JAIDM
*Journal of **AI** and **D**ata **M**ining*
*Vol. 1, No.2, 2013, 63-73.*

# QoS-Based web service composition based on genetic algorithm

M. AllamehAmiri, V. Derhami, M. Ghasemzadeh*

*Department of Electrical and Computer Engineering, Yazd University, Yazd, Iran.*

Received 01 October 2012; accepted 09 February 2013
**Corresponding author: m.ghasemzadeh@yazd.ac.ir (M. Ghasemzadeh)*

## Abstract
Quality of service (QoS) is an important issue in the design and management of web service composition. QoS in web services consists of various non-functional factors, such as execution cost, execution time, availability, successful execution rate, and security. In recent years, the number of available web services has proliferated, and then offered the same services increasingly. The same web services are distinguished based on their quality parameters. Also, clients usually demand more value added services rather than those offered by single, isolated web services. Therefore, selecting a composition plan of web services among numerous plans satisfies client requirements and has become a challenging and time-consuming problem. This paper has proposed a new composition plan optimizer with constraints based on genetic algorithm. The proposed method can find the composition plan that satisfies user constraints efficiently. The performance of the method is evaluated in a simulated environment.

**Keywords:** *Web Service, Web Service Composition, Quality of Service, QoS, Genetic Algorithm*.

## 1. Introduction
According to W3C definition "a web service is a software system designed to support interoperable machine-to-machine interaction over a network". It is an XML based, self-described software entity which can be published, located, and used across the internet using a set of standards, such as Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), and Universal Description, Discover and Integration (UDDI) [1]. Since web services can enable computer-computer communication in a heterogeneous environment, hence they are very suitable for an environment such as the internet. People can use the standardized web service model for rapid design, implement and extended applications. Many enterprises and corporations provide different web services to be more responsive and cost-effective. Google's SOAP Search API for information inquiry [2] and Amazon web services for doing enormous e-commerce activities [3] are good examples of such systems. A number of standards and protocols have been designed to use and publish web services over the internet. Some of the most commonly used standards are UDDI, SOAP and WSDL. Universal Description Discovery and Integration (UDDI) is an XML-based registry that provides a standard set of specifications for service description and discovery. It defines the information model, the service providers API for registering and publishing services and the API for service requesters to inquire for services. Web service provider registers their web services into UDDI registries. Simple Object Access Protocol (SOAP) is an XML based protocol specification for exchanging information between peers in the

decentralized, distributed environment. SOAP provides a simple and lightweight mechanism to communicate with web services. SOAP can form the foundation layer of a web services protocol stack. Web Service Description Language (WSDL) is used to describe the interfaces of all web services regardless of the underlying technology. The WSDL is defined: Services as collections of network endpoints, or ports. When service provider wants to register a web service to UDDI server (web service directory), it describes web service by WSDL and puts it in UDDI registry. As service requester looks for a web service in UDDI server, s/he receives the WSDL file of web services Figure 1 shows the IBM standard architecture of web services. This architecture provides a three level procedure to find an appropriate web service. First, service provider describes its web services in WSDL Format and puts them in a web service directory (registering web service). Then, service requester searches into web service directory to find a suitable web service. Finally, after selecting the web service, service requester can interact with the web service using SOAP protocol. There are some sophisticated applications that cannot be performed using a single, isolated web service. Consequently we need to use a composition of web services to perform complex tasks. An Example of synthesizing web services is a travel planning web service. When the client uses web service based system to plan a trip, the following steps will be taken into consideration in the service process.

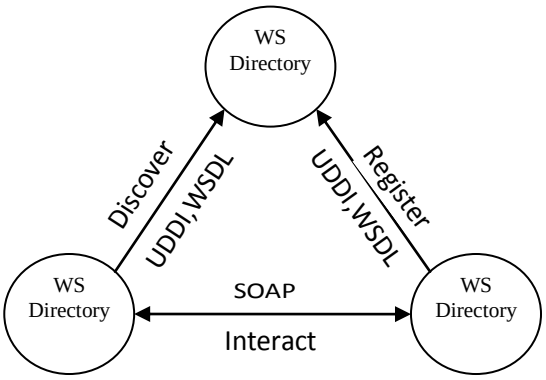



**Figure 1. Standard architecture of web services.**

At first, the client contacts a travel agency web service to reserve a hotel room and an airplane seat. Then the client selects the best reservation plan among the plans suggested to him/her by considering factors like schedule, financial condition, weather conditions and some other factors. In addition, the client may request services, such as a car rental agency or insurance. After all web services are selected, the client pays the reservation fee to the travel agency. Figure 2 provides an example of travel agency candidate web services and Figure 3 represents all composition plans of the candidate web services.

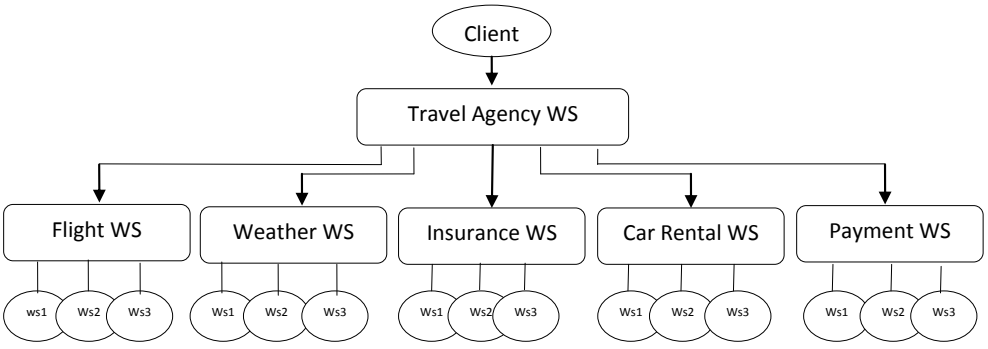



**Figure 2. An example of travel Agency web service and candidate web services for each task.**

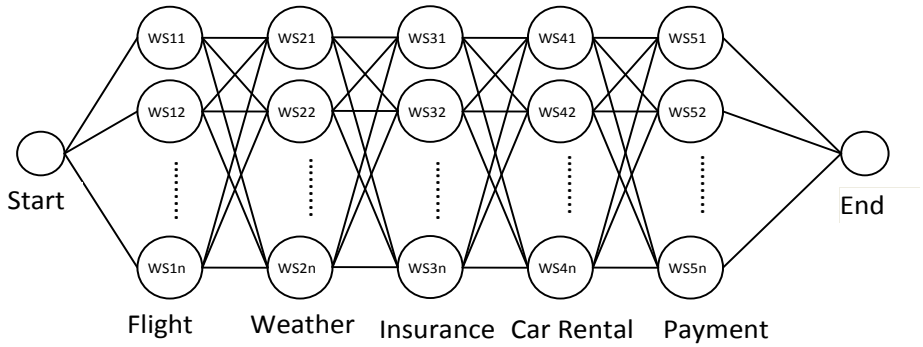



**Figure 3. All composition plans for travel agency example.**

Web service composition creates new functionalities by aggregating different services based on a specific workflow [4]. When there are more than one candidate web services for a task or process, there will be various combinations of web services having the same functionality with different qualities. For instance, if there are *m* tasks and *n* candidate web services, the number of all possible plans is $n^m$. In general, finding a composition plan that fulfils a client's QoS requirement is a time-consuming optimization problem. Combining web services of high QoS values in a reasonable computation time has been recognized as an important problem of web service composition [6]. We need to find a composition plan satisfying client's constraints without checking all combinations. This will be impractical even if there are a few services and tasks in the workflow. Typical QoS factors associated with a web service are executive cost and time, availability, successful execution rate, reputation, and usage frequency [5]. Also, there are other properties other than the above-mentioned factors, such as reliability, security and so on. To obtain a composition plan, we should first create a QoS model to describe the QoS aspects of web services. To create an appropriate model, service requester and service provider should agree on same definitions to the extent possible. After creating a QoS model, the second step is QoS based on web service discovery and selection. Unfortunately, WSDL only addresses functional aspects of a web service and does not contain any useful description for non-functional requirements [14]. Using the QoS model, service requester can filter inappropriate web services. A number of studies on web service selection have been carried out [7, 8]. One of the most well known techniques is ‘‘matchmaking" that is employed in situations where services with semantic descriptions for their functional attributes are available on the Internet search system [7]. It should be noted that the process of filtering web services consists of functional matchmaking and non-functional matchmaking. In functional matchmaking, web services that have different functionalities from the client are filtered out and on the other hand, in non-functional matchmaking, web services that don't have the appropriate quality are eliminated. At this stage, the candidate web services for each task are selected. In [8], a new QoS-based service registration and discovery model to explore the feasibility of QoS involving into UDDI registry information is suggested. In this model, service providers have to send QoS claims to service QoS certifiers, responding to the third party or forum web services, for certification. The service customer is responsible for verifying QoS claims. Finally, if QoS claims pass QoS certifier verification, the QoS information will be registered in the UDDI registry associated with function description. In the last stage, we should obtain the optimized web service composition plan from all available plans. As mentioned above, trying all combinations of web services is time consuming. A problem of web service composition is usually an NP-hard [9]. Several solutions have been suggested so far to solve this problem that one of them such as [10] is based on Leaner programming and some are based on AI (e.g. [6]). Genetic algorithm is effective approach to solve some kind of hard problem [22, 23, 24, 25, 26]. Our approach uses the genetic algorithm [20, 21] to solve this problem. To escape from local optimums, we present some modifications in crossover, mutation and selection approach. The remaining sections of this paper are organized in order. Next section presents related work. The third section describes the QoS model of web service composition. The fourth section proposes our GA based on composition algorithm. The computational results of the algorithm are given in the fifth section and the sixth section includes conclusion and future work.

## 2. Related work

Web service discovery and QoS based on the web service composition offer interesting applications of constraint satisfaction methods. In [10] a multiple criteria decision making with weighted sum model (to select a service) and integer programming (IP) approaches with branch and bound (to select an optimal solution) have been proposed. In [6] constraint satisfaction based on solution which combine simulated annealing [17, 18, 19] approach with Tabu search [16] has been proposed. The Tabu search is used for generating neighbour plans and simulated annealing heuristic is applied for accepting or rejecting the neighbour plan. In [11] a QoS-based web service composition algorithm that combines local strategy and global strategy has the following features. Initially, the services that have low QoS value are eliminated by local strategy and then the problem has reduced to a multi-dimension multi-choice 0-1 knapsack problem solved by the heuristic method. In [12] a model that expands traditional UDDI to describe the QoS attributes of

web services is presented. Also, a service proxy is added to this model by which all service compositions requested by service requester are found, bound and invoked. In [13] an automated web service composition is performed by hierarchical task network and SHOP2 HTN system is developed. This system takes OWL-S service model as input (client requirement) and executes the plan as a system result. High probability of Getting stuck in local optimum is the main problem of these methods. This is because it is unable to work more than one composition plan at the same time. At the same time, the probability in methods such as genetic algorithms and fish swarm algorithms working on several composition plans are less than above-mentioned method.

## 3. QoS based web service model

### 3.1. QoS properties description

The most important QoS properties used in this paper are response time, execution cost, availability, reputation and successful execution rate. The response time can be defined in several ways. For example, it can be defined as the time between sending request and receiving respond. This period involves receiving request massage time, queuing time, execution time and receiving response time by requester. Measuring these time sections is very difficult because they depend on network conditions. Alternatively, it can be measured as the time between receiving request by service provider and sending response to service requester. This time includes queuing time and execution time only affected by the web service workload. This value must be continuously updated for each web service because the work load of web service may change during the work time. Execution cost is a fee received by service provider from service requester for each execution. This fee is determined by service provider and may change according to web service provider's financial policy. Availability is the degree that a web service is accessible and ready for immediate use. This value can be defined as [uptime/ (uptime + downtime)]. Downtime includes the time that web service is inaccessible and time taken to repair it. This value should be updated by service provider. Reputation is the average reputation score of a web service evaluated by the clients. The individual reputation scores are likely to be subjective, but the average score becomes trustable as the total number of the usages increases [6]. The successful Execution Rate is the percentage of requests that a web service perform successfully when web service is available. It is computed by dividing the number of successful performed requests by the total number of requests. The QoS properties used in this paper is summarized in Table 1.

**Table 1. Description of QoS peoperties used in this paper**

| QoS property | Description |
|---|---|
| Response Time | Time between receiving request and sending response |
| Execution cost | Execution cost per request |
| Availability | $\frac{\text{UpTime}}{\text{UpTime}+\text{DownTime}}$ |
| Reputation | $\frac{\sum \text{Rep}_i}{\text{TotalNumber Of Usage}}$ |
| Successful Execution Rate | $\frac{\text{Numer of Successful Request}}{\text{TotalNumber of Request}}$ |

**Notations**

Descriptions of notations used in this paper are as follow:

$m$: number of tasks.

$n$: number of candidate web services for each task.

$p_i$: i-th atomic process of a composition schema ($1 \le i \le m$).

$ws_{ij}$: j-th candidate web service for the ith atomic process, ($1 \le i \le m$ , $1 \le j \le n$).

$d$: index of QoS property .

$w_d$: weight of the $d$-th QoS constraint defined by a client.

$Con_d$: permissible value of the $d$-th QoS property (constraints).

$Agg_d$: aggregated value of the $d$-th QoS property of a composition plan.

$b_{ij}$: binary decision variable (0 or 1). If $b_{ij}$=1 then $j$-th candidate web service is selected for $i$-th process.

### 3.2. QoS-based evaluation of web services

Since each QoS property may be measured in various metrics, they should be normalized for appropriate evaluation. The QoS properties are divided into two categories: First, negative values, such as response time and execution cost, and second, positive values, such as availability and reputation. The higher value in negative properties indicates the lower quality and the higher one in positive properties represent higher quality and vice versa. The following equations are used to normalize positive and negative properties, respectively:

$$q_{nrm} = \begin{cases} \dfrac{q - q_{\min}}{q_{\max} - q_{\min}} & q_{\max} - q_{\min} \neq 0 \\ 1 & q_{\max} - q_{\min} = 0 \end{cases} \quad (1)$$

$$q_{nrm} = \begin{cases} \dfrac{q_{\max} - q}{q_{\max} - q_{\min}} & q_{\max} - q_{\min} \neq 0 \\ 1 & q_{\max} - q_{\min} = 0 \end{cases} \quad (2)$$

After normalization, the local value of each web service that is candidate for a task will be computed from the following formula. Where Q is the number of QoS properties.

$$Local\,Value\,of\ ws_{ij} = \sum_{i=0}^{q} w_i q_i \quad (3)$$

### 3.3. Aggregation value of QoS property

Generally, composition plans are constituted from serial, cycle, XOR-parallel, and AND-parallel execution patterns. According to the definition of QoS properties in section 3.1, the aggregative value of web service composition is calculated regarding to its workflow pattern. The description and aggregation values of workflow patterns are discussed below.

Serial pattern is an execution pattern in which services are executed one after another and there is no overlap between execution periods of web services. Figure 4 illustrates this pattern and Table 2 represents the aggregation value of this pattern. According to Table 3 to calculate aggregation value of response time and execution cost, each web service value should be added to each other. Besides, in order to calculate aggregation value of availability and successful execution rate, web services values should be multiplied by each other because web services are independent from each other. The aggregative value of reputation is obtained by taking average of reputation values of web services.

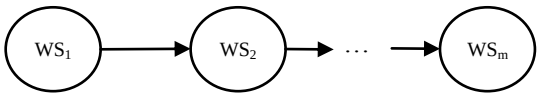



**Figure 4. Serial Pattern.**

Cycle pattern is a kind of sequential pattern in which the web service executes for limited cycles. According to Table 3, the aggregation values of this pattern are similar to sequential pattern. Figure 5 describes this pattern.

**Table 2. Aggregative QoS value for serial pattern**

| | |
|---|---|
| Response Time | $\sum_{i=0}^{m} WS.RT$ |
| Execution Cost | $\sum_{i=0}^{m} WS.EC$ |
| Availability | $\prod_{i=0}^{m} WS.Ava$ |
| Successful Execution rate | $\prod_{i=0}^{m} WS.Suc$ |
| Reputation | $\dfrac{\sum_{i=0}^{m} WS.\mathrm{Re}\,p}{m}$ |

**Figure 5. A cycle pattern.**

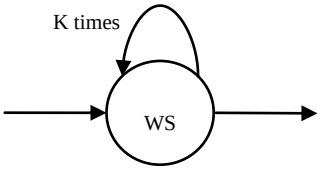



**Table 3. Aggrigative QoS value for a cycle pattern**

| | |
|---|---|
| Response Time | $m*(WS.RT)$ |
| Execution Cost | $m*(WS.RT)$ |
| Availability | $WS.Ava$ |
| Successful Execution rate | $WS.Suc$ |
| Reputation | $WS.\mathrm{Re}\,p$ |

XOR-parallel pattern is an execution pattern in which after the completion of the prior web service, one of the following web services just executes. In this pattern execution of each component is non-deterministic; therefore, to calculate the aggregation QoS effect of this pattern, the worst case should be calculated. We can obtain aggregative QoS values of this pattern as described in Table 4. Figure 6 depicts this pattern.

AND-parallel pattern is an execution pattern in which after the completion of the prior web service, the entire subsequent web services are executed simultaneously. The aggregative QoS values of this pattern are described in Table 5. Notice that to obtain aggregative response time, we use the Max function, because all subsequent

components are executed simultaneously. Figure 7 describes this pattern.

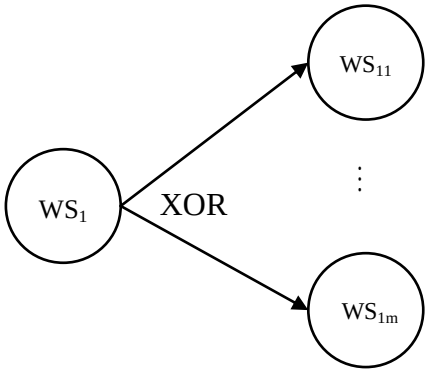



**Figure 6. XOR-parallel pattern.**

**Table 4. Aggregative QoS value for XOR-parallel pattern**

| | |
|---|---|
| Response Time | $Max(WS.RT)$ |
| Execution Cost | $Max(WS.EC)$ |
| Availability | $Min(WS.Ava)$ |
| Successful Execution rate | $Min(WS.Suc)$ |
| Reputation | $Min(WS.\mathrm{Re}\, p)$ |

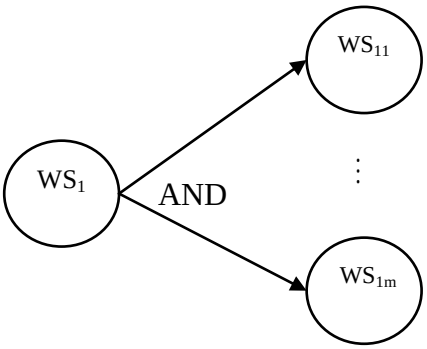



**Figure. 7. AND-parallel pattern.**

**Table 5. Aggrigative QoS value for XOR-parallel pattern**

| | |
|---|---|
| Response Time | $Max(WS.RT)$ |
| Execution Cost | $\sum_{i=0}^{m} WS.EC$ |
| Availability | $\prod_{i=0}^{m} WS.Ava$ |
| Successful Execution rate | $\prod_{i=0}^{m} WS.Suc$ |
| Reputation | $\frac{\sum_{i=0}^{m} WS.\mathrm{Re}\, p}{m}$ |

## 4. The GA based algorithm

In this section, we present our approach to find an optimal web service composition plan Since the number of all composition plans of this problem is very large ($n^m$), some ideas to improve GA are presented so that it quickly converges the appropriate composition plan. We introduce some idea for initialization, crossover and mutation of chromosomes. Also the method to escape from local optimum is represented. If the algorithm cannot find the optimal plan in a specific time, without losing best plans of previous step, the algorithm will escape from local optimum.

### Constraints

There are two constraints. The first constraint is that only one web service among candidate web services should be chosen for a task. In other words, the equation (4) has to be satisfied. The second constraint is that the service composition must satisfy user constraints. For negative QoS properties, such as execution cost and time the aggregation values must be smaller than user constraints. For positive QoS properties, aggregation values must be greater than user constraints. Equation (5) describes this constraint.

$$\sum_{i=0}^{n} b_{ij} = 1 \quad 1 \le i \le m \tag{4}$$

$$\begin{cases} Agg_d \le Con_d & \text{For negetive constraints} \\ Agg_d \ge Con_d & \text{For posotive constraints} \end{cases} \tag{5}$$

### Algorithm construction

To obtain relation between local optimum and global optimum chromosomes, several experiments have been carried out. We design a small example of composition plan with m=10 and n=6 in which the number of plans is $10^6$. Initially, all web services should be sorted according to their local values. Then, the best composition plan is found by using enumeration methods. In this method, all composition plans are obtained and evaluated. Table 6 shows the percentage of web services that are selected for global optimum plan. The closer this amount is to 9, the higher its local value will be. About 36.6% of web services in the composition plans have best local values too. It can be inferred that about 70% of web services that are selected for the best composition plan belong to 30% of the best web services that have high local values. Figureure. 8 depicts the values of fitness function of all plans.

In this diagram, each two adjacent points that demonstrate two adjacent plans, differ only in one web service. As depicted in this figure, for this small example, several local optimums exist so the algorithm should be changed so that it can escape from them; to escape from these local optimums, the various forms of randomness is required.

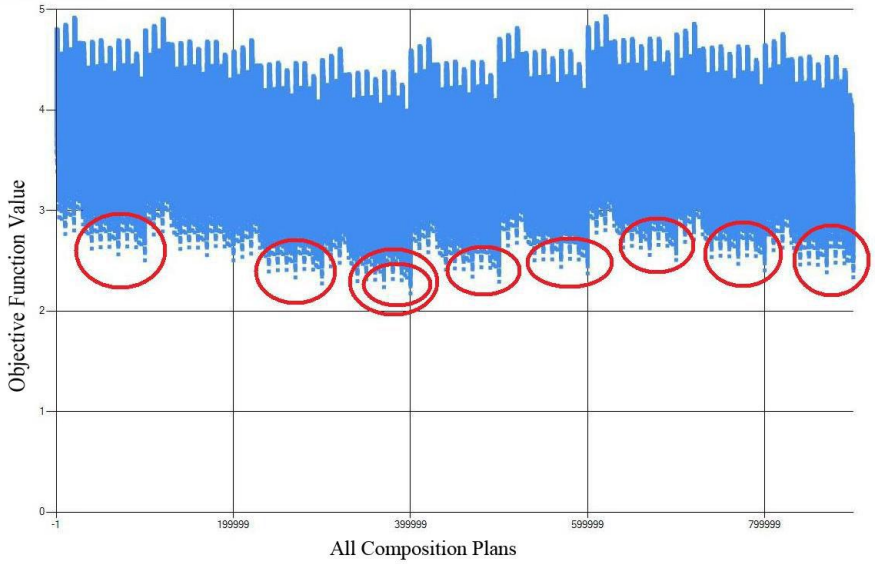



**Figure. 8. Value of all plans in example with 6 tasks and 10 web services.**

**Table 6. Relation between Local and global optimum**

| WS number | Percentage of repeat in best plan |
|---|---|
| 1 | 2.2% |
| 2 | 1.6% |
| 3 | 4.4% |
| 4 | 5.5% |
| 5 | 6.2% |
| 6 | 9.0% |
| 7 | 12.2% |
| 8 | 22.3% |
| 9 | 36.6% |

Construction of chromosome is described in Figure 9. Each chromosome consists of m genes and each gene has a value between 1 to n.

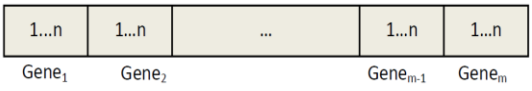



**Figure 9. Chromosome structure.**

The main body of the algorithm is summarized in Table 7 and specifications of its functions are described in Tables 8, 9, 10, 11 and 12. At first, we should calculate local value of each web service. This is done prior to execution of composition plan optimization. Local value is a criterion of goodness among the candidate web services of a specific task. To obtain the local value, the QoS properties are normalized according to equations (1) and (2), and then the local value is calculated using equation (3). To obtain an optimized composition plan at first, web services candidate for a task are sorted according to their local values. In the next step, chromosomes are generated. 20% of all chromosomes are selected from 20% of best web services that have high local value and the remaining 80% is selected randomly. For each chromosome fitness function is calculated by using equation (9). It is derived from objective functions of [6, 10, 15]. $D_1$ is used for negative values and $D_2$ is used for positive values. If the fitness function value is equal or smaller than 0, it will mean that the appropriate composition plan satisfying user constraints is found.

$$D_1 = \sum w_d .(\frac{Agg_d}{Con_d} - 1) \quad if \ Agg_d \geq Con_d$$

(For negetive values)

$$D_2 = \sum w_d .(\frac{Agg_d}{Con_d} - 1) \quad if \ Agg_d \leq Con_d$$

(For posotive values)

$$Fitness\ Function = D_1 - D_2 \qquad (9)$$

Crossover is a function to combine two or more parent chromosomes and obtain one or more child chromosomes. We define two kinds of crossover. In crossover type 1 shown in Figure 10, genes of a child are inherited from parents alternatively. In crossover type 2 shown in Figure 11, a certain percentage of genes are inherited from one parent and the other genes are inherited from the other

parent. For crossover operation, 20% of best chromosomes having high fitness are combined with each other by crossover type 1 and for the remaining 80%, each chromosome is combined with a randomly selected chromosome belonging to 20% of chromosomes with high fitness function by crossover type 2.

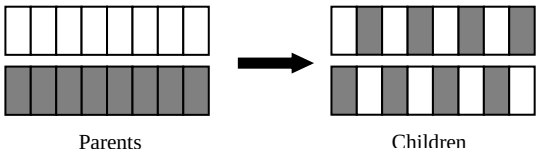



**Figure 10. Crossover type 1.**

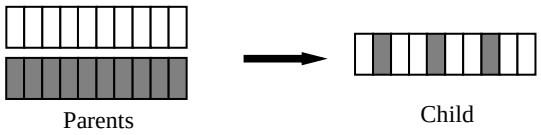



**Figure 11. Crossover type 2.**

To select chromosomes for the next step (selection function), constant number of best chromosomes from the previous step are selected and replaced with worst chromosomes in current step. This results in preservation of best chromosomes but accelerates the convergence of the algorithm to the local optimum. To escape from local optimum we design mutation and partial initialization chromosomes functions. In mutation, some genes of some chromosomes that are selected randomly will change with probability of $P_m$.

To fix the selection function accelerating the convergence of the algorithm to the local optimum, the Partial initialization chromosomes function is presented. In this function, a constant number of best chromosomes are kept and other chromosomes are generated randomly again.

## 5. Experiments

We have accomplished several experiments to evaluate our algorithm. The programming language used to do the evaluation is Java and the algorithm is executed on desktop PC with Pentium 2.2 GHz dual core CPU and 3 GB of RAM. We compare the execution time of our algorithm with enumeration method. The first experiment was performed with 30, 50 and 100 tasks. For each task we have 30, 40 and 50 web services. As shown in Figureure. 12, the maximum execution time of the algorithm is 377 milliseconds. In a separate run, another experiment is performed with 100 web services in which the number of tasks is 20, 40 and 50. In this experiment the maximum time is equal to 240 milliseconds. The results are shown in Figureure. 13. Figure 14 shows the result of enumeration method. In enumeration method, the plans are generated until the suitable plan is founded. From this diagram, it can be inferred that the time of execution increases exponentially when number of tasks increase linearly. Furthermore, we compare our work with the work represented in [6]. They provide a solution for composition plan optimization using a combination of Tabu search and simulated annealing approach. The result of this comparison is shown in Figure 15.

**Table 7. Main body of algorithm**

```
Function Composition_Plan_Optimizer
    Sort all web services according to their local value;
    Initialize chromosomes ();
    Sort all chromosomes according to their local score;
    Counter=0;
    While not find appropriate plan do
        Crossover ();
        Sorts all web services according to their local score;
        Selection ();
        Sorts all web services according to their local score;
        Mutation ();
        Sorts all web services according to their local score;
        Counter=counter+1;
        If (counter % T=0) Do
            Partial_initialization_Chromosome();
        End if
    End While
End Function
```

**Table 8. Initialization chromosomes function**

```
Function Initialization Chromosome
    For 20% of population do
        Select chromosome genes are selected randomly from 20% of best web services
    End For
    For 80% of population do
        Select chromosome genes are selected randomly.
    End For
End Function
```

**Table 9. Crossover function**

```
Function Crossover
    20% of best chromosomes are combined with each other by crossover type 1.
    80% of remaining chromosomes are combined with 20% of best chromosomes with crossover type 2.
End Function
```

**Table 10. Selection function**

| Function Selection<br>Replace N number of best chromosomes from previous step with N worst chromosomes of current step<br>End Function |
|---|

**Table 11. Mutation function**

| Function Mutation<br>With probability of $P_m$, some genes of some chromosomes are changed randomly<br>End Function |
|---|

**Table 12. partial_initialization_chromosomes function**

| Function Partial_initialization_chromosomes<br>Keep N number of the best chromosomes and other chromosomes are initialized again.<br>End Function |
|---|

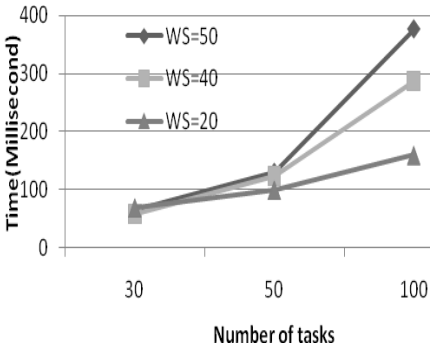



**Figure 12. Performance of GA based algorithm**

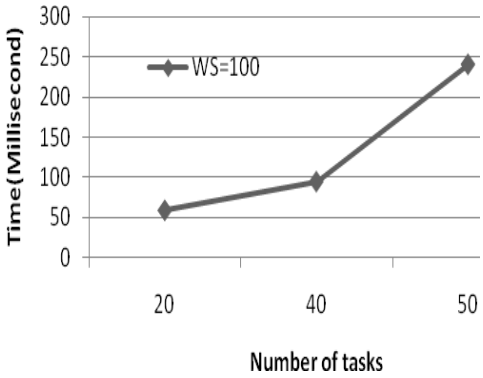
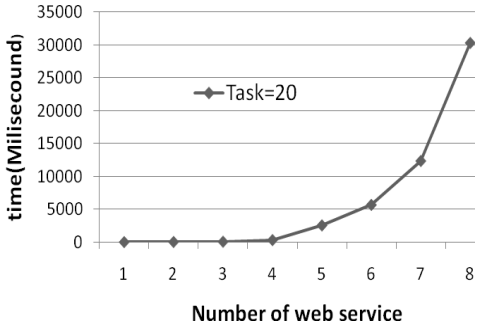
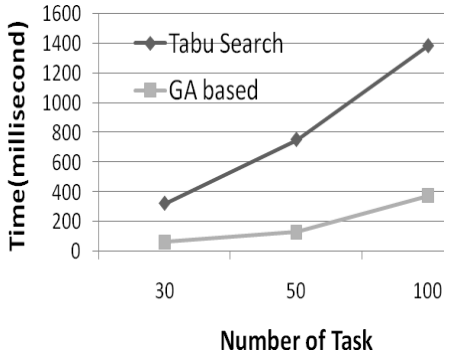



**Figure 13. Performance of GA based algorithm with 100 tasks.**




**Figureure. 14. Performance of enumeration method**




**Figureure. 15. Compare with Tabu search approach**

Furthermore, one of the important factors having the significant impact on the execution time is the population size. Figureure. 16 shows the impact of the population size on the execution time. In this experiment, the number of web services is 100 and the numbers of tasks are 20, 40 and 50 respectively. As it can be inferred from the diagram, the best population size is in range of 300 to 500.

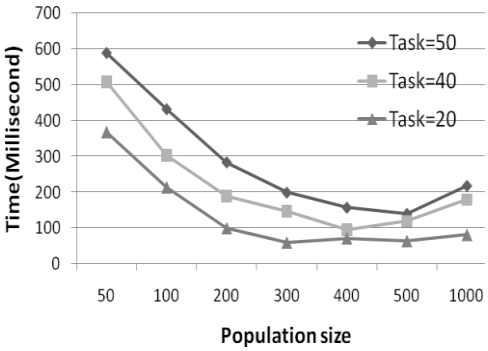



**Figure 16. Impact of population number on the performance of algorithm (web service=100)**.

## 6. Conclusion

In this paper, we showed how can find the suitable web service composition using genetic algorithms. Some new ideas for generating chromosomes, selection and crossover functions were proposed. The experimental results

demonstrated the advantages of the proposed ideas are to overcome local optimums. Experimental results show that since GA is a K beam search, it can find suitable composition plan much faster than other random search approaches. Therefore, it can be concluded that applying genetic algorithms in such problems has a great effect on improving computation time. As a future work, we suggest examining effects of using different formulas for fitness function.

**Acknowledgements**
This work was partially supported by Iranian Telecommunication Research Center (ITRC).

**References**

[1] CURBERA, F., M. DUFTLER, R. KHALAF, W. NAGY, N. MUKHI, S. WEERAWARANA, Unraveling the Web Services web, An Introduction to SOAP, WSDL and UDDI, IEEE Internet Computing, Vol. 6, (2002), pp. 86–93.

[2] KOSHMAN, S., Visualization-based Information Retrieval on the Web, Library and Information Science Research Vol. 28, (2006), pp. 192–207.

[3] CHEN, L.S., F.H. HSU, M.C. CHEN, Y.C. HSU, Developing Recommender Systems With the Consideration of Product Profitability for Sellers, Information Sciences, Vol 178, (2008), pp. 1032–1048.

[4] CHEN, Y., L. ZHOU, D. ZHANG, Ontology-Supported Web Service Composition: An Approach to Service-Oriented Knowledge Management in Corporate Services, Database Management Vol 17, (2006), pp. 67–84.

[5] O'SULLIVAN, J., D. EDMOND, A.T. HOFSTEDE, What's in a service? Distributed and Parallel Databases Vol 12, (2002), pp. 117–133.

[6] KO, J.M, C.O. KIM, I. KWON, Quality-of-Service Oriented Web Service Composition Algorithm and Planning Architecture, Systems and Software, Vol. 81, (2008), pp. 2079–2090.

[7] WANG, P., K.M. CHAO, C.C. LO, On Optimal Decision for Qos-Aware Composite Service Selection, Expert Systems with Applications, Vol 37, (2010), pp. 440–449

[8] RAN, S., A Model for Web Services Discovery with QoS, ACM SIGecom Exchanges, Vol 4, 2003, pp. 1 – 10.

[9] CANFORA, G., M.D. PENTA, R. ESPOSITO, M.L. VILLANI. An Approach for Qos-Aware Service Composition Based on Genetic Algorithms. Proc. Int. Conf. on Genetic and evolutionary computation, Washington DC, USA, (2005), pp. 1069–1075.

[10] HUANG, A.F.M., C.W. LAN, S.J.H. YANG, An Optimal Qos-Based Web Service Selection Scheme, Systems and Software, Vol 81, (2008), pp. 2079–2090.

[11] AI, W.H, Y.X. HUANG, H. ZHANG, N. ZHOU, Web Services Composition and Optimizing Algorithm Based on QoS, Proc. Int. Conf. on Wireless Communications, Networking and Mobile Computing, Dalian, (2008), pp. 1-4.

[12] LIU, Z., J. LI, J, LI, A. AN, J. XU, A Model for Web Services Composition Based on Qos and Providers' Benefit, Proc. Int. Conf. on Wireless communications, networking and mobile computing, Beijing, China, (2009), pp. 4562-4565.

[13] SIRIN, E., B. PARSIA, D. WU, J. HENDLER, D. NAU, HTN Planning for Web Service Composition Using SHOP2, Web Semantics Vol 1, (2004), pp. 377–396.

[14] D'AMBROGIO, A, A Model-driven WSDL Extension for Describing the QoS of Web Services, Proc. IEEE Int. Conf. on Web Services, (2006), pp. 789 – 796.

[15] LIANG, W.Y., C.C. HUANG, H.F. CHUANG, The Design With Object (DWO) Approach to Web Services Composition, Computer Standards & Interfaces, Vol 29, (2007), pp. 54-68.

[16] FERCHICHI, S.E., K. LAABIDI, S. ZIDI Genetic Algorithm and Tabu Search for Feature Selection, Studies in Informatics and Control, Vol. 18, No. 2, (2009).

[17] AARTS, E.H.L., P.J.M. VAN LAARHOVEN, Simulated Annealing: Theory and Applications, D. Reidel Publishing Company, (1987).

[18] CHAISEMARTIN, P., G. DREYFUS, M. FONTET, E. KOUKA, P. LOUBIÈRES, SIARRY P., Placement and Channel Routing by Simulated Annealing: Some Recent Developments, Computer Systems Science and engineering, Vol. 41, (1989).

[19] METROPOLIS, N., A.W. ROSENBLUTH, M.N. ROSENBLUTH, A.H. TELLER, E. TELLER, Simulated Annealing, J. Chem. Phys. 21, (1953).

[20] PATNAIK, S., Genetic Algorithms: A Survey, IEEE computer society, Vol. 27, No. 6, pp.17-26, (1994).

[21] ZOMAYA, P.F., Parallel Genetic Algorithms, Parallel & Distributed Computing, Handbook, McGraw Hul, (1996).

[22] RAJKUMAR, R. , P. SHAHABUDEEN, P. NAGARAJ, S. ARUNACHALAM, T. PAGE, A Bi-Criteria Approach to the M-machine Flowshop Scheduling Problem, Studies in Informatics and Control, Vol. 18, No. 2, (2009).

[23] DRIDI, H., R. KAMMARTI, M. KSOURI, PIERRE BORNE, A Genetic Algorithm for the Multi-Pickup and Delivery Problem with Time Windows, Studies in Informatics and Control, Vol. 18, No. 2, (2009).

[24] KAMMARTI, R., I. AYACHI , M. KSOURI, P. BORNE, Evolutionary Approach for the Containers Bin-Packing Problem, Studies in Informatics and Control, Vol. 18, No. 4, (2009).

[25] BOUKEF, H., M. BENREJEB, P. BORNE, A Proposed Genetic Algorithm Coding for Flow-ShopScheduling Problems, International Journal of Computers, Communications & Control, Vol. 2 , No. 3, (2007), pp. 229-240.

[26] CUBILLOS, C., E. URRA, N. RODRÍGUEZ, Application of Genetic Algorithms for the DARPTW Problem, International Journal of Computers, Communications & Control, Vol. 4, No. 2, (2009), pp. 127-136.