QoS-aware Service Composition Using NSGA-II[1]

Yujie Yao
Shanghai Jiao Tong University
No. 800, Dongchuan Road,
Shanghai, China
86-13901707409
yaoyujie@sjtu.edu.cn

Haopeng Chen
Shanghai Jiao Tong University
No. 800, Dongchuan Road,
Shanghai, China
86-13917262561
chen-hp@sjtu.edu.cn

ABSTRACT
Web services are rapidly changing the landscape of software engineering. One of the most important technologies introduced by them is Web service composition (WSC). In this paper, we model the Quality of Service (QoS)-aware service composition problem as a multiobjective optimization problem (MOP). A corresponding algorithm that based on Nondominated Sorting Genetic Algorithm-II (NSGA-II) is proposed to solve the problem. The algorithm generates a set of Pereto optimal solutions which satisfy users' requirement. Our experimental results show that by using the proposed algorithm, we can give service consumers several different feasible solutions which make decision easier.

Categories and Subject Descriptors
H.3.5 [**Information Storage and Retrieval**]: Online Information Services - *Web-based services*

General Terms
Algorithms, Experimentation.

Keywords
Web service, service composition, Quality of Service, Nondominated Sorting Genetic Algorithm-II

1. INTRODUCTION
Nowadays, with the rapid pace of change, the need for transformation from today's inflexible business environment to an agile enterprise that can change in productive and cost-effective ways has never been greater. More and more companies shift their focus from software functionality to business processes.

Usually, the function of single Web services is relatively limited in order to enhance reusability. When user requirements can't be satisfied, it is required that we compose multiple Web services to form a whole business process.

Thus, how to efficiently select Web services which have equivalent functionality is becoming more and more important. Web services have different QoS attributes. Some of them excel in the performance of response time while others are good at throughput capacity. Also, a service in high quality may charge for a higher price. We have to balance those QoS factors in a proper way.

In this paper, we model the QoS-aware service composition problem as a multiobjective optimization problem (MOP). Then, a corresponding service composition algorithm that based on NSGA-II is proposed.

This paper is organized as follows. Section 2 shows the related work. Section 3 introduces some background knowledge about MOP and NSGA-II. Section 4 elaborates our approach. Section 5 focuses on the implementation of our composition algorithm and the experimental results. Finally, Section 6 concludes this paper.

2. RELATED WORK
In recent years, a variety of approaches have been proposed to tackle the QoS-aware service composition problem. [1] discusses the QoS-aware selection and the detail of the traditional genetic algorithm's implementation. [2] also implements the traditional genetic algorithm and compares the performance of static and dynamic fitness approach. It also addresses that genetic algorithm outperforms integer programming if there are many abstract and concrete services. [3] presents a 3-layer Web service organization model to help getting executable composite service with well matching and compares dynamic programming and genetic algorithm.

Thus, the above approaches can provide only a single optimal solution to users. Sometimes it does work, but it may happen that the solution is not the appropriate one and the user hasn't any other choices. Also, the quality of solutions highly depends on the design of the aggregate functions.

The utility function technology which most approaches use unites multiple QoSs into a single attribute. It is based on the

1 This paper is supported by the National High-Tech Research Development Program of China (863 program) under Grant No. 2007AA01Z139.

following assumptions. One is that QoSs have relationships between each other. The other is that the QoS weight service consumers specified is precise and stable. However, it's not the case at all. Usually, relationship is not guaranteed and the weight could be arbitrary and inaccurate.

3. BACKGROUND

3.1 Multiobjective Optimization Problem

Literally speaking, a MOP has multiple objectives to be optimized. Those objectives are also formally called decision variables.

Definition 1: Decision Variable

The decision variables are the numerical quantities for which values are to be chosen in an optimization problem. These quantities are denoted as x_j, $j=1,2,...,n$. The vector x of n decision variables is represented by: $x=[x_1,x_2,...,x_n]^T$, where T indicates the transposition of the column vector to the row vector. [4]

Definition 2: Multiobjective Optimization Problem (MOP)

A MOP can be defined as the problem of finding a vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other. The term "optimize" means finding such a solution which would give the values of all the objective functions acceptable to the decision maker. [4]

3.2 Nondominated Sorting Genetic Algorithm-II (NSGA-II)

"Nondominated Sorting Genetic Algorithm" (NSGA) was first proposed by N.Srinivas and K.Deb [5]. The algorithm modifies the ranking procedure originally proposed by Goldberg [6]. The NSGA approach wasn't an efficient algorithm for its: (1) high computational complexity of nondominated sorting; (2) lack of elitism; (3) the need for specifying a sharing paremeter. [7,8]

Several years later, K.Deb et al. [7,8] proposed an improved version, NSGA-II, which is based on the original design of NSGA. Because of NSGA-II's low computational requirements, elitist approach, and parameter-less sharing approach, it is able to find much better spread of solutions.

NSGA-II builds a population of competing individuals, ranks and sorts each individual according to nondomination level, applies crowded-comparison operator to create new pool of offspring, and then combines the parents and offspring before partitioning the new combined pool into fronts. The NSGA-II then conducts niching by adding a crowding distance to each member. It uses this crowding distance in its selection operator to keep a diverse front by making sure each member stays a crowding distance apart. This keeps the population diverse and helps the algorithm to explore the fitness landscape.

Figure 1 shows the basic cycle of NSGA-II.

4. SOLVING THE WEB SERVICE COMPOSITION PROBLEM

In this section, a QoS evaluation model is first discussed to model user requirements. Then we formalize the WSC problem and a corresponding composition algorithm is put forward.

4.1 QoS Evaluation Model

In this section, we propose a QoS evaluation model. It takes consideration of QoS attributes for single service and composite services. We describe service composition models. For each model, we provide an aggregate function which calculates overall QoS values.

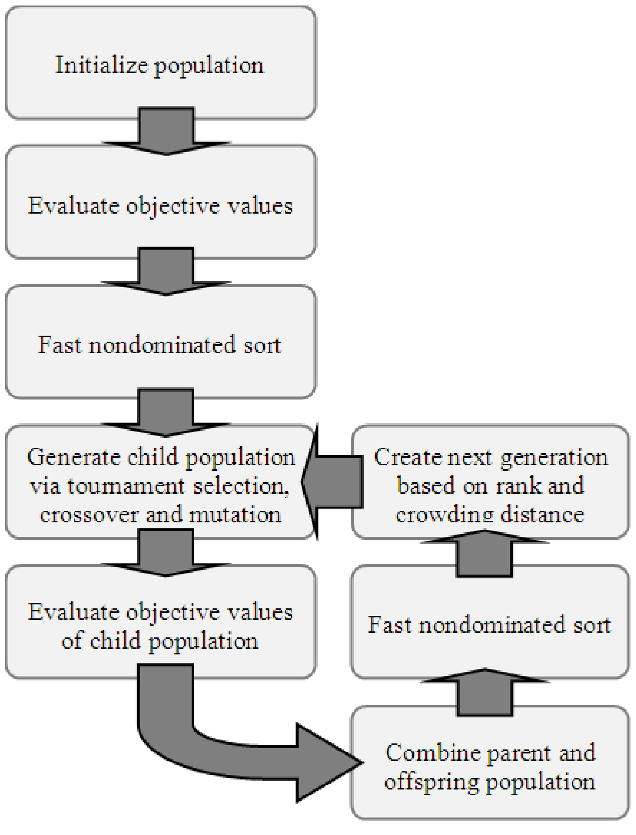
Initialize population
Evaluate objective values
Fast nondominated sort
Generate child population via tournament selection, crossover and mutation
Create next generation based on rank and crowding distance
Evaluate objective values of child population
Fast nondominated sort
Combine parent and offspring population

Fig.1. Basic Cycle of NSGA-II

4.1.1 QoS Evaluation Model for Single Service

In this paper, we consider five major QoS attributes which are price, response time, availability, throughput, and reputation. Detailed definitions list as follows.

- Price: the amount of money that the service consumer has to pay to the service provider for using a web service *s*. It is denoted as $Q_{pr}(s)$.
- Response Time: the time interval of a service *s* from request to response. It is denoted as $Q_{resp}(s)$.
- Availability: the percentage of time that a service *s* is operating. It is denoted as $Q_{av}(s)$.

- Throughput: the rate at which a service *s* can process requests. It is denoted as $Q_{th}(s)$.
- Reputation: the measure of the trustworthiness of a service *s*. It is denoted as $Q_{rep}(s)$.

To sum up, the QoSs of a single Web service can be defined as the vector of five decision variables. The vector is represented by:

$QoS(s)=[Q_{pr}(s), Q_{resp}(s), Q_{av}(s), Q_{th}(s), Q_{rep}(s)]^T$,

where *T* indicates the transposition of the column vector to the row vector.

4.1.2 *QoS Evaluation Model for Composite Service*

As for composite services, in addition to consider the QoS attributes of each individual service, the composition flow models and corresponding aggregate functions should also be considered.

There are four different ways that individual Web services can be integrated to form a business process, which are sequential, parallel, conditional, and loop. Figure 2 shows these four basic models.

For a composite service, the overall QoS can be defined as the vector which is represented by:

$QoS'(S)=[Q'_{pr}(S), Q'_{resp}(S), Q'_{av}(S), Q'_{th}(S), Q'_{rep}(S)]^T$,

where *S* is a single path composite service; $QoS'(S)$ indicates the overall quality of a composed service *S*. Each dimension is the aggregation of all services which is calculated by the aggregation functions in Table 1.

$Q'_{pr}(S)$ is the total price of *S*; $Q'_{resp}(S)$ is the total response time; $Q'_{av}(S)$ is the overall availability; $Q'_{th}(S)$ is the overall throughput; $Q'_{rep}(S)$ is the overall success ratio.

4.2 Modeling and Formalization Web Service Composition Problem

According to the proposed QoS evaluation model for composite web service, we can go a step further to model and formalize the WSC problem.

The WSC is the problem whose goal is to choose one or more service execution plans that have a good performance on most of the QoS factors and meet user requirements.

Its workflow is composes of *n* abstract service nodes, which are represented by $s_1, s_2,..., s_n$. According to the query information of i_{th} abstract service node s_i, m_i candidate concrete services are found, which are represented by $s_{i-1}, s_{i-2},..., s_{i-mi}$.

Solutions should minimize (or maximize) the components of a vector $QoS(S)=[Q_1(S),Q_2(S),...,Q_k(S)]^T$, where *S* is an n-dimensional decision variable vector $S=[S_1,S_2,...,Sn]^T$ from decision space Ω, and $Q_i(S)$ is the aggregation value of the whole workflow. A typical execution plan *S* may look like $S=[s_{1-2}, s_{2-4},..., s_{n-3}]$.

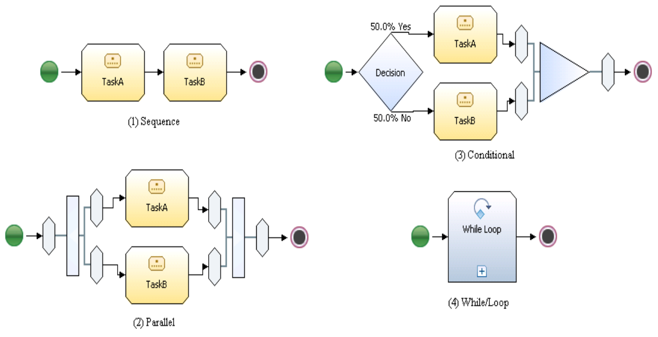

Fig.2. Composition Flow Models

Table.1. Aggregation Functions

	Sequential	Parallel	Conditional	Loop
Price	$\sum_{i=1}^{n} Q_{pr}(i)$	$\sum_{i=1}^{n} Q_{pr}(i)$	$\sum_{i=1}^{n}(P(i)\times Q_{pr}(i))$	$k\times Q_{pr}$
Response Time	$\sum_{i=1}^{n} Q_{resp}(i)$	$\max_{i=1...n}(Q_{resp}(i))$	$\sum_{i=1}^{n}(P(i)\times Q_{resp}(i))$	$k\times Q_{resp}$
Availability	$\prod_{i=1}^{n} Q_{av}(i)$	$\prod_{i=1}^{n} Q_{av}(i)$	$\prod_{i=1}^{n}(P(i)\times Q_{av}(i))$	$Q_{av}{}^{k}$
Throughput	$\min_{i=1...n}(Q_{th}(i))$	$\min_{i=1...n}(Q_{th}(i))$	$\sum_{i=1}^{n}(P(i)\times Q_{th}(i))$	Q_{th}
Reputation	$\frac{\sum_{i=1}^{n} Q_{rep}(i)}{n}$	$\frac{\sum_{i=1}^{n} Q_{rep}(i)}{n}$	$\sum_{i=1}^{n}(P(i)\times Q_{rep}(i))$	Q_{rep}

5. ALGORITHM IMPLEMENTATION AND RESULTS

In this section, we discuss in detail about each part of the algorithm and our modification. Finally, we analyze the results.

5.1 Algorithm Implementation

5.1.1 *Definition of Chromosome*

For genetic algorithms, one of the key issues is to encode a solution of the problem into a chromosome. In our model, the chromosome is encoded by an integer array with a number of items as Figure 3 shows. The sample chromosome consists of 7 genes that correspond to the 7 abstract service nodes. Each gene represents a single selection. For example, the 4_{th} gene “4” means that the 4_{th} candidate service is selected for the 4_{th} service node. Genes range from 1 to m_i, where *i* is the number of abstract service node.

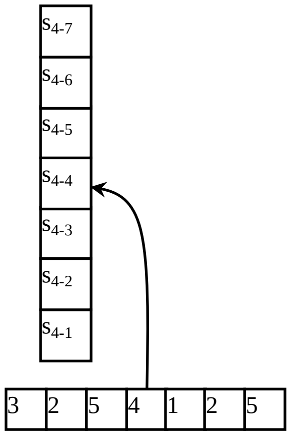

Fig.3. A Sample Chromosome

5.1.2 *Produce the Initial Population*

The size of population indicates the search ability of the algorithm. The bigger the population size is, the more dispersion the solution is. Small size of the population may cause the quick convergence at early runs and hence limit the search capability. In this paper, we define the population size 120.

5.1.3 *Genetic Operators*

Genetic operator consists of three basic operators, i.e., selection, crossover and mutation.

- Selection Operator

We use the binary tournament selection as the selection operator because it can obtain better result than the methods of proportional and genitor selection. The binary tournament selection runs a tournament between two individuals and selects the winner. An individual A wins another individual B only in two circumstances. One is that A's rank is higher than B. The other is A's rank is equal to B and A's distance is higher than B.

- Crossover Operator

We use the single-point crossover operator in our experiment. The crossover point is a random value from 1 to Nt (the number of genes in one chromosome).

We can see the crossover process from Figure 4.

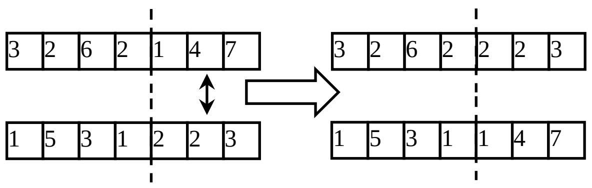

Fig.4. Crossover Operator

- Mutation Operator

It is worth noting that the large mutation rate can lead to loss of good solutions and damage the Pareto optimal solution, so mutation rate should be diminished according to the generation count. In this paper, we define the mutation rate 1%.

We can see the crossover process from Figure 5.

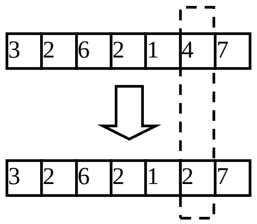

Fig.5. Mutation operator

- Pseudocode

The pseudocode for generate new population is shown in Figure 6.

We use the binary tournament selection, crossover, and mutation operator to form the whole procedure.

```
1. create two chromosome arrays $a_1$, $a_2$ of size
PopulationSize with the data of the last generated
chromosomes
2. shuffle orders of $a_1$, $a_2$
3. for $i \leftarrow 1$ to PopulationSize ($i$ increase by 4 during each
iteration)
    4. do choose $parent_1$ from binary tournament of $a_1[i]$ and
    $a_1[i+1]$
    5. choose $parent_2$ from binary tournament of $a_1[i+2]$ and
    $a_1[i+3]$
    6. do one point crossover between $parent_1$ and $parent_2$
    and 2 new chromosome are generated
    7. choose $parent_3$ from binary tournament of $a_2[i]$ and
    $a_2[i+1]$
    8. choose $parent_4$ from binary tournament of $a_2[i+2]$ and
    $a_2[i+3]$
    9. one point crossover between $parent_3$ and $parent_4$ and 2
    new chromosome are generated
    10. mutation
```

Fig.6. Pseudocode of Generate New Population

- A Modification

As the entire solution space is limited on the basis of the number of tasks and the max value of the number of services, the algorithm will convergent at early generations if a service is doing crossover with itself. So, we add a filter to remove duplication at the end of each generation.

5.1.4 *Evolvement*

The pseudocode of the evolvement is shown in Figure 7. Detailed algorithm of fast nondominated sort procedure and calculate crowding distance procedure can be referred to [7,8].

```
1. for iGen ← 0 to GenerationSize
    2. do aggregate QoS values
    3. fast nondominated sort
    4. calculate crowding distance
    5. calculate fitness value
    6. generate new population
    7. choose elites to next generation
```

Fig.7. Pseudocode of Evolvement

5.2 Experimental Results

In this section, the performance and effectiveness of the proposed algorithm is tested by the experiment.

Experimental related parameters:
Number of tasks: 10;
Number of services for each task: 10;
Number of QoSs for each service: 5;
Size of population: 120;
Size of generation: 100;
Crossover probability: 0.95;
Mutation probability: 0.01.

System environment related parameters:
CPU: Intel Core2 6300 1.86GHz×2
RAM: 2GB
OS: Microsoft Windows XP
JDK: Java SDK 1.6

Monitored and Calculated parameters:
Price
Response time
Availability
Throughput
Reputation
Average deviation of chromosomes
Average fitness of chromosomes
Best chromosome results

In our experiment, we calculate the aggregated value of each QoS separately by applying aggregation functions in Table 1 and normalize them in order to prevent a factor largely dominates others.

The "Average fitness of chromosomes" of NSGA-II is shown in Figure 8. From the results, we learn that during the iteration, the solutions become better and better.

We have tried another experiment in which the number of QoSs for each service is set as 2. Figure 9 shows the QoS tendency of the new experiment. We learn from the figure that the distribution is become wider.

As the distance of the chromosome means its similarity degree compared to other chromosomes, we consider the solutions with large distance value to be special. At the last generation, 10 individuals have the infinity distance value meaning that they are outstanding at one or more QoS attributes. After removing five solutions containing one or more bad attributes (< 0.4), we get five optimal solutions. After normalization, their QoS attributes are shown in Figure 10. The character that each solution has one or more particularly good QoS attribute can meet user's preference information very well.

Additionally, we compare the traditional GA and NSGA-II approaches. We implemented traditional GA which uses the fitness function technology while NSGA-II uses the multiobjective optimization concept and feed them with the same WS data and the initial population to compare their evolvement.

The "Average deviation" are shown in Figure 11 that NSGA-II greatly slows down the convergent speed compared to traditional GA. It means that NSGA-II has a wider distribution and service consumers will have more different choices. The high deviation value of the final generation also indicates that it is composed of lots of nondominated solutions.

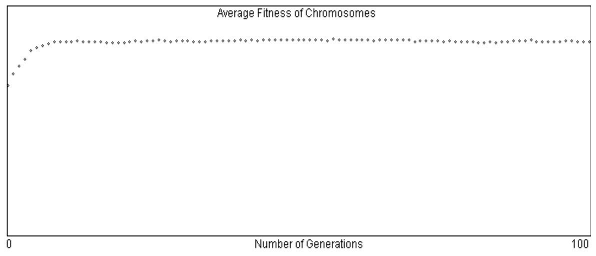

Fig.8. Average Fitness of Chromosomes

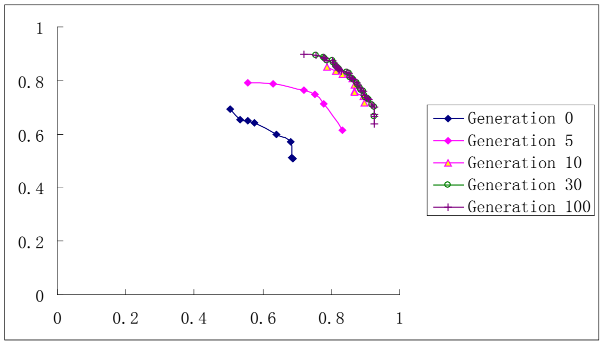

Fig.9. QoS Tendency of 2 QoSs

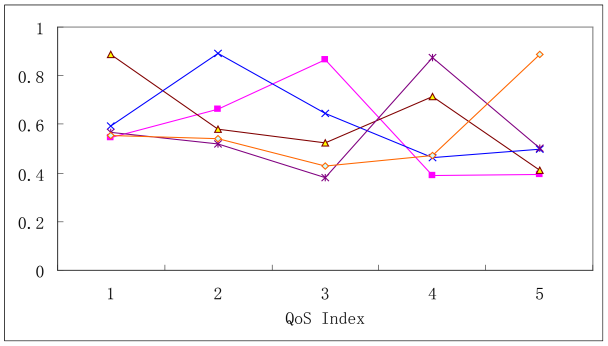

Fig.10. Five Best Results

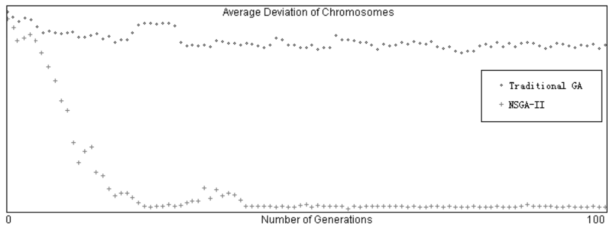

Fig.11. Comparison between Traditional GA and NSGA-II (Average Deviation)

6. CONCLUSIONS

This paper presents a method to solve the Web service composition problem. We present an approach to apply NSGA-II algorithm to get Pereto-optimal solutions.

A set of experiments are conducted to analyze and evaluate the performance of the proposed algorithm, and the experimental results show that our proposed algorithm can slow down the convergent speed, and generate a set of Pareto optimal solutions within a certain generation.

In the future, we will work on the efficiency improvement.

7. REFERENCES

[1] M. C. Jaeger and G. Mühl. QoS-based Selection of Services: The Implementation of a Genetic Algorithm. In Conference on Communication in Distributed Systems, Workshop on Service-Oriented Architectures and Service-Oriented Computing, March 2007.

[2] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An Approach for QoS-aware Service Composition based on Genetic Algorithms. In Genetic and Evolutionary Computation Conference, June 2005.

[3] Y. Gao, B. Zhang, J. Na, L. Yang, Y. Dai, and Q. Gong. Optimal Selection of Web Services with End-to-End Constraints. In IEEE Int'l Conf. on Grid and Cooperative Computing, October 2006.

[4] C.A. Coello and G.B. Lamont, D.A.V. Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, Springer-Verlag New York, Inc. 2006.

[5] N. Srinivas and K. Deb, Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms, Evolutionary Computation, 2(3):221–248, 1994.

[6] N. Srinivas and K. Deb, Multiobjective optimization Using Nondominated Sorting in Genetic Algorithms, Technical report, Department of Mechanical Engineering, Indian Institute of Technology, Kanput, India, 1993.

[7] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan, A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II, In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, Proceedings of the Parallel Problem Solving from Nature VI Conference, pages 849–858, Paris, France, 2000. Springer, Lecture Notes in Computer Science No. 1917.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II, IEEE Transactions on Evolutionary Computation, 6(2):182–197, April 2002