QoS-Based Web Service Composition Accommodating Inter-Service Dependencies Using Minimal-Conflict Hill-Climbing Repair Genetic Algorithm

Lifeng Ai and Maolin Tang
Queensland University of Technology
2 George Street, Brisbane, QLD 4001, Australia
{l.ai,m.tang}@qut.edu.au

Abstract

In the filed of semantic grid, QoS-based Web service composition is an important problem. In semantic and service rich environment like semantic grid, the emergence of context constraints on Web services is very common making the composition consider not only QoS properties of Web services, but also inter service dependencies and conflicts which are formed due to the context constraints imposed on Web services. In this paper, we present a repair genetic algorithm, namely minimal-conflict hill-climbing repair genetic algorithm, to address the Web service composition optimization problem in the presence of domain constraints and inter service dependencies and conflicts. Experimental results demonstrate the scalability and effectiveness of the genetic algorithm.

1. Introduction

Semantic grid [1] is viewed as the future e-Science infrastructure in which there is a high degree of easy-to-use and seamless automation and in which there are flexible collaborations and flexible computations on a global scale. The realization of semantic grid requires a good convergence of a large number of technologies in different research fields, such as agent technologies, knowledge technologies, service orient architecture (SOA) and its enabling technology Web service, e-Science workflow and so on.

This paper is an attempt to enrich present work on the problem of QoS-based Web service composition for e-Science workflow optimization, by incorporating a typical feature of semantic grid, that is, a semantic and service rich environment full of different kinds of context-based constraints imposed on Web services leading to inter service dependencies and conflicts among Web services.

According to [2, 3], QoS-based e-Science workflow optimization could be done in two phases. The first phase is service discovery, which is to discover a set of semantically equivalent Web services (or candidate services) for each abstract Web service (or task) by filtering available services based on service metadata according to the abstract specification. The second phase is to select the optimal service from many semantically equivalent Web service services with respect to their QoS attributes, such as response time, price, reputation, reliability and availability. The second phase is the so called QoS-based selection of services for e-Science workflow optimization.

In this research we assume that the abstract specification of an e-Science workflow is given and the set of semantically equivalent Web services for each abstract Web service has been acquired. Thus, our research focuses on how to select a concrete Web service for each of the abstract Web services in the abstract specification such that that the overall QoS of the composition is optimal satisfying constraints on inter-service dependencies and conflicts.

Although QoS-based selection of Web services for workflow optimization has drawn the attention from the research community [2, 3, 4, 5, 7] and has been intensively investigated in the past few years and several effective approaches have been proposed, the problem of QoS-based selection of services for workflow optimization accommodating the constraints on inter-service dependencies and conflicts is ignored or deemphasized. We contend that in semantically rich and service rich environment, it is significant to consider inter-service dependencies and conflicts that are caused by domain constraints, technological constraints and some other context-based constraints imposed on Web services.

For example, a spatial information workflow requiring two services: a data manipulation service to process GPS data and transform data into standard data format to facilitate the visualization of data on digital map(standard format including "$Shapefile$", "KML"); and a data visualization service to display data on digital map. In this case, GPS data is first manipulated and transformed into "$Shapefile$" data by one service at one step might need to ensure that the

978-0-7695-3535-7/08 $25.00 © 2008 IEEE
DOI 10.1109/eScience.2008.110

workflow chooses a compatible data visualization service that can support "$Shapefile$" data displaying in the subsequent steps. This is an example of technical constraints between Web service. Examples of other kinds of constrains can be found in [6, 10, 11]. From this example, it is easy to understand the selection of a concrete service is not a standalone operation as it may constrain selection of others. In other words, the constraints on dependencies and conflicts between concrete Web services should be taken into account in the selection of Web service for workflow optimization. Thus, it is desirable to look into the problem with the constraints.

This paper proposes a repair genetic algorithm, namely *minimal-conflict hill-climbing repair genetic algorithm* or *MCHC-repair GA*, to solve the problem. Compared with the existing approaches, such as integer programming (IP), GA is slower, but has better scalability and can handle the QoS attributes and the constraints on dependencies and conflicts easier. An important problem in applying GA to solve the Web service composition problem is to handle the infeasible solutions. One of the proven effective technique for handling the infeasible solution problem is repairing [17]. Thus, in this paper we present a repair GA for the QoS-based Web service composition with inter service dependencies and conflicts.

The remainder of the paper is organized as follows. In section 2 reviews related work. Section 3 formulates the Web service composition problem. Section 4 elaborates the repair GA approach in detail. Section 5 reports and discusses our experimental results. Finally, section 6 concludes this research.

2. Related Work

The issue of e-Science workflow optimization incurs lots of interest and researches on this issue focus on the realization of dynamic Web service discovery by incorporating the semantic discovery annotation techniques [3], and on dynamic Web service selection for workflow optimization by applying different optimization techniques [5, 7, 8, 15] in the community of artificial intelligence(AI), as well as on the provision of soft framework [2, 4] to support dynamic discovery and dynamic Web service selection.

Focusing on the issue of dynamic Web service selection, most researches concentrate on the investigation of QoS-based Web service selection for workflow optimization. In [15], the authors provided a comprehensive discussion on QoS-constraint Web service composition, and proposed two new models for this problem: the combinatorial model and the graph model. The combinatorial model defines the optimization problem as a multi-dimension multi-choice 0-1 knapsack problem (MMKP). The graph model transforms the optimization problem into a multi-constraint optimal path (MCOP) problem. Typical optimization techniques applied to tackle this optimization problem embrace integer programming [5], genetic algorithm [7] and some heuristic algorithms [15].

The above researches only consider end-to-end QoS constraints, i.e. the response time, price of the composite Web service should be less than a value given by users, and the reputation reliability,availability of the composite Web service should be greater than a expected value, while the constraints on inter-service dependencies and conflicts are ignored by these researches. In [6, 9], authors demonstrated the necessity of considering more kinds of constraints in workflow optimization tailoring to more flexible and scalable environments. The author further transformed the problem of QoS-constraint Web service composition considering domain constraints and inter-service dependencies into to a constraint satisfaction problem. To tackle this problem, researchers proposed constrained-driven Web service composition frameworks basing on the semantic techniques(i.e. OWL-S) to represent the constraints, and some reason techniques to generate compatible sets of services at runtime. However, after the compatible sets of services have been generated, there is still a requirement of optimization technique applied in runtime Web service composition such that the workflow has optimal QoS while the selected service don't violate the constraints basing on the generated compatible sets of services.

A few work claimed to deal with QoS-based service selection with inter-service conflicts [8, 10]. In their work, they both adopted the IP-based method to handle the service conflicts constraints. However, their approaches are only suitable for small size problems,namely the number of abstract services, candidate services and constraints is small, because of the poor scalability of integer programming approaches. Thus, their approaches are not suitable for the environments having large number of services and various kinds of constraints.

To sum up, most researches in this field only consider end-to-end QoS constraints, while the constraints on inter-service dependencies and conflicts are ignored. Some researches use IP-based method to handle the constraints on dependencies and conflicts, but the method lacks scalability. In this paper, we propose a new GA approach to the constrained Web service composition problem. Different from the existing methods, this GA can handle inter-service constraints on dependencies and conflicts between the concrete Web services and also it has good scalability that is suitable for large-scale and complex problem.

3. Problem Formulation

Given an abstract specification for a workflow, optimal QoS-based Web service composition in the presence of in-

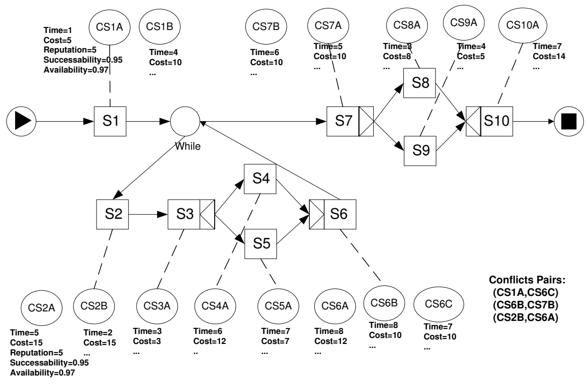


Figure 1. Example of workflow with service bindings and constraints on conflicts

ter service dependencies and conflicts problem is to select a concrete service for each of the abstract Web services in the abstract specification such that the value of an objective function is maximum subject to constraints between concrete Web services.

An example illustrated in Figure 1 is used to demonstrate the problem. In Figure 1, a specification of workflow is defined using YAWL notions [14]. Abstract Web services are represented by rectangles, and concrete Web services are represented by ellipses. All concrete services corresponding to an abstract Web service are functionally equivalent. The choice between them can be dictated by Quality of Service (QoS) attributes. In this paper, we consider the five typical attributes: response time, price, reputation, availability and reliability. Normally, users may desire the composite service with minimal response time and price and maximal reputation, availability and reliability. These quality criteria have different weights of preference. To this aim, the objective function is defined as weighted sum of the quality criteria and the maximal value of objective function reflects the best satisfaction.

The constraints on conflicts and dependencies should also be met. In [6], the authors introduce a mechanism that can check the constraints and generate them which can then be used in Web service selection. In this paper, we assume the constraints set has been generated and are considered as input of the problem. On the bottom right of Figure 1 is an example of constraints set on conflicts. For example, $(CS1A, CS6C)$ denotes that when selecting concrete service $CS1A$ for abstract service S_1, the concrete service $CS6C$ corresponding to abstract service S_6 can't be selected.

According to above requirements, the problem is formulated as follows:

Given

(1) a set of abstract Web services $A = \{A_1, A_2, \cdots, A_n\}$;

(2) a set of implementations, or concrete Web services, for each of the abstract Web service A_i, $C_i = \{c_{i1}, c_{i2}, \cdots, c_{im}\}$;

(3) the QoS values w_{ij}^1, w_{ij}^2, w_{ij}^3, w_{ij}^4, and w_{ij}^5 for response time, price, reputation, reliability, and availability, respectively, for concrete Web service c_{ij};

(4) the weights for the QoS criteria, W_1, W_2, W_3, W_4, and W_5, for response time, price, reputation, reliability, and availability, respectively, where $\sum_{k=1}^{5} W_k = 1$;

(5) a set of dependencies between concrete Web services $D = \{(c_{i_1j_1}, c_{i_2j_2})|$ if abstract Web service i_1 uses the j_1^{th} concrete Web service, then abstract Web service i_2 must use the j_2^{th} concrete Web service$\}$;

(6) a set of conflicts between concrete Web services $C = \{(c_{i_1j_1}, c_{i_2j_2})|$ if abstract Web service i_1 uses the j_1^{th} concrete Web service, then abstract Web service i_2 must not use the j_2^{th} concrete Web service$\}$

where $1 \leq i \leq n, 1 \leq j \leq m$,

Find $X = (x_1, x_2, \cdots, x_n)$, meaning abstract Web service A_i uses concrete Web service c_{ix_i}, such that

$$F_{obj}(X) = \sum_{l=1}^{2}\left(\frac{Q_l^{max} - Q_l(X)}{Q_l^{max} - Q_l^{min}} * W_l\right) + \sum_{l=3}^{5}\left(\frac{Q_l(X) - Q_l^{min}}{Q_l^{max} - Q_l^{min}} * W_l\right) \text{ is maximal}$$

subject to the constraints D and C, where Q_l^{max}, Q_l^{min} represents the max and min value of the l^{th} QoS criterion, $1 \leq l \leq 5$ (the way of calculating Q_l^{min},Q_l^{max},$Q_l(X)$ is the same as the method given by Jaeger, M.C. etc. in [12]).

4. Proposed Approach

GAs [16] are search techniques which could be used for global optimization problem and they perform a search by evolving a population of candidate solutions through the use of nondeterministic operators and by improving incrementally the individuals forming the population by mechanisms inspired from those of genetics (e.g., crossover and mutation). A typical GA consists of the following steps: (1) create an initial population consisting of randomly generated solutions. (2) generate new offspring by applying genetic operators, namely selection, crossover and mutation, one after the other. (3) evaluate the fitness value of each individual in the population. (4) repeat step 2 and 3 until the algorithm converges.

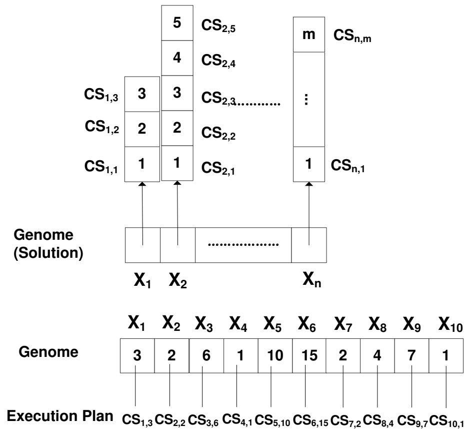
Since GAs are directly applicable only to unconstrained optimization, two different constraints handling mechanisms could be incorporated into GAs to handle the constrains. One is penalty mechanism, that is, to give penalty to *infeasible individuals*(the individuals who violate the constraints) when evaluating the fitness of the reproduced individuals. The GA with penalty mechanism is called *penalty GA*. An alternative technique is to use domain-specific knowledge to fix up those infeasible individuals in the population such that all the individuals in the population are always feasible. The GA with such a repairing mechanism is called *repair GA*.

In this research, we choose repair GA to solve the problem. Specifically, our GA incorporate with an efficient repairing method called minimal-conflict hill-climbing repair, which is a local search method and the notion of MCHC repairing is that the search can be guided by a value-ordering heuristic, the minimal-conflicts heuristic, that attempts to minimize the number of constraint violation after each step. MCHC-repair GA presents three advantages compared to penalty GAs: First, unlike penalty-based GA in which the fitness function would have to be carefully designed to handle the constraints of different kinds, the fitness of our repairing GA is easier to define and easier to calculate as all the individuals in the population are always feasible and therefore it does not need to consider how to calculate the fitness value of the infeasible individuals. The extra work of the repairing GA is restoring the feasibility of infeasible individual via repairing mechanism, while the repairing mechanism in our GA is very simple to be implemented. Second, penalty GAs use minimal a priori knowledge and not exploiting local information might lead to the excessively slow convergence before providing an accurate solution, the incorporation of the local repair heuristics like MCHC could accelerate the converge to an accurate solution. Third, the search of the MCHC-repair GA would be more effective as it does not to explore those search spaces that represent infeasible solutions.

To realize the MCHC-repair GA, we need to define an appropriate problem representation, fitness function, genetic operators and also the repair operator (MCHC) as well. The methods we have employed are elaborated in the following sub-sections.

4.1 Problem representation

The genome is encoded by an integer array with a number of items being equal to the number of abstract service composing the process. Each gene i range from 1 to m, where m is the number of candidate services for the abstract service x_i. The method is illustrated in Fig 2. The bottom part of the Fig 2 is an illustrative example basing on the workflow specification illustrated in Fig 1.



$CS_{2,5}$
$CS_{n,m}$
$CS_{2,4}$
$CS_{1,3}$
$CS_{2,3}$
$CS_{1,2}$
$CS_{2,2}$
$CS_{1,1}$
$CS_{2,1}$
$CS_{n,1}$
Genome (Solution)
X_1 X_2 X_n
X_1 X_2 X_3 X_4 X_5 X_6 X_7 X_8 X_9 X_{10}
Genome
3 2 6 1 10 15 2 4 7 1
Execution Plan $CS_{1,3}$ $CS_{2,2}$ $CS_{3,6}$ $CS_{4,1}$ $CS_{5,10}$ $CS_{6,15}$ $CS_{7,2}$ $CS_{8,4}$ $CS_{9,7}$ $CS_{10,1}$

Figure 2. Encoding and decoding

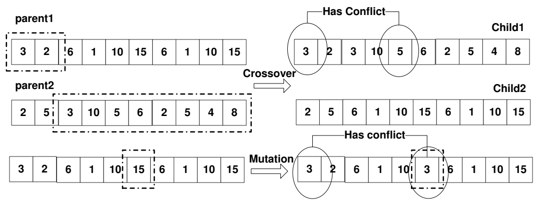


Figure 3. Crossover and mutation

4.2 Elitism

The elitist operator preserves the best solutions found by maintaining a group of them in the next generation. In our GA, the best two individuals are kept alive across generations.

4.3 Genetic operators

The GA adopts one-point crossover operator. Rank-based selection mechanism is used to select individuals to participate in crossover, in which the individuals with higher rank (better fitness value) will be more likely to be selected. The selected individuals are arranged in couple and each pair of couple will generate two offspring using the one-point crossover operator. The mutation operator is conducted by randomly selecting an abstract service (i.e., a position in the genome) and replacing the current concrete Web service with another. Figure 3 illustrates the basic idea behind the two genetic operators. However, the solutions of the new generation will be, in general, infeasible. In this case, the repair operator (MCHC repairing) are applied to enforce feasibility.

4.4 Fitness function

A fitness function is used to measure the quality of the individuals in the population according to the given optimization objective. In our approach, we use the objective function $F_{obj}(X)$ defined in previous section as the the fitness function directly.

4.5 MCHC repairing operator

In order to enforce feasibility, we propose a repair operator named minimal-conflicts hill-climbing (MCHC) to quickly repair the individuals who violate the constraints on inter-service dependencies and conflicts. The repair operator is conducted after the generation of new offspring by genetic operators(crossover and mutation).

MCHC [17] is a heuristic approach and the idea behind the approach is the minimal-conflict heuristic: selecting a variable that is in conflicts and assigning it a value that minimizes the number of conflicts. In MCHC repair, the infeasible individual is iteratively repaired using the heuristic operator until it becomes feasible. The repairing process is like a hill-climbing process, in each step, it will make a small change to the solution, and each time the solution improves a little (with fewer conflicts). Normally both variable and value selection involve an element of randomness: variables are chosen at random from all those that have conflicts and values are chosen at random from the set of minimal-conflict values for the selected variable. Sometime the solution might don't improve anymore after a certain times of repairing. To avoid the endless repairing, we use the maximal repairing times to control the termination of the algorithm when the solution can't fix. The repair approach is showed in Figure 4.

In Figure 4, N_{fix} denotes current repairing times and N_{max} denotes the maximal allowed times for repairing one solution. The repairing process stops until finding a feasible solution or reaches the maximal repairing times N_{max}.

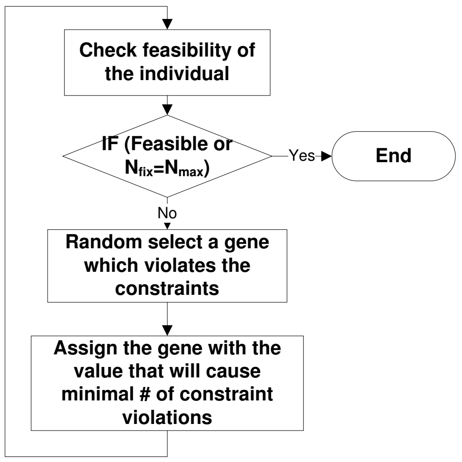

Check feasibility of the individual
IF (Feasible or N_{fix}=N_{max})
Yes
End
No
Random select a gene which violates the constraints
Assign the gene with the value that will cause minimal # of constraint violations

Figure 4. Minimal-conflict hill-climbing algorithm to repair an individual

5 Evaluation

The performance of MCHC-repair Genetic Algorithm is tested via simulation. We tested the effectiveness and scalability of the MCHC-repair by applying it to various composite Web services with different number of abstract Web services, different number of implementations of concrete Web service and different constraints density. The following subsections elaborate the experiments.

5.1 Simulation model

- **Composite Web services and candidate Web services.** We used the workflow given in Figure 1 as the build block to construct Web service composition test problems of different kinds. The workflow contains 10 distinct abstract Web services and involves four types of Web service composition structures: *sequential*, *parallel*, *branch*, and *loop* (the detail explains of YAWL notations and how composition structures are represented by YAWL notations could be found in [14]). In the experiments, we need to construct Web service problems with different number of abstract Web services. To construct a Web service composition problem with 20 abstract Web services, we concatenate two building blocks; To form a Web service composition problem with 30 abstract Web services, we concatenate three building blocks; $\cdots$. In this way, we constructed ten Web service composition problems whose abstract Web service numbers range from 10 to 100 with an incremental of 10. In the generated Web service composition test problems, the execution probability for each path *branch* and the number of iterations of the *loop* must be known so as to calculate the QoS of the composite Web services. In the experiments, we assume each path of the same parallel or branch has uniform execution probability. For example, if a *branch* has 4 pathes, then the execution probability of each path is $1/4 = 0.25$. In the experiments, we set iterations time of a loop structure to 5.

- **QoS.** The QoS values for concrete Web services are generated randomly (In real world, the QoS parameters could be acquired by QoS monitor). The ranges of the values for time, price, reputation, availability and reliability are $[1, 10]$, $[1, 10]$, $[1, 5]$, $[0.9, 1]$, and $[0.9, 1]$, respectively, and the weighting of these criteria are fixed to 0.4, 0.3, 0.1, 0.1, and 0.1, respectively.

- **Constraints.** The constraints between task i and j is generated by randomly picking two candidate Web services, with one for task i and another for task j.

5.2 The GA parameters

The population size of GA is 150 and the maximum number of generations is 200. The crossover probability and mutation probability are set to 0.9 and 0.08 respectively. The maximum iteration time of the hill climbing algorithm for repairing one unfeasible solution N_{max} is set to 50.

5.3 Environments

The repairing GA algorithm has been implemented in Microsoft Visual C# 2005. For performing the simulations, a demo (a data generator) have also been implemented. All the experiments were conducted in a desktop computer with a 2.33 GHz Intel Core 2 Duo CPU and a 1.95 GB RAM.

5.4 Impact of the number of abstract Web services on MCHC-repair GA

This experiment was to investigate the impact of the number of abstract Web services on the computation time of the MCHC-repair GA, and also investigated whether the number of abstract Web services will affect the repairing time. In the experiment, we changed the number of abstract Web services from 10 to 100 with an increment step 10, and we fixed the number of concrete Web services for each abstract Web services to 30 and fixed the number of constraints to 60. Considering the stochastic nature of GA, for each test problem we run 10 times and then calculated the average computation time. To study the impact of the number of abstract Web service on repairing time, we compared the computation time of MCHC-repair GA with the computation time of the GA without repairing. The difference between MCHC-repair GA and the GA without repairing is MCHC-repair GA spending extra time T_{repair} in repairing, namely the repairing time of MCHC-repair GA $T_{repair} = T_{MCHCGA} - T_{GANoRepairing}$. The experimental results are shown in Figure 5. The figure reveals the computation time increases linearly from 1.0 second to 6.7 seconds when the number of abstract Web services increases from 10 to 100, denoting that MCHC-repair GA scales well with the number of abstract service increase. Figure 5 also illustrates that as the number of abstract Web services increases, the computation time of MCHC-repair GA always keep almost the same with the computation time of GA without repairing, implying that repairing cost of MCHC-repair GA is not affect by the number of abstract Web services in workflow.

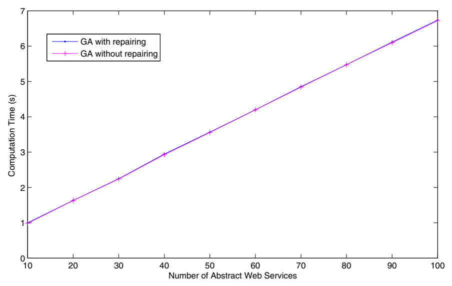


Figure 5. The impact of the number of abstract Web services on the computation time

5.5 Impact of the number of concrete Web services on MCHC-repair GA

This experiment was to investigate how the computation time and the repairing time increases with the number of concrete Web services increases. In this experiment, we varied the number of concrete Web services for each abstract Web service from 10 to 100 with an increment of 10, but fixed the number of abstract Web services and the number of constraints to 10 and 60, respectively. For each test problem we run 10 times and then calculated the average computation time. To study the variation of repairing time as the number of concrete Web services increases, we also compare with the computation time of MCHC-repair GA and GA without repairing. Figure 6 shows the experimental results. It can be seen from the figure that the computation time almost remains about 1.04 seconds when the number of concrete Web services increases from 10 to 100. Compared with GA without repairing, MCHC-repair GA spends 0.01% extra time in repairing. The results demonstrate that the MCHC-repair GA scale very well as the number of concrete Web services increases and also the repairing time don't affect by the number of concrete Web services.

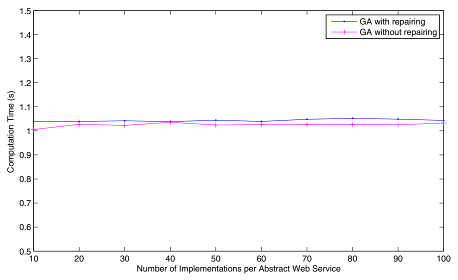


Figure 6. The impact of the number of concrete Web services on the computation time

Table 1. Results of GA with and without repairing operator under different strength of constraints

Test Problems	GA without Repairing		GA with Repairing	
Constraints (#)	Time (sec)	Ave. Constraint Violations (#)	Time (sec)	Ave. Constraint Violations (#)
1×5	1.60991917	0	1.60934702	0
2×5	1.62227303	1	1.63463586	0
3×5	1.63030831	3	1.64416745	0
4×5	1.65289725	5	1.65871566	0
5×5	1.66049209	6	1.69164984	0
6×5	1.68587765	6	1.72433684	0
7×5	1.68168765	7	1.73089489	0
8×5	1.70420541	10	1.75225460	0
9×5	1.71474703	12	1.76567950	0
10×5	1.71946626	12	1.80644116	0

5.6 Impact of the constraints density on MCHC-repair GA

This experiment is to study the impact of the strength of constraints on the computation time and the repairing time. In this experiment, the number of abstract Web services was fixed to 20 and the number of concrete Web service for each of the abstract Web service was also fixed to 20. We increased the number of constrained tasks pairs to strengthen the constraint density. The constraints densities varies from $1 \times 5, 2 \times 5, 3 \times 5$... to 10×5. Here, $1, 2, 3...10$ represent the number of constrained tasks pairs and also reflect the constraints density. In each constrained tasks pair, the number of concrete Web services pairs existing the relationship of constraints were fixed to 5. For each test problem we run 10 times and then calculated the average computation time. Figure 7 shows the experimental results. It can be seen from the figure that the average computation time increases linearly from 1.6 seconds to 1.81 seconds as the number of constraints increases from 1×5 to 10×5. We also compared the computation of MCHC-repair GA with GA without repairing. As depicted in Table 1, GA without repairing can find the solution when the constraints strength is loose. In our experiment, GA without repairing find a feasible solution when the constraint density is 1×5. The MCHC-repair GA can always find a feasible even thought the constraint is strict, i.e. when the constraint density is 10×5. Figure 7 shows that the repairing time increases linearly with the intensity of constraints. This is because more infeasible offspring are generated during the evolution, which leads to the increase in the repairing time. Another reason is as the constraint intensity increases, it becomes more difficult and therefore more time to turn an infeasible individual into a feasible individual.

In the above experiments, the MCHC-repair GA always

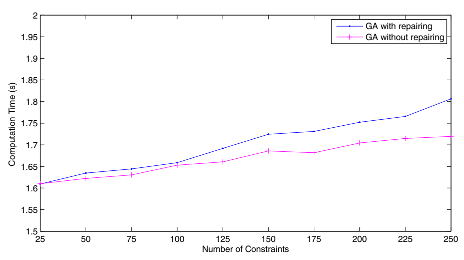


Figure 7. The impact of the number of constraints on the computation time

finds a feasible solution that meets the constraints for all tested composite Web service problems. When the constraint intensity is not strong, the average fitness value is over 0.80 indicating the QoS is very good. As the constraint intensity increases, the fitness value decreases (but the average fitness value is still over 0.70 in our test cases). This is because some of candidate Web services with good quality are no long feasible to be selected as their constraints become restrict. Based on the above three experiments, the following conclusions can be drawn:

- The MCHC-repair GA is scalable. The computation time increases linearly when the number of abstract Web service and the constraint density increases. The computation time changes slightly when the number of constraints or the number of concrete Web services increases.
- The MCHC-repair GA is effective. For all the tested Web service composition problems, the MCHC-repair GA found a quality feasible solution.

6 Conclusion

This paper has studied the problem of QoS-Based Web services composition for e-Science workflow optimization in presence of service dependencies and conflicts, and has proposed a repair GA (MCHC-repair GA) for the problem. The performance of the algorithm has been evaluated by simulation. The proposed MCHC-repair GA has the following merits:

- It is scalable. The experimental results have shown that the computation time of the proposed algorithm increases linearly when the number of the abstract Web services or constraint density increases, and that the computation time is not affected by the number of concrete Web services. Thus, the proposed algorithm can be used for large scale environments with a large number of Web services and constraints.
- It is extensible. Although the proposed MCHC-repair GA currently considers only five non-functional properties, it can be easily extended to include more non-functional properties by incorporating the new non-functional properties in the fitness function and does not need to make any changes to the other part of the proposed GA.
- It is efficient and effective. For all the tested problems, the proposed MCHC-repair GA can find a good feasible solution in seconds.

References

[1] D. De Roure, N. R. Jennings, and N. R. Shadbolt, "Research Agenda for the Semantic Grid: A Future e-Science Infrastructure," National e-Science Centre, Edinburgh, U.K., UKeS-2002-02, Dec. 2001.

[2] D. W. Walker, O. F. Rana, Y. Huang, and L. Huang, Workflow optimisation for e-science applications, In 2005. 27th International Conference on Information Technology Interfaces, Jun. 2005, pp. 86-91.

[3] L. Huang, D. W. Walker, Y. Huang, and O. F. Rana, Dynamic Web Service Selection for Workflow Optimisation, In Proceedings of the UK e-Science All Hands Meeting, Sept. 2005.

[4] E.M. Maximilien, and M.P. Singh, "A framework and ontology for dynamic Web services selection", Internet Computing, IEEE, Sept. 2004, pp. 84-93.

[5] L.Z. Zeng, et al., "QoS-aware Middleware for Web Services Composition", IEEE Transactions on Software Engineering, May 2004, pp. 311-327.

[6] K. Verma, et al., On Accommodating Inter Service Dependencies in Web Process Flow Composition, In AAAI Spring Symposium on Semantic Web Services, 2004, pp. 37-43.

[7] C. Gerardo, et al., An approach for QoS-aware service composition based on genetic algorithms, In Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, 2005, pp. 1069-1075.

[8] A.Q. Gao, et al., QoS-Driven Web Service Composition with Inter Service Conflicts, In Frontiers of WWW Research and Development - APWeb 2006, 2006, pp. 121-132.

[9] R. Aggarwal, et al., Constraint Driven Web Service Composition in METEOR-S, In Proc. IEEE Int. on Services Computing, 2004, pp. 23-30.

[10] C. Nizamuddin, et al, Constraint satisfaction in dynamic Web service composition, In Proceedings. Sixteenth International Workshop on Database and Expert Systems Applications, Aug. 2005, pp. 658- 664.

[11] B. Felix, Introducing Context-Based Constraints, In Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering, Springer-Verlag, Jan. 2002, pp. 130-154.

[12] M.C. Jaeger, G. Rojec-Goldmann, and G. Muehl, QoS Aggregation for Web Service Composition using Workflow Patterns, In Proc. IEEE Int. Conf. on the Enterprise Distributed Object Computing, 2004, pp. 149-159.

[13] O. Yeniay, "Penalty Function Methods for Constrained Optimization With Genetic Algorithms", Mathematical and Computational Applications, 2005, pp. 45-56.

[14] W.M.P. van der Aalst, A.H.M. ter Hofstede, "YAWL: Yet Another Workflow Language". Information Systems, 2005, pp. 245-275.

[15] Y. Tao,Y. Zhang, and K.-J. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints", ACM Trans on the Web, 2007.

[16] J.M. Renders, and S.P. Flasse, "Hybrid methods using genetic algorithms for global optimization", IEEE Transactions on Systems, Man, and Cybernetics, Part B, 1996, pp. 243-258.

[17] M. Steven, et al., Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, Elsevier Science Publishers Ltd., 1992, pp. 161-205.