Optimal Approach of Hybrid Genetic Algorithm for Web Service Selection Problem

Kalpesh Lad[1], Trupti Manik[2]

[1]LD Collage of Engineering, Gujarat Technological University, Ahmedabad, Gujarat, India

[2] LD Collage of Engineering, Gujarat Technological University, Ahmedabad, Gujarat, India

Abstract: ***Web Service composition is an important problem to formulate any business process. It's all about creating an abstract web service using some existing web service. A web service may have many implementation having same functionality but with different Quality of Service (QoS) parameters. So, researchers are making effort to select the web service implementation as a part of web service composition so that composite web service gives best overall QoS values or optimal than any other possible composition. Some approaches have come up to address the same problem. The genetic algorithm, which we have considered, addresses the same problem. The genetic algorithm gives constrained driven approach to select the web service for composition. This approach gives much better result than its predecessor. But, it takes longer to execute. If the algorithm takes much time then the system will take much time to respond. This paper introduces an approach to converge the genetic algorithm in a lesser time.***

Keywords: Web Service, Constrained, Web Service Composition, Quality of Service.

1. Introduction

Web service technology is based on open XML standards (i.e. SOAP, WSDL, and UDDI) and has features such as interoperability, decoupling and just-in-time integration, which make it possible to build new value-added web services using existing web services. This is so called Web service composition [1].

A web service may have different implementation available by different providers. These implementations have the same functionality, but may have different Quality of Service (QoS) values. Like they may have different response time, reputation, price and availability, etc,. Thus, a significant research problem in the web service composition is how to select an implementation for each of the web services in the composite web service. Most of the time, a implementation of the web service is not better than other implementation of the same web service for all the QoS criteria. An implementation may be better than the other in terms of some QoS criteria, but may not be as good as other in terms of other QoS criteria. Thus, it's almost impossible or difficult to have some implementation having good QoS values without compromising anything. Hence, the objective of the web service selection problem is to select the implementation of web service in such way that overall QoS is maximal. This web service selection problem is also called QoS-aware web service composition.

In the web service selection there might be some web service implementations that are dependent on each other. When selecting an implementation for one web service, we must select a particular implementation for another web service in the composite web service. For example, when building a travel booking web service, if we select a particular travel insurance web service that only accepts payments made by Master cards, then we must select a payment web service that accepts Master cards. This kind of constraints is called dependency constraint. In addition, in the web service selection there might be some web service implementations that conflict with each other. When selecting an implementation for one web service, we must not select a particular implementation for another web service in the composite web service. For example, when building a travel booking web service, if we select a particular flight booking web service implementation that does not accept deposits made by Master cards, then we must not select an implementation for the payment web service that supports Master cards. This type of constraints is called conflict constraint. In the web service selection, both dependency constraints and conflict constraints must be considered [1].

Although various optimal web service selection problems have been intensively studied and different approaches have been proposed in past [1], [2], [3], [5], [6], [7], [8], [9], [10] the study on the optimal web service selection problem with constraints remains open. From the computational point of view, the web service selection problem is a typical constrained combinatorial optimization problem. Thus, genetic algorithms might be efficient and effective for solving the problem. The approach [1] in paper address the same problem. They have proposed a genetic algorithm to select the optimal constrained web services as a part of web service composition. This algorithm (HGA-Hybrid Genetic Algorithm) [1] is a hybrid one of its two predecessor penalty-based genetic algorithm (PGA) [2] and the repairing-based genetic algorithm (RGA) [3]. The HGA algorithm has a local optimizer to improve the quality of the composition. It was not there in previous two algorithms. As this algorithm checks for all the alternatives of the service, it takes longer to generate result. Because of that, response time of the system is more. The new genetic algorithm checks only those alternatives which have possibility to improve the quality of population. Here we checked few alternatives; hence time for

infeasible alternatives is saved. The new algorithm has implemented and evaluation is done on the basis of some statistics.

The remainder of the paper is organized as follows. First of all, we formulate the research problem in Section II. Then, we review related work in Section III. After that, we present our new hybrid genetic algorithm and statistics results in Section IV. Finally, we conclude this this research in Section VI.

2. Problem Formulation

When building a new composite web service, the following process is usually is followed. First of all, we design a workflow for the composite web service. Fig. 1 is an example of workflow, which consists of 10 web services W1, W2, • • •, W10. Then, we get the information about all available implementations for each of the web services in the workflow. This can be done by using a web service discovery tool or a web service broker. The information includes their URLs, the inter-dependencies and mutual conflicts between the web service implementations, as well as their QoS values of interest. And then tool which uses genetic algorithm used to select the optimal combination of web service implementations accommodating the constraints.

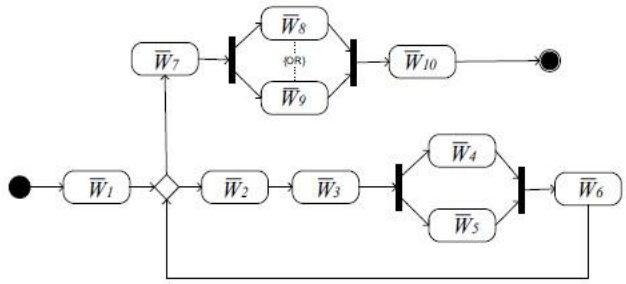

Figure 1An example of workflow for web service composition[1]

In this problem formulation, we follow the terminologies used by the web service community. In the rest of paper, when we say abstract web service, we refer to a web service in a workflow and when say concrete web service; we mean the implementation of an abstract web service. In addition, in the problem formulation, we only consider five most popular QoS attributes. However, the problem formulation and the hybrid genetic algorithm can be easily extended to include any new QoS attributes or to exclude any of the five QoS attributes.

We have taken [1],

- a workflow of a composite web service, which contains a set of abstract web services W = {W1, W2, · · ·, Wn} where n is the total number of abstract web services in the workflow;
- all the available concrete web services for each of the abstract web services $\begin{Bmatrix} (w_{11}, w_{12}, \dots, w_{1\,k_1}), (w_{21}, w_{22}, \dots, w_{2\,k_2}), \dots, \\ (w_{n1}, w_{n2}, \dots, w_{n\,k_n}) \end{Bmatrix}$ where w_{ij} represents the j[th] concrete web service of abstract web service Wi and k_i is the total number of concrete web services of abstract web service Wi;
- QoS values for response time, price, reputation, reliability and availability for each of the concrete web services w_{ij}, $v_{ij}^1, v_{ij}^2, v_{ij}^3, v_{ij}^4$ and v_{ij}^5 respectively, where $1 \le i \le n$ and $1 \le j \le k_i$;
- weights for QoS criteria, c^1, c^2, c^3, c^4 and c^5 for response time, price, reputation, reliability and availability, respectively, where $c^1 + c^2 + c^3 + c^4 + c^5 = 1$;
- a set of conflicts between the concrete web services $C = \{(w_{i_1 j_1}, w_{i_2 j_2})|$ if the j_1^{th} concrete web service is selected for abstract web service x_{i_1} ,then abstract web service w_{i_2} must not select the j_2^{th} concrete web service $\}$, where $1 \le i_1, i_2 \le n$, $1 \le j_1 \le k_{i_1}$ and $1 \le i_2 \le k_{i_2}$;
- a set of dependencies between the concrete web services $D = \{\{(w_{i_1 j_1}, w_{i_2 j_2})|$ if the j_1^{th} concrete web service is selected for abstract web service x_{i_1} ,then abstract web service w_{i_2} must select the j_2^{th} concrete web service$\}$, where $1 \le i_1, i_2 \le n$, $1 \le j_1 \le k_{i_1}$ and $1 \le i_2 \le k_{i_2}$;

a selection plan $X = (x_1, x_2, \dots, x_n)$, where x_i is a concrete web service of abstract web service W_i and $1 \le i \le n$ such that

$$F(X) = \sum_{k=1}^{2} \left(\frac{v_k^{max} - v_k(X)}{v_k^{max} - v_k^{min}} * c^k \right) + \sum_{k=3}^{5} \left(\frac{v_k(X) - v_k^{min}}{v_k^{max} - v_k^{min}} * c^k \right) \quad (1)\ [1]$$

is maximal, where v_k is the aggregated QoS value for criteria k, and v_k^{max} and v_k^{max} represent the possible maximal and minimal aggregated QoS values of criterion k, respectively, where $1 \le k \le 5$ (the aggregated QoS values calculation follows the methods presented in [4]) Subject to all the constraints in *C* and *D* are satisfied.

3. Related Work

The exiting genetic algorithm HGA [1] utilizes local optimizer to improve the quality of individual in the population. The local optimizer goes through all the concrete service selection of the abstract web service. It tries the alternative by replacing existing one. If the modified version of the composition is better than the existing one, then composition plan is replaced with the new scheme. To check whether the new composition plan is better or not, it is necessary to compute fitness value and aggregated QoS value. Now, those alternatives have no possibilities to increase fitness value will be wasting time to compute aggregated QoS value and fitness value.

In this paper new proposed genetic algorithm focuses on problem discussed above. It first check QoS parameters of the alternative Web Service. If it is better than existing selection then only go for fitness value else simply skip those alternatives.

4. Proposed Genetic Algorithm

This section elaborates our hybrid genetic algorithm. This hybrid genetic algorithm uses a local optimizer to improve

the individuals in the population and utilizes a knowledge based crossover operator.

4.1 Genetic Encoding

An individual in the population of our hybrid genetic algorithm represents a web service selection plan and it is encoded in an array of n integers $x_1 x_2 \ldots x_n$, where n is the total number of abstract web services in the workflow of the composite web service. In the genetic encoding scheme, each gene represents an abstract web service in the composite web service and a value of the gene represents concrete web services of the abstract web service.

4.2 Fitness Function

Infeasible individuals may have some schemata that are essential to build the optimal solution. If the infeasible individuals are excluded, the GA may not produce an optimal or near-optimal solution. Thus, the strategy adopted by our GA is to allow infeasible individuals in the population, but gives a penalty to their fitness values. The following two general guidelines are used when defining the fitness function: firstly, it should be guaranteed that an infeasible individual has less fitness value than any feasible individual. Secondly, an infeasible individual that violates more constraints should be more harshly penalized than an infeasible individual that violates less constraint. Equation 1 gives the definition of the fitness function [1].

$$Fitness\ (X) = \begin{cases} 0.5 + 0.5 * F(X), & if\ V(X) = 0 \\ 0.5 * F(X) - \dfrac{V(X)}{V_{max}}, & otherwise \end{cases} \qquad (2)$$

where F(X) is the objective function defined in the problem formulation, V (X) stands for the total number of constraint violations in X, and V_{max} is the maximal number of possible constraint violations. Thus, when V (X) equals to zero, it implies X is a feasible individual; otherwise, X is an infeasible individual.

According to Equation 1, if an individual X is feasible, then its fitness value is given by the expression$0.5 + 0.5 * F(X)$. If an individual X is infeasible, then its fitness value is calculated by the expression$0.5 * F(X) - \frac{V(X)}{V_{max}}$, in which the component $\frac{V(X)}{V_{max}}$ is the penalty given to the infeasible individual X Thus, the more constraints that an infeasible individual violates, the more penalties it receives.

4.3 Genetic Operator

Different from the crossover operator used in the penalty based genetic (PGA) algorithm and the repairing-based genetic algorithm (RGA), the crossover operator used in the hybrid genetic algorithm is a knowledge-based one [1]. The knowledge-based crossover operator takes two parents, $p1$ and $p2$, and produces two children $c1$ and $c2$. When producing $c1$, firstly the crossover operator identifies all the concrete web service selections in $p1$ that do not violate any constraints, and then copies these concrete web service selections to $c1$. The rest concrete web service selections in $c1$ are copied from $p2$. Fig. 2 illustrates the ideas. In the figure, we assumed that

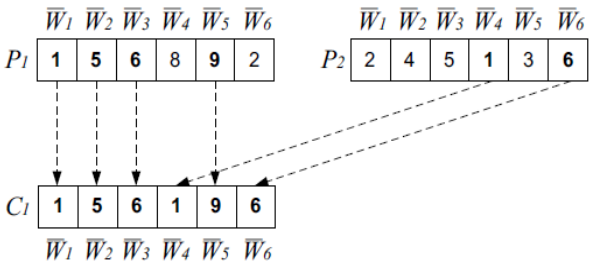

Figure 2 Knowledge-based crossover operator [1]

every highlighted concrete web service selection does not conflict with any other concrete web service selection $p1$. Thus, those selections are copied to $c1$. Now, $c1$ has the concrete web service selection for abstract web services $W1$, $W2$, $W3$ and $W5$. For the rest abstract web services $W4$, $W6$, their concrete web service selections are copied from the corresponding concrete web service selections in $p2$.

Similarly, when producing c2, firstly the crossover operator finds all the concrete web service selections in p2 that do not violate any constraints, and copies them to c2. The rest concrete web service selections in c2 are copied from p1.

The mutation operator [1] is the same as the mutation operator used in the penalty-based genetic algorithm (PGA) and the repairing-based genetic algorithm (RGA). It randomly selects a concrete web service selection for an abstract web service and replaces the concrete web service selection with an alternative concrete web service of the abstract web service.

4.4 Local Optimizer

Given an individual (candidate for the solution) which may or may not be feasible, the local optimizer is to optimize the individuals in the population. The local optimizer is used at the beginning of the genetic algorithm to improve the individuals in the initial population, which are randomly generated, and at the end of each generation to improve the individuals in the population.

The local optimizer improves the fitness value of an individual by increasing its overall QoS value and reducing the number of constraint violations, if any, simultaneously. This is done by systematically checking all the concrete web service selections one by one to see if there exists an alternative concrete web service that gives the individual a better fitness value. If the fitness value is improved, then the current web service selection is replaced with the alternative concrete web service. According to the definition of the fitness function, when the fitness value of an individual increases either the overall QoS value increases, or the number of constraint violations, if any, decreases, or both. Thus, the local optimizer contributes to both maximizing the overall QoS value and minimizing the number of constraint violations of an individual.

4.5 Algorithm Description

As we discussed before, in this paper we have change the local optimizer to reduce the computation time. After checking that current web service selection has no constraints violation, we will try only those alternatives which have higher QoS value by prioritizing with weighted

```
Randomly generate a sequence of abstract web services, W_{x_1}, W_{x_2} ... W_{x_n}
for x = x1 to xn do
        if the concrete web service selection at position x does not violate any constraints then
          while each concrete web service w of Wx having no constraints do
            Calculate the fitness value of the new web service selection plan;
            if the new fitness value is greater than the best fitness value then
              Update the web service selection plan;
            end if
          end while
          Replace the concrete web service selection with the best alternative;
        else
          while each concrete web service w of Wx do
            Calculate the fitness value of the new web service selection plan;
            if the new fitness value is greater than the old fitness value then
              Replace the web service selection with alternative and exit;
            end if
          end while
        end if
end for
output X
```

Listing 1 Local Optimizer [1]

```
Randomly create an initial population of PopSize individuals, Population;
for each individual in Population do
        optimize it using the local optimizer and then calculate its fitness value using the fitness function
end for
find the best individual best in the initial population and store its fitness value into best fitness;
while termination condition is not true do
        for ∀x ∈ Population do
            calculate its fitness value, F(x);
            if the fitness value is greater than best fitness then
                best = x;
                best fitness = F(x);
            end if
        end for
        select individuals from Population using the roulette selection strategy and pair them up;
        for each pair of the selected parents do
            probabilistically use the crossover operator to produce two children, child1 and child2;
            probabilistically use the mutation operator to mutate child1 and child2;
            replace the parents with the children;
        end for
end while
output best.
```

Listing 2 A hybrid genetic algorithm for the web service Selection Problem [1]

QoS values and numbers of constraints that web service have. If web service selection has number of constraints than only those alternatives are tried which have lesser number of constraints. Thus, all infeasible individual are skipped and time to compute fitness value and aggregated QoS value will be saved. This will save considerable amount of time to produce the result. Existing algorithm is shown below. Listing 1 is local optimizer of the old scheme. Listing 2 is genetic algorithm for web service selection problem. Steps for proposed local optimizer is shown Listing 3.

Step 1. Randomly Generate Sequence of abstract Web Services.
$W_{x_1}, W_{x_2}, \ldots W_{x_n}$

Step 2. For all the abstract Web Service check following Step (Step 3).

Step 3. Check if the concrete Web Service selection does have any constraints.
If No, then go to Step 4.
If Yes, then go to Step 5.

Step 4. For each concrete Web Service try their alternatives *having QoS criteria better than current (priorities with weight of QoS and number of constraints)*.
a. Calculate new Fitness value.
b. If it is better than previous then update selection plan else skip.

Step 5. For each concrete web service try other alternatives *having less number of constraints*.
a. Calculate new Fitness value.
b. If it is better than previous then update

Listing 3 Modified Local Optimizer

5. Evaluation

The only difference between new and old scheme is in local optimizer. Some alternatives will be skipped in new scheme. But it brings considerable amount of time saving.

Let's take an example. It takes 20 milliseconds to compute fitness value of the individual. If a problem submitted have 5 individuals, each having 5 abstract web service and each abstract web service have 5 concrete web service and for each of them 5 alternatives are ignored means for them fitness value is not computed, then we can save at least 20*5*5*5*5=12500 milliseconds. That's considerable amount of time when we talking about the response on web.

6. Conclusion

The new scheme will not take all the alternatives into consideration of web services but will only consider the web services which match the selection criteria based on QoS parameters and constraint values. Hence the time to compute other things will be saved. And response time of the new genetic algorithm will be better than old one.

In future, new genetic algorithm will be compared more precisely with HGA, RGA and PGA.

References

[1] L. Ai and M. Tang, ―A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition", IEEE Congress on Evolutionary Computation, Barcelona Spain, 2010.

[2] ——, ―QoS-based web service composition accommodating inter-service dependencies using minimal-conflict hill-climbing repair genetic algorithm," in Proc. IEEE Fourth International Conference on e-Science, Dec. 2008, pp. 119–126.

[3] ——, ―A penalty-based genetic algorithm for QoS-aware web service composition with inter-service dependencies and conflicts," in Proc. International Conference on Computational Intelligence for Modeling, Control and Automation,, Dec. 2008, pp. 738–743.

[4] M. Jaeger, G. Rojec-Goldmann, and G. Muhl, ―QoS aggregation for web service composition using workflow patterns," in Proc. The 8th IEEE International Conference on Enterprise Distributed Object Computing Conference, Sept. 2004, pp. 149–159.

[5] E. Maximilien and M. Singh, ―A framework and ontology for dynamic Web services selection," IEEE Internet Computing, vol. 8, no. 5, pp. 84–93, Sept.-Oct. 2004.

[6] K. Verma, R. Akkiraju, R. Goodwin, P. Doshi, and J. Lee, ―On accommodating inter service dependencies in web process flow composition," in Proc. AAAI Spring Symposium on SWS, 2004, pp. 37–43.

[7] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, ―QoS-aware middleware for web services composition," IEEE Transactions on Software Engineering, vol. 30, no. 5, pp. 311– 327, May 2004.

[8] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, ―An approach for QoS-aware service composition based on genetic algorithms," in Proc. the 2005 conference on Genetic and evolutionary computation., New York, NY, USA: ACM, 2005, pp. 1069–1075.

[9] D. Ardagna and B. Pernici, ―Adaptive service composition in flexible processes," IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 369–384, June 2007.

[10] T. Yu, Y. Zhang, and K.-J. Lin, ―Efficient algorithms for web services selection with end-to-end qos constraints," ACM Trans. on Web, vol. 1, no. 1, p. 6, 2007.

Author Profile

Kalpesh Lad received the B.E. degree in Information Technology from Dharmsinh Desai Institute of Technology in 2011. Now he is studying his Master of Engineering from LD Collage of Engineering, Ahmedabad since 2011. Now he is doing his dissertation in Web Service Composition.

Trupti Manik received the B.E degree in Information Technology from Shri S'ad Vidya Mandal Intitute of Technology in 2009 and M.E. degrees in Computer Science and Engineering from Parul Institute of Technology in 2012. Presently she is working as Asst. Professor at L.D. College of Engineering.