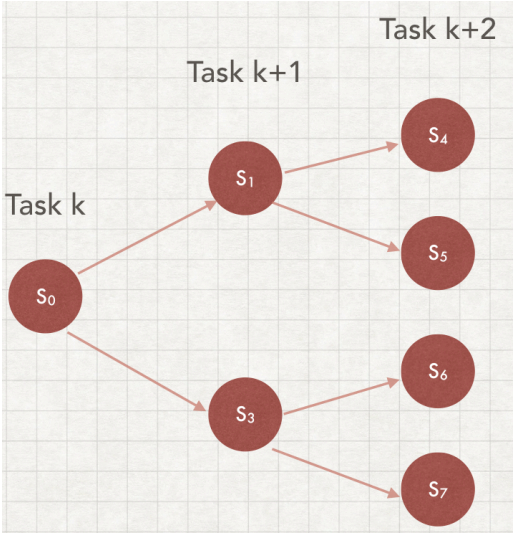

Figure8 : Skyline algorithm

Psudocode of Skyline algorithm and its sub function selection are described as below:

```
1  Skyline algorithm: apply function selection at every search step
2
3  Function selection:
4  Input : service_list
5  Output : ans
6
7  def selection(service_list):
8          ans={}
9          Next = Union(every service in service_list.next)
10         while Next != {}:
11                 s = QoS best in service_list and s.next != {}
12                 add s to ans
13                 service_list = service_list − s
14                 service.next = service.next − s.next
15                 Next = Next − s.next
16                 return ans
```

Advantages of skyline algorithm over full search could be illustrated by analysing the former instance (Figure8). As mentioned above, according to full search algorithm, $\{s_1, s_2, s_3\}$ in task k and its accessible set nextall $\{s_4, s_5, s_6, s_7\}$ in task k+1 is generated. First, service with highest QoS and not-null next set, which refers to s_3 here, is picked up and put into the answer set. s_1 possess the next high QoS and not-null next set and accessible to $\{s_4, s_5, s_6\}$. But s_6 is occupied by s_1, the next set of s_1 is modified to $\{s_4, s_5\}$. Here modified s_1 is put into the answer set as well. Since union set of s_3 and s_1 equals to next all, the search is end up. Thereby search space is pruned to 4 routes, with the same QoS acquainted by full search and no comprise in answer quality.

4.2.2 Naive Variation Algorithm

Next, I describe naive variation algorithm capable pf solving extended composition problem in a direct way. Naive variation algorithm calculates the exact representation of basic variation based on degree variation referred by customers. Then the designated basic variation is substituted into skyline algorithm respectively. Upon acquiring partially optimised workflow, those workflows are substituted into Scheme and exported to users.

```
Naive Variation algorithm:
Input: a workflow template, a number of services, function requirement, the
    degree of Variation
Output: a Scheme as selection advice to user.

for basic_variation in degree of variation:
        local_best_workflow = Skyline(basic_variation)
        put local_best_workflow into temp_ans
        scheme = gen_scheme(temp_ans)
        return scheme
```

4.2.3 Developed Variation Algorithm

Developed Variation Algorithm is an improved version of Naive Variation Algorithm.

```
Developed Variation algorithm:
Input: a workflow template, a number of services, function requirement, the
    degree of Variation
Output: a Scheme as selection advice to user

for basic_variation in degree of variation:
        divide basic_variation in degree of variation into some groups by
            whether containing the same input
        local_best_workflow = Skyline(group)
        put local_best_workflow into temp_ans
        scheme = gen_scheme(temp_ans)
        return scheme
```

Chapter 5

Experiments

In this chapter, we mainly introduce the experiments we do in this study. Chapter 1 introduce the actual runtime and implementation environment. Chapter 2 declares the advantage of setting extended composition problem as well as the testing methods and the evaluation of its. In chapter 3 presents the effort I make to speed up the naive algorithm and the evaluation about how many times it becomes faster by comparing the step number and runtime.

5.1 Implementation

In this experiment, the environment of programming work is python 2.7. The evaluation part is executed with the environment of R 3.2.3. In order to make sure that all the situations can be tested, we make settings as follow. I set 10 tasks linking one by one as a work flow template where each task owns 10 services with different functions and QoS. The QoS of each services contains a vector composted by 10 normalised numbers (between 0 to1). For each of them, which services it will link with from the next task is totally randomised. Thus, the relationship of the input from start task and the output of end task are also randomly decided.

5.2 Extended Composition Problem

In this section, the degree of QoS can be improved by allowing compromise in functional requirements are evaluated. I calculated random service composition problem for 1000 times. 172 cases of invalid composition are excluded. The QoS of three degrees of variation are compared with the QoS of fixed functional requirements. The result is shown as Table 1:

Performance of three Variation degree in 828 valid cases

	Improved scheme numbers	average improvement degree(vs fixed functional requirements)
$FQ_{compatible}$	453	5.089129%
$FQ_{trade\text{-}off}$	485	8.827387%
$FQ_{compromise}$	566	11.80829%

*Table*1

I define Scheme improvement degree as the sum of improvement define of every workflow in the Scheme. From Table 1, we could find that with the increase of tolerance of the degree of variation, the Scheme improvement degree and search space are also increasing.

5.3 Algorithm for Extended Composition Problem

In this section, the improvement of search space and time is evaluated. Table2 depicts the relationship of step numbers and runtime between full search and skyline algorithm. Step numbers is a criteria for search space, that searching from a service to a next service is called one step.

Comparison of the search space and time of Full search and Skyline

	step numbers	**runtime(second)**
Full search	2999026	16.729
Skyline	1974.1	0.5128
Full search/ Skyline	1519.1864647181	32.6228549141966

*Table*2

Comparing with the the performance of full search and skyline algorithm, we could find that latter's search space is 1/1519 of former, and search time is 1/32. Table3 depicts the relationship of step numbers and runtime between naive

variation algorithm and developed variation algorithm.

Comparison of the search space and time of different Variation

	step numbers	runtime(second)
Fixed functional requiremnet	2113	0.569703
Naive Variation algorithm $FQ_{compatible}$	6550	2.017894
Naive Variation algorithm $FQ_{trade\text{-}off}$	8837	2.642321
Naive Variation algorithm $FQ_{compromise}$	13062	3.992750
Developed Variation algorithm $FQ_{compatible}$	4100	0.992360
Developed Variation algorithm $FQ_{trade\text{-}off}$	5456	1.231265
Developed Variation algorithm $FQ_{compromise}$	5867	1.263835

*Table*3

Comparing with the the performance of naive variation algorithm and developed variation algorithm, we could find that latter's search space is 1/2 or 1/3 of former.

Chapter 6

Discussion

This paper discussed proposal in the first section and algorithm for Extended Composition Problem in the second.

6.1 Proposal

In subsection 1 demonstrated significance of the extended problem in actual experiments. And subsection 2 demonstrated the proposed algorithm is capable of improving calculation in experiments.

6.1.1 Extended Composition Problem

Improving the definitions could improve quality of solutions of proposals conspicuously(Table1). As a result, quality of $FQ_{compatible}$ was enhanced by 5.1%, $FQ_{trade-off}$ by 8.8%, $FQ_{compromise}$ by 11.8%. Quality of $FQ_{compromise}$ was enhanced most remarkably. By comparing fixed functional requirement ' s calculating time of 3 variation degree with that of developed variation algorithm, we could find that the latter algorithm spent 1.74, 2.16, 2.22 times of time, respectively, to the former one. Run time of $FQ_{trade-off}$ was similar to $FQ_{compromise}$, but improvement level of the former was lower by 3%, which is possibly result from their same group numbers. Thus $FQ_{compromise}$ is preferred to $FQ_{trade-off}$. However, $FQ_{compromise}$ still has some defects. For the instance mentioned in introduction, when destination of web service producer was to help user to rent a car for shopping, either utilising searching engine s1 (capable of searching any car) or engine s2 (capable of searing kei car only) could achieve the same goal. However, when provider aimed at helping user to rental a car for mountain climbing, results recommended by engine 2 might include kei cars lack of enough power for climbing. If $FQ_{compatible}$ were selected, workflow including s2 would appear in scheme, which is actually incapable of reaching the goal. As a result, $FQ_{compatible}$ and $FQ_{compromise}$ could be regard as two possible options. $FQ_{compatible}$ is capable of fast calculation while quality of results was compromised. While $FQ_{compromise}$ could generate results with higher quality but costing longer time, especially, invalid results deviated from user 's demand might be generated. Thus users still have judge the best choice.

6.1.2 Algorithm for Extended Composition Problem

As described in related, search space of full search equal to total task numbers power of service numbers in the task. Search space of skyline algorithm equal to total task numbers power of services numbers after selection. Step numbers are reduced to 1/1519 by skyline algorithm(Table2). That is, average $\frac{1}{\sqrt[10]{1519}} = 48\%$

of services are remained in every tasks. It also reduces time to 1/32. But it is not so remarkable compared to reduction of step numbers, since extra selection was run during each step.

Search time is in proportion to search space in Table3. Thus search space of naive variation algorithm and developed variation algorithm is compared. Search space of FQ-compatible is 63% of naive variation algorithm. While this ratio change to 62% for FQ trade-off and 45% for FQ-compromise. Thus, developed variation algorithm could save much time than naive variation algorithm with increasing search space.

6.2 Future Work

In this section, we discuss about the shortage of this paper as well as the future work of this study.

6.2.1 QoS constraints

In this paper, we didn't set constraints for QoS which simplify this study to some extend and also make it easier to calculate. However, to tackle more completive and more practical service composition problem in the future, we need to further improve the skyline algorithm[8].

6.2.2 adaptive

In this paper, users have to set the variation degree by themselves. In the future, it is also able to design the system to automatically precept the appropriate variation degree according to the needs of users.

6.2.3 change scability

In order to reduce the computational complexity, we set 10 tasks with 10 services each. In the future, we can control the number of the task but increase the services of each task or keep the number of the services but increase the total number of tasks to test its scalability. It could also be possible to set the number of tasks and services as random number to observe the performance of the workflow under different condition.

Chapter 7

Conclusion

In service composition, since there is a gap between user's objective and functional requirements, which is difficult to detect and confirm, a extended composition problem is needed to deal with it. This paper presents the study of service composition which provides a new definition of composition problem and the algorithm to solve the problem as well as the optimization of the algorithm. Besides, it also provides the result of the search space and search time of the three degree variations and the optimizaton as well. $FQ_{compatible}$ and $FQ_{compromise}$ could be regard as two possible options. $FQ_{compatible}$ is capable of fast calculation while quality of results was compromised. While $FQ_{compromise}$ could generate results with higher quality but costing longer time, especially, invalid results deviated from user 's demand might be generated. Thus users still have to judge the best choice.

References

[1] Mohammad Alrifai, Dimitrios Skoutas, and Thomas Risse. Selecting skyline services for qos-based web service composition. In *Proceedings of the 19th international conference on World wide web*, pages 11–20. ACM, 2010.

[2] Thomas L Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5228–5235, 2004.

[3] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57. ACM, 1999.

[4] Massimo Paolucci, Takahiro Kawamura, Terry R Payne, and Katia Sycara. Semantic matching of web services capabilities. In *The Semantic Web—ISWC 2002*, pages 333–347. Springer, 2002.

[5] Michael P Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: a research roadmap. *International Journal of Cooperative Information Systems*, 17(02):223–255, 2008.

[6] Florian Wagner, Fuyuki Ishikawa, and Shinichi Honiden. Qos-aware automatic service composition by applying functional clustering. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 89–96. IEEE, 2011.

[7] Florian Wagner, Benjamin Klöpper, Fuyuki Ishikawa, and Shinichi Honiden. Towards robust service compositions in the context of functionally diverse services. In *Proceedings of the 21st international conference on World Wide Web*, pages 969–978. ACM, 2012.

[8] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. *ACM Transactions on the Web (TWEB)*, 1(1):6, 2007.

[9] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z Sheng. Quality driven web services composition. In *Proceedings of the 12th international conference on World Wide Web*, pages 411–421. ACM, 2003.

[10] Chengwen Zhang, Sen Su, and Junliang Chen. Diga: Population diversity handling genetic algorithm for qos-aware web services selection. *Computer Communications*, 30(5):1082–1090, 2007.