Composing Services of Different Granularity and Varying QoS Using Genetic Algorithm

Twittie Senivongse and Nitirojht Wongsawangpanich

Abstract—**Service composition is a way of constructing new services from existing services and is considered a multi-objective optimization problem since it is concerned with which instances of the services should compose an optimal composition solution, often with regard to multiple quality of service (QoS) requirements. This paper addresses two composition problems: (1) Any service instance whose size of capability, or granularity, does not fit exactly any service in the composition plan will not be considered in the composition even though it has good QoS which can contribute to the quality of the composition solution. (2) QoS of service instances may vary due to conditions and time of use, and hence a composition that is optimal at one time may not be at another. To tackle these problems, we propose how to use a genetic algorithm to discover a composition solution by taking into account service instances of different granularity. Moreover artificial neural network is used to predict the QoS, i.e., response time, reliability, and availability, of the service instances so that the algorithm can discover the composition with the best predicted overall QoS. Experiments show that QoS prediction is useful since composition at different time periods yields different composite services. Also, considering service granularity can increase the opportunity to obtain composite services of good QoS.**

Index Terms—**Service composition, service granularity, QoS, genetic algorithm, artificial neural network.**

I. Introduction

Service composition is a way of constructing new services from existing services by defining a group of service types or abstract services that are to collaborate, and selecting instances of those abstract services to form concrete composition plans or composite services. Researches in this area consider composition as a multi-objective optimization problem and use several optimization algorithms to determine a group of service instances that constitute the optimal solution for the abstract plan, especially with regard to multiple quality of service (QoS) requirements. Given a great number of service instances and hence a great number of potential compositions, genetic algorithms (GA) and their modifications are attractive and commonly used since they can efficiently discover an estimate of the optimal composition without having to consider all possible solutions [1]-[6].

Manuscript received May 28, 2011.

T. Senivongse is with the Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330 Thailand (phone: +66 2 2186996; fax: +66 2 2186955; e-mail: twittie.s@chula.ac.th).

N. Wongsawangpanich is also with the Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330 Thailand.

This paper addresses two problems regarding service composition using GA. First, any service instance whose granularity (or size of capability) does not fit exactly any abstract service will not be considered in the composition. To elaborate on this point, we first discuss a basic use of GA to compose services. Fig. 1 shows a composition scheme which is a specification of abstract services AS_1 to AS_n which are required in a composition. Each abstract service has several service instances as candidates to be selected in composition, e.g., AS_1 has three candidates - SI_{11}, SI_{12}, and SI_{13}. GA can randomly generate possible solutions (or chromosomes) for the composition scheme, evaluate the quality (or fitness) of the chromosomes, and further generate offspring chromosomes from good parent chromosomes, by using operators such as mutation and crossover. The GA process is repeated until a chromosome with satisfactory fitness is found; it becomes the solution to the composition problem.

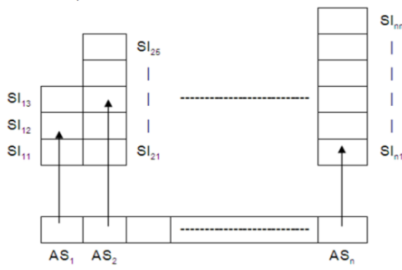

Fig. 1. Composition scheme and service instances (adapted from [1]).

The chromosomes are often encoded with binary encoding. Suppose a composition scheme comprises three abstract services AS_1, AS_2, and AS_3. AS_1 has three service instances (SI_{11}, …, SI_{13}), AS_2 has five service instances (SI_{21}, …, SI_{25}) and AS_3 has six service instances (SI_{31}, …, SI_{36}). AS_1 would require two bits to encode three possible service instances, whereas AS_2 and AS_3 both need three bits to encode five and six service instances respectively. Therefore every possible chromosome in this composition is eight-bits long. As an example, a chromosome representing a composition plan that is composed of SI_{12}, SI_{23}, and SI_{31} would be encoded as 10011001. That is, for a particular composition scheme, the chromosomes are of the same length.

However, some service instances may have granularity that does not fit exactly any of the abstract services in the scheme. They can be problematic in the generation of offspring chromosomes. Assume that a service instance $SI_{[2,3]1}$ has coarse granularity as it is capable of the tasks of both abstract services AS_2 and AS_3. Including it in a chromosome, e.g. with SI_{11}, means this chromosome complies with a different scheme which comprises only two abstract services AS_1 and $AS_{[2,3]}$. It is likely that the chromosomes under this new scheme are not eight-bits long; if AS_1 has three service instances, (requiring two bits) and $AS_{[2,3]}$ has one service instance (requiring one bit), the length of the chromosomes under this scheme is then three bits. It would be problematic to apply mutation and crossover operators to parent chromosomes of different schemes or different lengths, e.g. eight-bit vs. three-bit chromosomes. On the one hand, excluding such a coarser-grained service instance like $SI_{[2,3]1}$ from the composition prevents such a problem in the GA process. On the other hand, leaving it out may reduce the opportunity to obtain good chromosomes since the coarse-grained instance may have good QoS which can contribute to the quality of the composition solution. Using $SI_{[2,3]1}$, for example, may result in better response time than calling two separate service instances of AS_2 and AS_3.

The second problem addressed by this paper is that QoS of service instances may vary due to conditions and time of use. A number of clients, network conditions, and service maintenance are common reasons for the variability. Therefore a chromosome which is found optimal at one time may not be at another since the QoS of its constituent service instances changes over time.

To answer to the problems above, we propose an approach to use GA for service composition when service instances are of different granularity. In particular, the focus is on the case of service instances having granularity coarser than abstract services. Artificial neural network is used to predict the QoS of candidate service instances so that GA can discover a composition solution with the best predicted overall QoS for the time of use. Here we assume that a composite service, once being composed to fulfill a business process of an organization, is likely to be used again at a certain time. Therefore QoS prediction based on the past behavior of the service instances should be beneficial.

Section II discusses research work related to this paper. Section III gives the detail of our service composition methodology with regard to different service granularity and varying QoS. Experiments on service composition are presented in Section IV and results in Section V. Section VI concludes the paper with future outlook.

II. Related Work

Several optimization algorithms have been used to tackle service composition problems. GAs, in particular, have been proposed by many researches due to their ability to search efficiently for an estimate of the optimal solution within the solution space. The attempt by Canfora *et al.* [1] is among the first that shows the power of GA with binary encoding over integer programming when composing services based on time, cost, reliability, availability, and custom attributes; GA is more scalable and suitable to handle generic QoS, and its solution has good quality also. Several researches follow this approach but propose modifications to simple GA or consider additional QoS such as successful execution rate, integrity, and reputation. Amiri and Serajzadeh [2] propose modifications to the crossover operator and the selection of parent chromosomes to escape from local optima. Li *et al.* [3] propose a pseudo-parallel GA with a relation matrix encoding mode of chromosomes which, unlike one-dimension encoding such as binary encoding, can express all composite paths and support service re-planning. Liu *et al.* [4] use ant colony algorithm to improve the generation of GA's initial population chromosomes. Pichanaharee and Senivongse [5] use the estimation of distribution algorithm to improve the generation of new offspring chromosomes in the GA process. They also propose QoS-based service provision schemes by which QoS guarantee is based on levels of service, dependency among different kinds of QoS, and partnership with other services. Similar idea is presented by Tang and Ai [6] where composition using GA considers dependency constraints and conflict constraints between service instances within the chromosomes. Nevertheless, these modifications to GA have not considered the service granularity issue.

On QoS variability, some researches, such as [7], present a framework to handle QoS monitoring and service re-binding when the actual QoS of the service instances in the composition plans deviates. Another approach would be to adopt QoS prediction based on the actual QoS in the past. Gao and Wu [8] supports service selection by using artificial neural network to predict execution duration and transaction state of a single service invocation, given the following inputs – reliability, availability, bandwidth, and request time. We adopt this QoS prediction approach but will apply to the composition problem since the composition which considers the past behavior of service instances, rather than their QoS values published by the providers, should give a more realistic solution.

In summary, the comparison between our approach and related work is shown in Table I.

TABLE I
COMPARISON WITH RELATED WORK

Paper	QoS-Based Composition	GA or Its Modification	Services of Different Granularity	QoS Prediction
[1]-[7]	y	y	n	n
[8]	n	n/a	n/a	y
Our paper	y	y	y	y

III. Service Composition Methodology

The composition methodology is depicted in Fig. 2. Each step is explained as follows, along with some details of supporting software.

A. Define Flow of Abstract Services and Identify Service Instances

In this step, the flow of abstract services called the initial composition scheme is defined. Each abstract service fulfills an atomic task within the flow. Then, candidate service instances of different granularity are identified; service discovery mechanisms or service search engines can help with this matter. We consider the instances whose granularity matches any single abstract services or any groups of abstract services. Fig. 3 shows the initial composition scheme, comprising *n* abstract services, and their matching service instances. For example, $SI_{[1,1]1}$ can fulfill $AS_{[1,1]}$ whereas $SI_{[1,2]1}$ can fulfill $AS_{[1,1]}$ and $AS_{[2,2]}$ altogether.

B. Monitor QoS of Service Instances

All service instances are called periodically by our monitoring software, written in Java, in order to obtain their QoS data, i.e., response time, reliability, and availability. The following data are recorded respectively: request time, response time in seconds (i.e. reply time - request time), reliability status if the correct reply is returned within an expected time frame (1 = reliable, 0 = unreliable), and availability status when the call is made (1 = available, 0 = unavailable). An example of the data recorded on one call to a service instance is <1299873421098,78,1,1>.

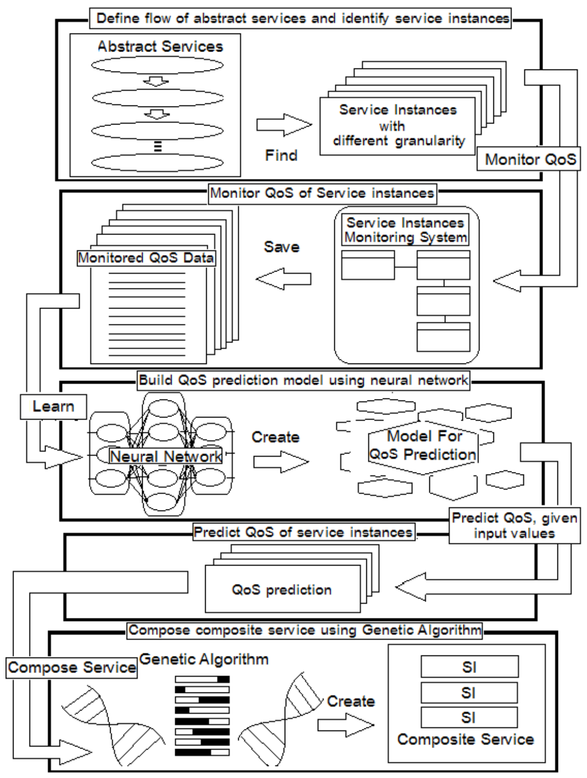
Define flow of abstract services and identify service instances
Abstract Services
Find
Service Instances with different granularity
Monitor QoS
Monitor QoS of Service instances
Monitored QoS Data
Save
Service Instances Monitoring System
Learn
Build QoS prediction model using neural network
Neural Network
Create
Model For QoS Prediction
Predict QoS, given input values
Predict QoS of service instances
QoS prediction
Compose composite service using Genetic Algorithm
Compose Service
Genetic Algorithm
Create
SI
SI
SI
Composite Service

Fig. 2. Service composition methodology.

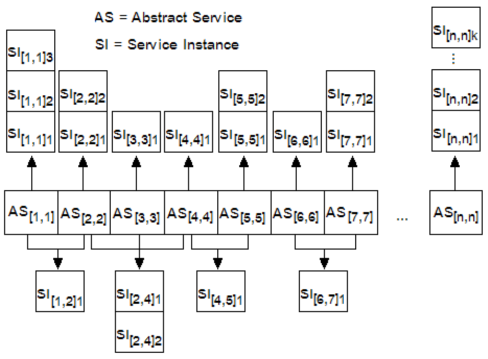

Fig. 3. Composition scheme and service instances of different granularity.

C. *Build QoS Prediction Model Using Neural Network*

Artificial neural network [9] is employed to predict QoS values of service instances to be used later in composition. Specifically, we use the backpropagation neural network to build the mathematical model for QoS prediction. The use of neural network comprises the learning phase and prediction phase. In the learning phase, the monitored QoS data of each service instance from Section III.B are used to train the neural network to obtain a model that, given the input data (i.e., the request time), can generate the output (i.e., response time, reliability, or availability of the service instance). Fig. 4 depicts a neural network with four layers of hidden nodes that we use. The error between the output of the model and the target data is propagated backward through the network to adjust the weights on the network nodes and hence adjust the model to better map the input to the output. The procedure is repeated for each entry of the training data until the error is small enough. Ten-fold cross validation is also used to determine the fit of the model. We use the Weka framework [10] to develop a Java program to build the neural network.

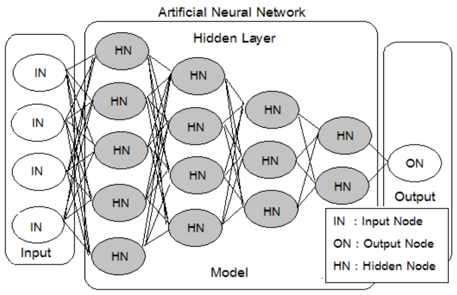

Fig. 4. Artificial neural network.

D. *Predict QoS of Service Instances*

In the prediction phase, we predict the QoS output for each service instance with regard to the request time input (hour, minute, second). That is, there are three models for each service instance – response time, reliability, and availability models.

E. *Compose Composite Service Using GA*

In this step, GA is used to discover a composite service with the best overall QoS at a specified time of use. As mentioned in Section I, the length of the chromosomes in GA is fixed, but when the granularity of some service instances is coarser than any abstract service in the initial composition scheme, other composition schemes emerge and the length of the chromosomes could vary, causing a problem for GA.

To overcome this, we propose a new way to encode and generate chromosomes in order to handle the case of coarse-grained service instances. We define the length of the chromosomes to be the longest which could accommodate the most fine-grained service instances:

$$CL = \frac{ASL}{SIL} \tag{1}$$

where *CL* is the chromosome length, *ASL* is the number of abstract services or atomic tasks in the initial composition scheme, and *SIL* is the number of abstract services which the most fine-grained service instances could fulfill. Suppose an initial composition scheme consists of seven abstract services $AS_{[1,1]}, \ldots, AS_{[7,7]}$ ($ASL = 7$) with relevant service instances as in Fig. 3. The number of abstract services which the most fine-grained service instances (i.e., $SI_{[1,1]k}, \ldots, SI_{[7,7]k}$) could fulfill is 1 ($SIL = 1$). Thus the chromosome length *CL* is 7.

The rest of this section describes the GA process.

1) *Generate Initial Population*

GA randomly generates a number of chromosomes as the initial population. As many service instances as the chromosome length are picked randomly for a chromosome, e.g., for a chromosome of length 7, seven service instances are picked such as Fig. 5(a).

2) *Find Valid Composition Plans*

The chromosomes in *1)* may not yet represent valid composition plans. In this step, find valid plans by grouping the service instances that altogether can fulfill the tasks within the initial composition schemes. Fig. 5(b) shows two valid plans obtained from the chromosome in Fig. 5(a). Note that random selection of service instances in *1)*, in effect, randomly selects other possible schemes for GA to consider.

3) Evaluate Quality of Composition Plans

In this step, each valid composition plan in *2)* has its overall QoS evaluated based on the predicted QoS values of each composing service instances obtained from Section III.D. We adopt the aggregation functions per flow construct and QoS attribute from [1], as shown in Table II. The QoS of a flow construct aggregates the QoS of the tasks $t_1,\ldots, t_m$ within the construct. Note that p_{ai} is the probability that one of the n cases of Switch is chosen, and k is the number of Loop iterations; they both are defined by the composite service designer.

TABLE II
QOS AGGREGATION FUNCTIONS [1]

QoS	Sequence	Switch	Fork	Loop
Response Time (T)	$\sum_{i=1}^{m} T(t_i)$	$\sum_{i=1}^{n} p_{ai} * T(t_i)$	$Max\{T(t_i)_{i\in\{1\ldots p\}}\}$	$k * T(t)$
Reliability (R)	$\prod_{i=1}^{m} R(t_i)$	$\sum_{i=1}^{n} p_{ai} * R(t_i)$	$\prod_{i=1}^{p} R(t_i)$	$R(t)^k$
Availability (A)	$\prod_{i=1}^{m} A(t_i)$	$\sum_{i=1}^{n} p_{ai} * A(t_i)$	$\prod_{i=1}^{p} A(t_i)$	$A(t)^k$

The overall QoS of each composition plan p is determined by the following fitness function adapted from [1]. The three QoS values of the plan are normalized to [0,1] and w_i is the relative importance the composite service designer gives to each QoS attribute:

$$F(p) = \frac{w1 * ResponseTime(p)}{w2 * Reliability(p) + w3 * Availability(p)} \qquad (2)$$

The objective is to find a composition plan with the minimum fitness value.

4) Determine Quality of Chromosome

In this step, use the best composition plan and its fitness value to represent the chromosome and its fitness. In Fig 5(c), the chromosome is represented by the plan with the fitness of 0.73. In the case that no valid composition plans are found in step *2)*, the fitness of the chromosome would be assigned to a value greater than 1, e.g., 1.5, so that this chromosome with bad fitness will be excluded from the GA process in the next step.

5) Select Parent Chromosomes

Repeat steps *2) – 4)* to determine a composition plan and fitness for every chromosome in the initial population. Once this is done, select good chromosomes to generate the next generation of population. The best chromosome will be retained in the next generation (i.e., elitism), and other good chromosomes will be selected as parent chromosomes to further produce offspring.

6) Generate New Population

To produce offspring from parent chromosomes, mutation and crossover operators are applied to the parent chromosomes with probability defined by the composite service designer. Mutation is done by randomly selecting a position in a parent chromosome and randomly picking a new service instance to replace the one at that position. Fig. 5(d) shows a mutation of the service instance at 4th position. For crossover, a position in two parent chromosomes is chosen as a crossover point for swapping their front and rear parts. Fig. 5(e) shows a crossover at 3rd position. Repeat steps *2) –* 4) to determine a composition plan and fitness for every chromosome within the new population.

7) Determine Stop Condition

The GA process of generating new generations of population, evaluating chromosomes, and selecting parents to produce new chromosomes repeats until a stop condition is satisfied, e.g., when the number of generations to run GA is reached. Then, select the best chromosome of the last generation as the optimal composition.

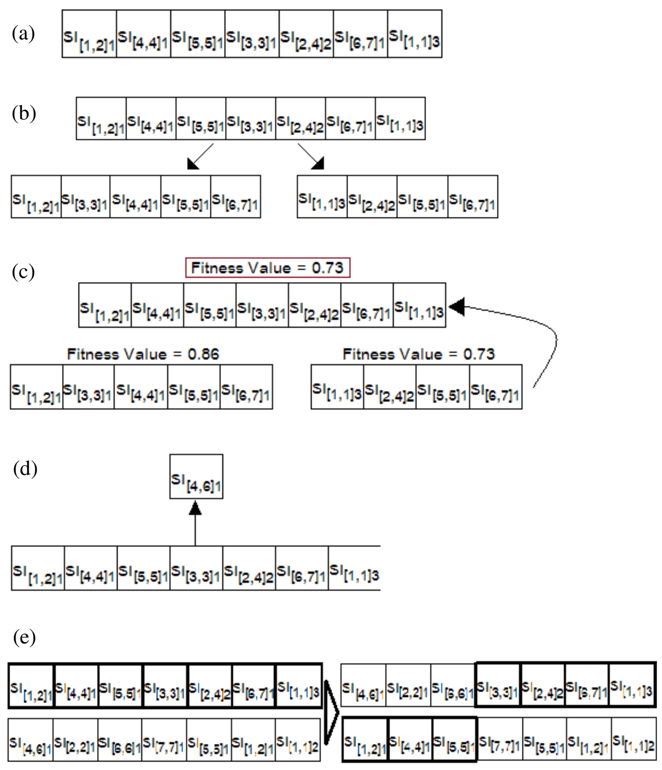

Fig. 5. Generation of chromosome and evaluation of its fitness (a) chromosome in initial population (b) valid composition plans (c) fitness of chromosome (d) mutation (e) crossover.

IV. EXPERIMENTS

In our experiments, we define a sequential flow in Fig. 6 as an initial composition scheme. The flow comprises five abstract services which, given an IP address, will determine the location and nearby place, get weather forecast, and convert temperature unit and translate the forecast into a specified language. We discover some REST Web services from the Internet and also develop some of our own in order to obtain 39 service instances in total. The hosts of these service instances are located in several countries including United States, Germany, United Kingdom, Canada, The Netherlands, France, and Thailand. The granularity of these service instances vary as listed in Table III. We monitor them by invoking them periodically over a period of one month to record response time, reliability, and availability. For reliability, the expected time frame for a correct reply to return is set to 10 seconds.

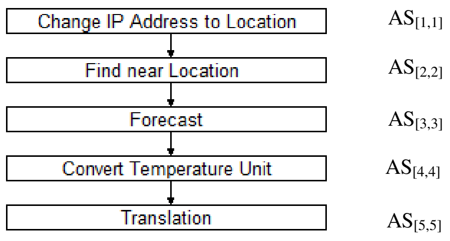

Fig. 6. Initial composition scheme.

TABLE III
39 SERVICE INSTANCES

Granularity	Host
Change IP Address to Location [1,1]	api.hostip.info
Change IP Address to Location [1,1]	en.utrace.de
Change IP Address to Location [1,1]	geoplugin
Change IP Address to Location [1,1]	Gaze
Change IP Address to Location [1,1]	ipinfodb
Change IP Address to Location [1,1]	freegeoip
Find near Location [2,2]	Geoplugin
Find near Location [2,2]	geonames
Find near Location [2,2]	geonamesWIKI
Find near Location [2,2]	gaze-the mySociety
Find near Location [2,2]	api.wunderground
Forecast [3,3]	National Digital Forecast Database
Forecast [3,3]	Yahoo! Weather
Forecast [3,3]	geonames
Forecast [3,3]	wunderground
Forecast [3,3]	worldweatheronline
Convert Temperature Unit [4,4]	DuckDoDuck
Convert Temperature Unit [4,4]	Visual Dataflex
Convert Temperature Unit [4,4]	raisondetre
Convert Temperature Unit [4,4]	cp.eng.chula
Translation [5,5]	Google
Translation [5,5]	Anowa
Translation [5,5]	dict.tu
Forecast+Convert Temperature Unit+Translation [3,5]	Geonames
Change IP Address to Location+Find near Location [1,2]	geoplugin
Change IP Address to Location+Find near Location [1,2]	duckdoduck
Change IP Address to Location+Find near Location [1,2]	2.api.in.th
Change IP Address to Location+Find near Location [1,2]	3.api.in.th
Change IP Address to Location+Find near Location [1,2]	4.api.in.th
Change IP Address to Location+Find near Location [1,2]	5.api.in.th
Change IP Address to Location+Find near Location [1,2]	raisondetre
Change IP Address to Location+Find near Location [1,2]	cp.eng.chula
Change IP Address to Location+Find near Location+Forecast [1,3]	duckdoduck
Change IP Address to Location+Find near Location+Forecast [1,3]	2.api.in.th
Change IP Address to Location+Find near Location+Forecast [1,3]	3.api.in.th
Change IP Address to Location+Find near Location+Forecast [1,3]	4.api.in.th
Change IP Address to Location+Find near Location+Forecast [1,3]	5.api.in.th
Change IP Address to Location+Find near Location+Forecast [1,3]	raisondetre
Change IP Address to Location+Find near Location+Forecast [1,3]	cp.eng.chula

Since we use Weka, the monitored QoS data are put in an arff file to train the neural network. An example of the data sets for training response time, reliability, and availability models of a service instance is shown in Table IV. There are 39 x 3 = 117 models to train in total. Using 10-fold cross validation, we adjust the training parameters of Weka until the root mean square error and mean absolute error of each model are close to 0 and then accept the model.

The parameters for training response time models are as follows: learning rate is 0.27573, momentum is 0.1391, the number of epochs is 1500, and the number of hidden layers is 4 with 5, 4, 3, and 2 hidden nodes.

The parameters for training reliability and availability models are as follows: learning rate is 0.277, momentum is 0.137, the number of epochs is 1500, and the number of hidden layers is 4 with 4, 4, 3, and 3 hidden nodes.

The QoS models can predict the QoS values of each service instance at a certain time of use. An example of the input data sets for the prediction is shown in Table V. Then we use the predicted QoS data for composition using GA. We use equal weight for w_i in (2).

TABLE IV
TRAINING DATA SETS OF A SERVICE INSTANCE

Response Time (arff)	Reliability (arff)	Availability (arff)
@relation service_qos	@relation service_qos	@relation service_qos
@attribute HOUR numeric	@attribute HOUR numeric	@attribute HOUR numeric
@attribute MIN numeric	@attribute MIN numeric	@attribute MIN numeric
@attribute SEC numeric	@attribute SEC numeric	@attribute SEC numeric
@attribute RES numeric	@attribute REL numeric	@attribute AVL numeric
@data	@data	@data
13,57,26,78.0	13,57,26,1	13,57,26,1
15,12,11,63.0	15,12,11,1	15,12,11,1
15,22,57,63.0	15,22,57,1	15,22,57,1
…	…	…

TABLE V
QOS PREDICITON INPUT FOR A SERVICE INSTANCE

Response Time (arff)	Reliability (arff)	Availability (arff)
@relation service_qos	@relation service_qos	@relation service_qos
@attribute HOUR numeric	@attribute HOUR numeric	@attribute HOUR numeric
@attribute MIN numeric	@attribute MIN numeric	@attribute MIN numeric
@attribute SEC numeric	@attribute SEC numeric	@attribute SEC numeric
@attribute RES numeric	@attribute REL numeric	@attribute AVL numeric
@data	@data	@data
21,52,16,?	21,52,16,?	21,52,16,?
17,32,51,?	17,32,51,?	17,32,51,?
15,32,7,?	15,32,7,?	15,32,7,?
…	…	…

V. RESULTS

Service composition is performed for 48 instances of time, i.e., to find the best composite service for use at each half hour of a day. We summarize how many times any service instances appear in the top five compositions at each time instance as in Fig. 7. Table VI lists some of the best composite services with their fitness values, and Fig. 8 shows the number of occurrences each composite service is found the best.

All the information shows that both coarse-grained and fine-grained service instances are part of all 48 solutions; no solution is composed of only fine-grained service instances which match exactly the five abstract services in the initial composition scheme. Therefore it is useful to consider coarse-grained service instances when designing a composite service since they are likely to produce the solution of good quality.

Table VI also shows that at different time of day, QoS of service instances varies, resulting in the change of the best solution. QoS prediction is then beneficial to service composition. Fig. 9 presents the best and average fitness at 48 time instances. This graph shows that at certain period, e.g., between 13:30-21:30, the composition solution remains the best throughout the period and therefore is appropriate for reuse during that time. In addition, the graph suggests that even though the best composite services may vary by time, the composite service designer may want to keep the one being used and not change to a new solution if the difference in their fitness is small, since finding a new solution incurs overheads.

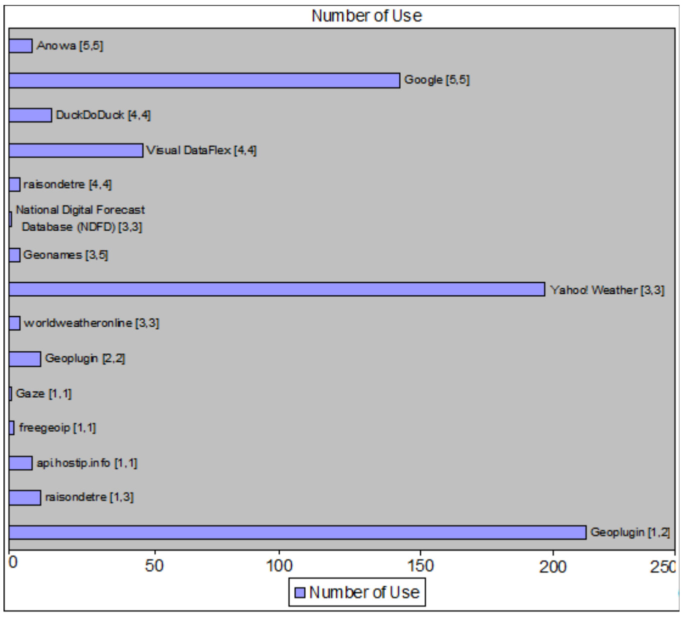

Fig. 7. Inclusion of service instances in top five compositions at each time instance.

TABLE VI
SOME OF 48 COMPOSITE SERVICES FROM EXPERIMENTS

Request Time	Fitness	Best Composite Service
00:14:07	0.1203	Geoplugin [1,2]+Yahoo! Weather [3,3]+ DuckDoDuck [4,4]+Google [5,5]
00:51:17	0.1248	Geoplugin [1,2]+Yahoo! Weather [3,3]+ DuckDoDuck [4,4]+Anowa [5,5]
02:47:32	0.1203	Geoplugin [1,2]+Yahoo! Weather [3,3]+ DuckDoDuck [4,4]+Google [5,5]
…		
03:35:45	0.1717	Geoplugin [1,2]+api.wunderground [3,3]+ DuckDoDuck [4,4]+Google [5,5]
…		
06:48:43	0.1945	raisondetre [1,3]+DuckDoDuck [4,4]+Google [5,5]
…		

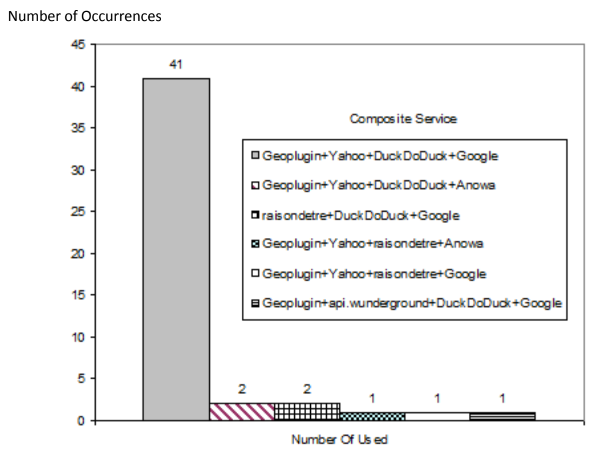

Fig. 8. Best composite services and number of occurrences.

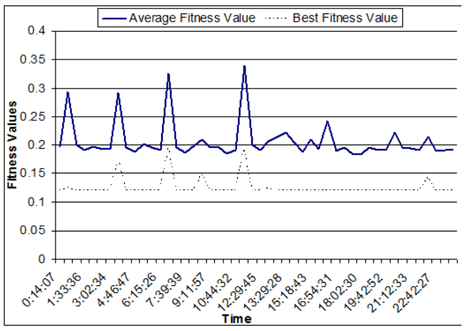

Fig. 9. Best and average fitness at different time.

VI. CONCLUSION

This paper presents a service composition methodology which handles service granularity and QoS variability issues. We use GA to compose services by proposing a new way to encode and generate solution chromosomes and incorporating prediction of service QoS by neural network. Experiments are conducted on 39 Web service implementations, and the results show that coarse granularity and QoS prediction both contribute to finding composite services of good quality.

Our approach still requires the composite service designer to determine granularity of service instances. That is, the designer needs to specify which abstract services in the initial composition scheme a particular service instance can fulfill. For future work, semantic descriptions can be used to better automate this task, e.g., using ontology to describe the scope of tasks of each service instance. A composition framework can also be developed to better support QoS monitoring, training and re-training of QoS models when appropriate, and prediction.

REFERENCES

[1] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *Proc. of the 2005 Conf. on Genetic and Evolutionary Computation (GECCO 2005)*, 2005, pp. 1069-1075.

[2] M. A. Amiri and H. Serajzadeh, "QoS aware web service composition based on genetic algorithm," in *Proc. of 2010 5th Int. Sym. On Telecommunications (IST 2010),* 2010, pp. 502-507.

[3] J. H. Li, S. Q. Chen, Y. J. Li, and G. L. Li, "Application of genetic algorithm to QoS-aware web services composition," in *Proc. of 3rd IEEE Conf. on Industrial Electronics and Applications (ICIEA 2008),* 2008, pp. 516-521.

[4] H. Liu, F. Zhong, B. Ouyang, and J. Wu, "An approach for QoS-aware web service composition based on improved genetic algorithm," in *Proc. of 2010 Int. Conf. on Web Information Systems and Mining (WISM 2010),* 2010, pp. 123-128.

[5] K. Pichanaharee and T. Senivongse, "QoS-based service provision schemes and plan durability in service composition," in *Proc. of 8th IFIP WG 6.1 Int. Conf. on Distributed Applications and Interoperable Systems (DAIS 2008)*, LNCS 5053, 2008, pp. 58-71.

[6] M. Tang and L. Ai, "A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition," in *Proc. of 2010 IEEE Congress on Evolutionary Computation (CEC 2010),* 2010, pp. 1-8.

[7] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "A framework for QoS-aware binding and re-binding of composite web services," *J. Systems and Software,* vol. 81, no. 10, pp. 1754–1769, Oct. 2008.

[8] Z. Gao and G. Wu, "Combining QoS-based service selection with performance prediction," in *Proc. of 2005 IEEE Int. Conf. on e-Business Engineering (ICEBE 2005)*, 2005, pp. 611-614.

[9] M. Anthony and P. L. Bartlett, *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

[10] The University of Waikato. Weka 3: Data mining software in Java [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/