SFL Algorithm for QoS-based Cloud Service Composition

Ali Younes
Abdelmalek Essaadi University
Tetouan, Morocco

Mohamed Essaaidi
Abdelmalek Essaadi University
Tetouan, Morocco

Ahmed El Moussaoui
Abdelmalek Essaadi University
Tetouan, Morocco

ABSTRACT
With the advent of cloud service-based applications and Software as a Service (SaaS), new applications have recently known an increasing use of service-oriented architecture (SOA). This model has allowed computer science and associated industries to build new customized applications, by using the available and the existing cloud services bridged together dynamically to form a complex workflow process with more functionalities. However cloud services with similar and compatible functionalities may be offered by multiple providers but may also be offered at different QoS levels. Hence, to build a composite service with a high QoS, a decision should be made based on end-to-end QoS. This work proposes a new approach, for QoS-aware cloud service composition, which addresses a universal model, with end-to-end QoS. It also proposes an effective evolutionary method based on Shuffled Frog Leaping Algorithm (SFLA), which is satisfying global and local constraints. Therefore, in order to evaluate the robustness of the proposed approach, we have evaluated the impact of several parameters that are highly significant in evolutionary methods, such as the impact of the population size, number of candidate services per task and number of criteria. The experimental results show that the chosen algorithm performs better than the ones based on Genetic Algorithm (GA).

Keywords
Cloud services composition, QoS optimization, SFLA, GA.

1. INTRODUCTION
With the adoption of service oriented architecture (SOA) paradigm and the concept of Software as a Service (SaaS), cloud resources have been encapsulated as "services" in the form of virtualized resources, which are offered through many middleware infrastructures in the Internet [1], [2]. Hence, it has enabled building new dynamic and flexible applications by invoking and integrating the existing services hosted by multiple providers to form a complex workflow process with more functionality. As users require more precise and accurate results in a short time with a high Quality of Service (QoS), the QoS factor is used to state the quality of services by service providers, which refer to the nonfunctional properties of service, such as price, response time, availability, reputation, security and so on[3].

However, a single service can provide valuable functionalities for consumers, but most of the time, a single concrete service cannot satisfy them individually [1]. In this case, a composition process is requested to build a new service by using the available services bridged together that satisfy consumers' requirements, called "composite service". Therefore, many standards and models [5], [6] such as BPEL workflow and IA Planner are proposed which provide tools to design workflows process. A major limitation of these models is that they take into account only the functional control dependencies among tasks regardless of others aspects, such as Quality of Service (QoS).

When cloud infrastructures are used in public, multiple providers offer virtual resources and services often with similar and compatible functionality, but may also be offered at different QoS levels [4]. Therefore, especially in applications with online service customers, QoS-based service composition needs to be performed in a short time, and decision must be made based on end-to-end QoS.

The problem of QoS-based service composition, aims to choose in each task one service from all candidate services, hosted by multiple providers that can perform the functional requirement for this task, and maximizes the overlay utility value of the composite service. The selection of service from each task with the highest utility value does not provide a correct solution, due to the fact that this selection does not guarantee that all the end-to-end QoS are maximized. Hence, different combinations from each task need to be considered.

Therefore, this work, suggests a new model for composition process, which defines a new software component called "virtual service" that inherits all its parameters (functional and non-functional) from one single atomic service or from a set of atomic services, according to functional composition requirements. Next, in order to address the general case of the composition, a composition graph with non-identical tasks is represented which is defined by a composition matrix. Later on, we are going to scrutinize an approach based on Genetic Algorithm (GA) proposed in [7], which will be compared to the proposed approach based on Shuffled Frog Leaping Algorithm (SFLA).

The paper is organized as follows. Section 2 discusses the related work in this field. Section 3 gives an overview about existing composition models, with a focus on different composition structures and the QoS parameters. Section 4, describes our model, and gives the QoS computation for virtual service and composition process. Problem formulations and descriptions of the algorithms are presented in section 5. Section 6 shows the experimental evaluation and the comparison of different algorithms. We end by section 7 giving our conclusions on this work.

2. RELATED WORKS
The number of papers dedicated to this subject reflects a real vitality on this area of research. Various approaches have been proposed. Most of them try to improve the existing techniques used within the framework of web services orchestration like BPEL workflow or IA Planner [1] [3], [4]. Other methods [9], [12], [13], focus on semantic similarities between services parameters but it deal only with aspects of the functional

composition. However, other approaches such as [12],[13], [14], propose the computation of QoS-aware service composition by considering their similarities on the semantic level of links, they were focused on computation of QoS-aware web service composition and different techniques were proposed to handle the optimization of multi-path compositions. For example, in [13] an integer programming (IP) is proposed to solve multi criteria decision making MCDM problem, whereas method [14] resolves similar problems by using Fuzzy-MADM technique. In the same line the work [12] extends the linear programming model to include local constraints.

Linear programming methods suffer from poor scalability due to the exponential time of computation. Therefore, heuristic algorithms are applied to efficiently find a near-to-optimal solution in a reasonable tradeoff of computation time and problem size. Hence, the work in [7], proposes a heuristic technique based on a genetic algorithm to resolve the QoS-based service composition, which introduces interesting modifications, whether in crossover, mutation and selection phases in order to escape from local optimums and to expedite algorithm to deliver results in a timely manner. Otherwise, methods [18] and [19], propose a solution based on the genetic algorithm, within which Tabu search is used to generate neighbor plans and simulated annealing is applied for accepting or refusing the neighbor plan. Method [17] proposes a combination of the ant colony algorithm and the genetic algorithm, which transforms the problem of selecting optimal execution path for composite web service into a selection of the optimal path in the weighted directed acyclic graph. The work [7] has modeled the problem using the combinatorial model and the graph model. The combinatorial model defines the problem as a multidimensional multi-choice 0-1 knapsack problem (MMKP), and the graph model defines the problem as a multi-constraint optimal path (MCOP) problem, and then, proposes two heuristics algorithms to solve the problem.

Others works [6] [19], [20] propose end-to-end QoS optimization computation assuming that a certain path will be better executed than others according to the probability of paths execution. For example method [6], proposes a universal model coupled with a branch and bound algorithm, but the convergence of the algorithm is not always possible. Also, the authors in [9], propose a universal QoS model for service composition, which develops a flexible constraint satisfaction framework, and a utility function to build the objective function, and in turn, propose a branch and bound-based heuristic algorithm BB4EPS. Whereas, the work in [16], proposes an approach based on skyline method, which reduces the number of candidate services to be considered, in order to effectively select the optimal services for the composition. Whilst, [23] proposes a non-cooperative game-based mathematical model to analyze the competitive relationship between tasks, and an iterative algorithm that converges to Nash-equilibrium is proposed.

This work proposes a new approach based on a universal model, with end-to-end QoS, expected to deliver an optimal solution in a shortest time while at the same time, satisfying global and local constraints.

3. COMPOSITION MODELS

The composition or the aggregation of services is a process that involves building new services called "composite service", by assembling existing services, offered by multiple providers, in a workflow process. This process specifies which services are to be invoked in what order and under what preconditions. In composition process a "service" can be "atomic service" or "composite service". Composition can be either static or dynamic [6].

A static composition uses atomic services in an unchangeable way depending on the context of the customer [22]. However, there are two main approaches for static composition (orchestration and choreography).

Orchestration: A central coordinator composes a business process of services and is responsible for invoking them and forms a workflow. Common industry standard protocols for service orchestration are:

- XLANG (XML Busines Process Langage) of Microsoft,
- BPML (Business Process Modeling Langage) of BPMI,
- BPEL4WS (Business Process Execution Langage for Web Services), result of the grouping of IBM, Microsoft and BEA, also called BPEL or XSBPEL.

Choreography: Equal parties take part in business collaboration and communicate in a peer-to-peer model. There is no central coordinator [22]. Instead there is a conversation definition that determines the interactions between the participants. Web Services Choreography Description Language (WSCDL) is the corresponding protocol standard which exists in theory but has not been adopted widely in the industry. The common protocols found in the literature for this type of static composition are:

- WSFL (Web Service Flow Langage) of IBM,
- WSCL(Web Service Conversation Langage) of Hewlett-Packard,
- WSCI (Web Service Choregraphy Interface) of SUN.

However, this type of composition creates inflexible applications, sometimes inappropriate with customer requirements.

Dynamic composition it rather sought in applications with online service customers, which its transactions should be posted in real-time. The composition of services cannot be predefined in advance and will be done at run time also. Most of works [22] in this area, have generally formulated this issue as a problem of discovery of the semantic connection between services. Wherein race is to discover a semantic similarity between the output parameters of first service and the input parameters of next ones and most of them are limited to the functional composition aspects.

3.1 Composition structures in a workflow

Independently of composition model (static or dynamic). The composition process is in charge of building, in a workflow, a new composite service, by using atomic services, offered by multiple providers, and should be connected together by different composition structures. Figure 1 shows a composite service example, which demonstrate a brief scenario of a composition process, which the service S_1 is followed by S_2 and S_3 in XOR split composition structure (conditional), with a probability of $p_1$and $p_2$respectively, service S_5 is followed by either S_6 or S_7 in AND split structure (parallel), and service S_8 is followed by S_9 in sequential structure, and S_4 may be executed for at most K times which represent a loop structure.

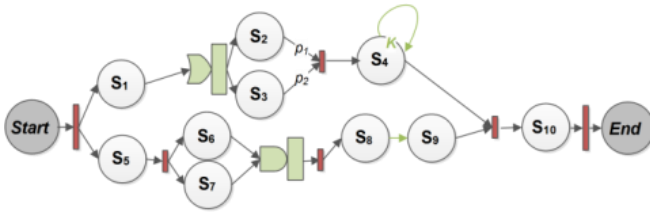

Fig 1: Composite service example

Therefore, we represent hereafter the most composition structures widely considered in literature: Sequential, AND split (fork), XOR split (conditional), Loop, AND join (Merge) and XOR join (Trigger) as shown in Figure.2.

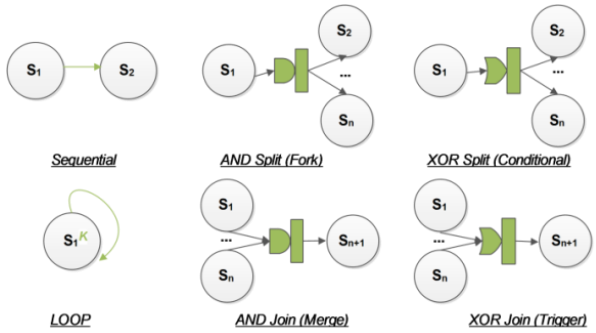

Fig 2: Composition structures in a workflow process

3.2 QoS requirement for cloud service

There are several QoS properties defined in standards ISO8402 [10] and ITU E.800 [11], which refer to the nonfunctional properties of services, such as price, response time, availability, reputation, security and so on. They are used to state the quality of services and marked by service providers. In this work four QoS properties are considered. Response Time, Cost, Reliability and Availability, are defined as follows:

- Response Time (T): is the time interval from when the service is requested and delivered to the user. It includes the total processing time of the service.
- Cost (C): is the cost required to be paid by the customer for the execution of the service. It is considered as an important parameter because certain cloud services cannot be accessed without paying for it and also these Services are costly. The cost of a service is divided in two parts: cost of transmission of request which is omitted in practice, and cost of services.
- Reliability (R): is the measurement of the services that correctly serve the users requests.
- Availability (A): is the probability that the service is accessible.

4. SYSTEM MODEL

Let's assume that each cloud service is described in one of the existing semantic languages, such as WSDL-S (Web Service Description Language-Semantics), SAWSDL (Semantic Annotations for WSDL), OWL-S (Web Ontology Language for Services) or WSMO (Web Service Modeling Ontology) [5], [6].

Definition.1: A virtual service (vs) is an abstract software component, which behaves as a uniform semantic service in a given domain and inherits all its parameters (functional and non-functional) from one atomic service or from a subset of atomic services that can be connected together by one of the composition structures such as (Fork, Loop, Conditional,…) according to functional composition requirements.

For simplification reasons, we will be limited to define the most important composition structures for(vs) commonly used in the composition standards such as (WS-BPEL workflow and IA Planner or BPML). Therefore, Figure 3 presents a simple view of generation of (vs) which can be either:

- **Identical** (vs): represented as a single atomic service,
- **Loop** (vs): represented as l iteration of one single atomic service,
- **Parallel** (vs): represented as a set of atomic service constructed to be executed at the same time,
- **Conditional**(vs): Which the compositor chooses, at run time, one atomic service from a set of atomic services with a probability p, according to the functional composition result. Where

Also, each virtual service records the relationship between its atomic services (composition structure) and performs the complex computations of its QoS parameters internally. Thus, we get rid of managing the different possible combinations of composition.

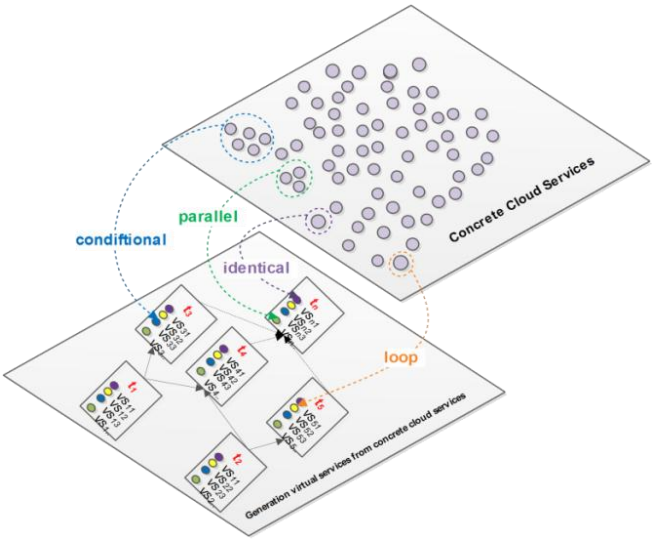

Fig 3: Generation of VS from concrete cloud services

Additionally, those virtual services are grouped, into multiple subsets called tasks, according to their functional composition. Each task is performed by the execution of one single virtual service.

Definition.2: a task (t_i) is represented by a set of virtual services which is performed by the execution of one single virtual service (vs_{ij}) that is ranged in this task. (vs_{ij}) denotes the j^{th}qualified virtual service which can be selected to accomplish the ith task.

Where, $(1 < i < n\ )$, n is the number of tasks, $(1 < j < T_i)$, (T_i) is the number of candidates virtual services arranged in the task (t_i).

Definition.3: A composition matrix CM $(K \times n)$ is a binary matrix, constructed according to the functional composition results, where K is the number of composite services is and n is the number of tasks.

$$CM[ki] = \begin{bmatrix} cm_{11} & .. & cm_{1n} \\ .. & cm_{ki} & .. \\ cm_{K1} & .. & cm_{Kn} \end{bmatrix}$$

Where,

If the i^{th}task (t_i) is selected in the k^{th} composite service (cs_k), then $cm_{ki} = 1$ otherwise $cm_{ki} = 0$.

Implicitly, each vs_{ij} participates in a composite service k it is marked by vs_{ij}^{k}. Therefore the graph of composition is done as shown in Figure 4.

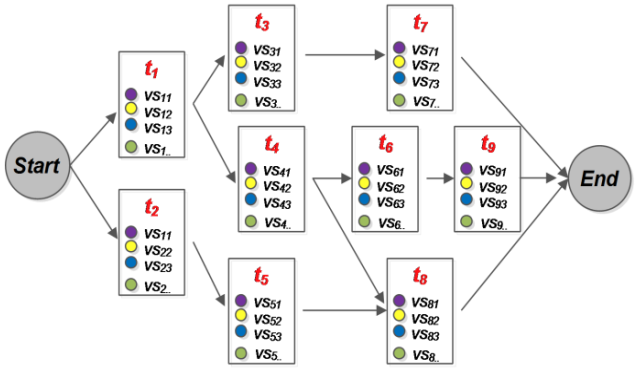

Fig 4: Composition graph example with non-identical tasks

Thus, unlike most of the existing works, which represent the composition process as a series of super classes (identical tasks) [5], [7], [8], this graph represents a universal model, based on a workflow of a series of non identical tasks. Those are performed by the execution of one virtual service in each task.

4.1 QoS Computation for a virtual service

This work is interested to make decision in order to select a composite service with a high QoS. However, there are many QoS parameters with non-uniform units and ranges measurement, which can be used to evaluate cloud services. Also, a QoS parameter may be positive or negative. The values of positive parameters need to be maximized (reputation and availability), whereas the values of negative parameters need to be minimized (execution cost and response time) [5], [6], [14]. According to these different descriptions, Table.1 gives the QoS parameters computation for an individual virtual service vs_{ij} according to its structures and QoS parameters.

Table 1. QoS Parameters Calculation Formulas for a Virtual Service vs_{ij}

Response time, Execution cost	*Identical*	$Q_q(vs_{ij}) = Q_q(s_x)$
	Loop	$Q_q(vs_{ij}) = Q_q(s_x) * l$
	Parallel (time)	$Q_q(vs_{ij}) = max(Q_q(s_x))$
	Parallel (cost)	$Q_q(vs_{ij}) = \sum_{l=1}^{nbr} Q_q(s_x)$
	Conditional	$Q_q(vs_{ij}) = \sum_{l=1}^{nbr} Q_q(s_x) * p_l$
Reputation, Availability	*Identical*	$Q_q(vs_{ij}) = Q_q(s_l)$
	Loop	$Q_q(vs_{ij}) = Q_q(s_x) * l$
	Parallel	$Q_q(vs_{ij}) = \prod_{l=1}^{nbr} Q_q(s_x)$
	Conditional	$Q_q(vs_{ij}) = \sum_{l=1}^{nbr} Q_q(s_x) \times p_l$

Where, q is an index of QoS parameters, which can be either (execution cost, response time, Reputation or Availability). And s_x is the x^{th} atomic service selected by vs_{ij}, and $1 < x < X$, and X is the total number of atomic services in the system.

$p_l = \frac{1}{\text{nbr}}$, represents the probability in which s_l is selected in conditional structure by vs_{ij}, $\sum_{l=1}^{nbr} p_l = 1$, and nbr is the number of atomic services that ranged in a parallel or conditional vs_{ij} .

4.2 QoS Computation for a composition process

In order to unify the measurement of different QoS parameters, the QoS values $Q_q(vs_{ij})$ of vs_{ij} need to be normalized before calculating the comprehensive quality of the composite service. A Max-Min normalization method is adopted, eq(1) is applied for positive parameters (reputation and availability) and eq(2) is applied for negative parameters (execution cost and response time).

$$Q_q(vs_{ij}) = \begin{cases} \frac{Q_q^{max} - Q_q(vs_{ij})}{Q_q^{max} - Q_q^{min}}, & Q_q^{max} - Q_q^{min} \neq 0 \\ 1, & Q_q^{max} - Q_q^{min} = 0 \end{cases} \quad (1)$$

$$Q_q(vs_{ij}) = \begin{cases} \frac{Q_q(vs_{ij}) - Q_q^{min}}{Q_q^{max} - Q_q^{min}}, & Q_q^{max} - Q_q^{min} \neq 0 \\ 1, & Q_q^{max} - Q_q^{min} = 0 \end{cases} \quad (2)$$

Q_q^{min} And Q_q^{max} are, respectively, the minimum and maximum values of the q^{th} attribute of all virtual services.

A composition service is represented as "sequence of tasks" , implicitly sequence of (vs_{ij}^{k}). The QoS values $Q_q(cs_k)$ of (cs_k) are determined by the corresponding QoS values of its virtual services that compose, which are selected at different tasks in a sequence model of composition. Therefore, by using the binary composition matrix presented previously and the normalized values of vs_{ij}^{k}, table 2 gives the formulas to calculate QoS parameter values for a candidate composition service cs_k as a sequence of virtual service [6], [8], and [14].

Table 2. Formulas to calculate QoS values of cs_k as sequence of vs_{ij}

Execution cost	$Q_c(cs_k) = \sum_{i=1}^{n} (Q_c(vs_{ij}^{k}) * cm(ki))$
Response time	$Q_t(cs_k) = \sum_{i=1}^{n} (Q_t(vs_{ij}^{k}) * cm(ki))$
Availability	$Q_a(cs_k) = \prod_{i=1}^{n} (Q_a(vs_{ij}^{k}) * cm(ki))$ $where\ cm(ki) \neq 0$
Reputation	$Q_r(cs_k) = \frac{1}{T_k} \sum_{i=1}^{n} (Q_r(vs_{ij}^{k}) * cm(ki))$

Where $cm(ki)$is the value of the binary composition matrix which can be 0 or 1, and $T_k$is the number of tasks that compose the composite service cs_k , $T_k \leq$ n .

5. APPROACH DESCRIPTION

Evidently, the selection of a virtual service to each task, with a highest utility value does not provide a correct solution, due to the fact that this selection does not guarantee that all the end-to-end QoS are maximized. Hence, different combinations of virtual services from each task need to be considered. In effect, finding an optimal concretization of a composite service is a Non-deterministic Polynomial-time hard (NP-hard) problem which different strategies can be adopted.

Therefore, the problem can be modeled by means of a fitness function and by the global constraints imposed by users and also by local constraints when choosing the candidate's virtual service in each task. However, the fitness function needs to maximize some QoS parameters (e.g., Reputation, Availability), while minimizing others (e.g., Response time, Execution cost).

Suppose there are x QoS parameters to be maximized and y QoS parameters to be minimized, the fitness function of a composite service cs_k is defined by a weighted sum method which can be formulated as:

$$\boldsymbol{F}(\boldsymbol{cs_k}) = \sum_{\alpha=1}^{x} \boldsymbol{w_\alpha} * \left(\frac{Q_\alpha(cs_k)-\mu^\alpha}{\sigma^\alpha}\right) + \sum_{\beta=1}^{y} \boldsymbol{w_\beta} * \left(1 - \frac{Q_\beta(cs_k)-\mu^\beta}{\sigma^\beta}\right) \quad (3)$$

Where w_α and w_β are the weights for each QoS parameter, $0 < w_\alpha, w_\beta < 1$, and $(\sum_{\alpha=1}^{x} w_\alpha + \sum_{\beta=1}^{y} w_\beta = 1)$.

σ and μ are the average and the standard deviation of QoS values for all composite services.

The mathematical formulations of the QoS-based service composition are as follows:

$$\boldsymbol{Max}\left(\boldsymbol{F}(\boldsymbol{cs_k})\right) \quad (4)$$

Subject to

$$\begin{cases} \boldsymbol{Q_\alpha}(\boldsymbol{cs_k}) \geq \boldsymbol{Q_\alpha^{Constraint}}, & \alpha = (Reputation, Availability) \\ \boldsymbol{Q_\beta}(\boldsymbol{cs_k}) \leq \boldsymbol{Q_\beta^{Constraint}}, & \beta = (time, cost) \\ \sum_{i=1}^{T_i} \boldsymbol{vs_{ij}^k} = \boldsymbol{1}, & \forall\, k \in CS = (cs_1, cs_2, \dots cs_k, \dots, cs_K) \end{cases}$$

In the next subsections, we are going to investigate two algorithms to find a composite service that maximizes end-to-end QoS parameters. The first is based on a Genetic Algorithm (GA) constructed by using the proposed approach in [8] and the second is an improved algorithm based on Shuffled Frog Leaping Algorithm (SFLA).

5.1 Genetic Algorithm

In this part we use the GA method proposed in [7]. Besides, in order to address the general case and to align our model, some adjustment and parameterization are added to this method. Thus, each chromosome represents by a composite service cs_k (a potential solution of our problem). As shown in figure 5, each chromosome is defined by a set of items (tasks), which are selected according to the binary value {0,1} of the composition matrix. Each task, in turn, contains an index to its set of virtual services, which are candidates also to be selected to perform this composite service.

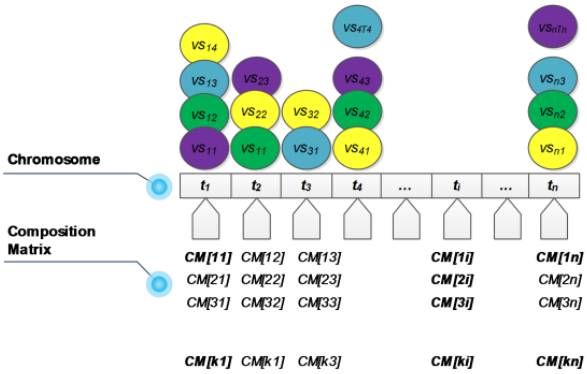

Fig.5. Genotype encoding with participating tasks for each composite service

At first, the virtual services candidates for each task are sorted according to their local values by using eq (5).

$$Q_{local}(vs_{ij}^k) = \sum_{\alpha=1}^{x} w_\alpha * Q_\alpha(vs_{ij}^k) + \sum_{\beta=1}^{y} w_\beta * (1 - Q_\beta(vs_{ij}^k)) \quad (5)$$

Next step, 20% of all genomes are selected from 20% of best virtual services that have high local value and 80% is selected randomly, each chromosome is associated with the fitness value, which is calculated based on the fitness function defined by eq(3). Once chromosomes are defined as described in Figure.5, they reproduced the population by performing genetic operation such as crossover, selection and mutation, exactly as presented in [7]. But, unlike what has been stated in this method, the algorithm was modified so as to find the values of $\boldsymbol{F}(\boldsymbol{cs_k})$ utmost.

5.2 Shuffled Frog Leaping Algorithm

Shuffled frog leaping algorithm (SFLA), is introduced by Eusuff and Lansey [15], is a meta-heuristic evolutionary algorithm, it is inspired from mimicking the behavior of frogs searching for food placed on separate stones haphazardly positioned in a pond that has a maximum quantity of food [22], [23]. SFLA is designed to seek a global optimal solution, which combines the benefits of a gene-based Memetic Algorithm (MA) and social behavior-based particle swarm optimization (PSO).

SFLA is especially used for continuous optimization problem. Nevertheless, it is adjusted by integrating crossover operations analogous as in genetic algorithm [7]. The initial population in SFLA consists of a set of frogs (candidate solutions) that is partitioned into several groups (memeplexes). In each memeplex, frogs perform a local search, conduct local exploration of solution space. After a predefined number of memetic evolution steps, the information's are passed between groups for interchange information's in the shuffling process. The local search and the shuffling process are carried out alternatively until the convergence criterion is satisfied [15].

Frogs representation is shown in figure 6, which each individual frog represents a feasible solution (a composite service), is encoded as a set of sequence of virtual services, and each memeplex, is represented by group of sequence tasks which we can performs a local search, and integrates crossover operation. The crossover operation is performed, in the same memeplex, according to a randomly selected position, by combining the former part of a first task with the latter part of another. Hence, the main parameterization and adjustment of SFLA is described in figure 7.

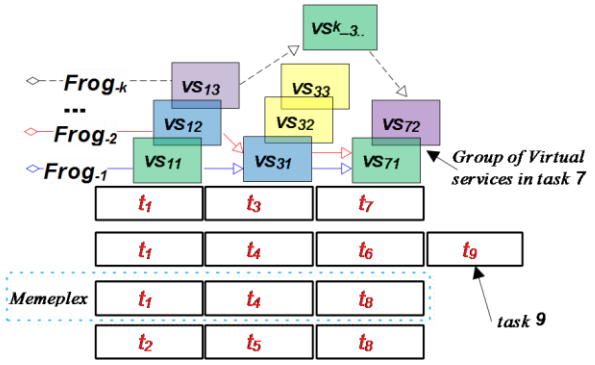

Fig. 6. Frogs and memeplexes representation

The initial population of SFLA is selected randomly from a set of composite services $CS \equiv \{cs_1, cs_2, \ldots, cs_k, \ldots, cs_K\}$ represents a set of K frogs. Next, the frogs are sorted in a descending order according to their fitness function by using (3). Then, the entire population is divided into various groups, M memeplexes, each containing N frogs, $K = N \times M$. Each memeplex performing a local search, within each memeplex, the individual frogs hold ideas, which can be influenced by the ideas of other frogs, and evolve through a process of memetic evolution. In this process, the first frog goes to the first memeplex, the second frog goes to the second memeplex, frog M goes to the QUOTE M^{th} M^{th} memeplex, and frog $M + 1$ goes to the first memeplex, and so on.

Within each memeplex (Local search), the frogs with the best and the worst fitness are identified as F_b and F_w respectively. Also, the frog with the global best fitness is identified as F_g. Then, an evolution process (crossover operation) is applied to improve only the frog with the worst fitness in each cycle. Accordingly, the position of the frog with the worst fitness is adjusted as follows:

Change in frog position

$$\boldsymbol{D_i = rand(\,) \times (F_b - F_w)} \qquad (6)$$

New position

$$\boldsymbol{F_w = current\ position\ F_w + D_i} \qquad (7)$$

Where rand( ) is a random number between 0 and 1, $D_{max} \leq Di \leq Dmin$, and Dmax is the maximum allowed change in a frog's position. If this process produces a better solution (frog), it replaces the worst frog. Otherwise, the calculations in equations (6) and (7) are repeated with respect to the global best frog (F_g replaces F_b). If no improvement becomes possible in this latter case, then a new solution is randomly generated to replace the worst frog with another frog having any arbitrary fitness. The calculations then continue for a specific number of evolutionary iterations within each memeplex.

6. EXPERIMENTAL EVALUATION

To evaluate the efficiency of the proposed methods, we have accomplished several experiments to test our algorithms, the simulations were carried out using Java language on a Pentium 2.70GHz CPU and 4 GB of RAM desktop personal computer with Windows 7, and all the simulation values are done at an average of 10 executions.

In each simulation, we initialize the parameters of the experience. Firstly we generate randomly a number of atomic services and for each service its QoS attributes value (response time, cost, availability, and reliability) randomly also, within a common range for each parameter. Next, according to the parameters of the simulation, a number of (VS) is generated belong to their structures (Identical, Loop, Parallel, Conditional), and assigned to their tasks. Later, the binary composition matrix is constructed which represent the participating tasks for each composite service. Finally, we run our implementation for the both algorithms.

6.1 Success ratio

The purpose of the first experiment was to determine and ensure that our algorithms are capable of giving results that represent the optimal solution. We have explicitly added a series of services that compose a composition process with a high quality of services. This experience is performed with a very large population size, and is executed in 10 separating times, as shown in Figure.8, which demonstrates the goodness and performance of our methods either for SFLA or GA.

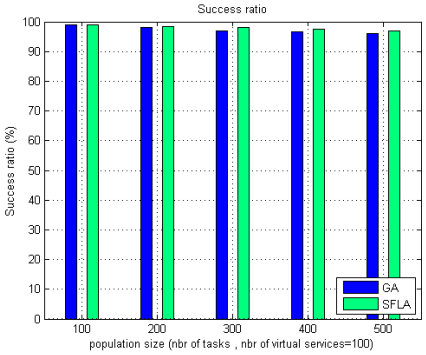

Fig. 8. Success ratio

6.2 Computation time with respect to the number of tasks (Population size)

In the next experiment, in order to study the performance of our algorithms, we created sets of randomly generated test cases, with a varying parameter in each test that influences the performance of the algorithms. We analyzed the impact of varying the number of tasks, the number of candidate services and the number of constraints. Each set of test cases is solved by comparing the computation time of GA and SFLA.

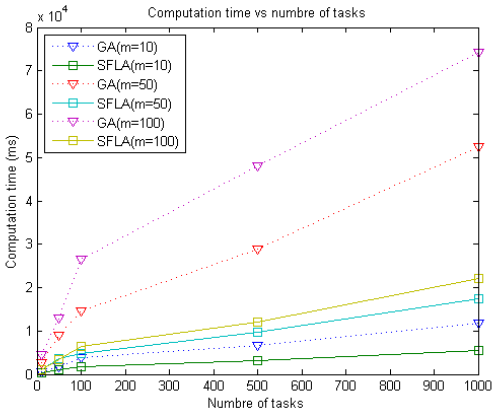

Fig.9. Computation time vs number of tasks

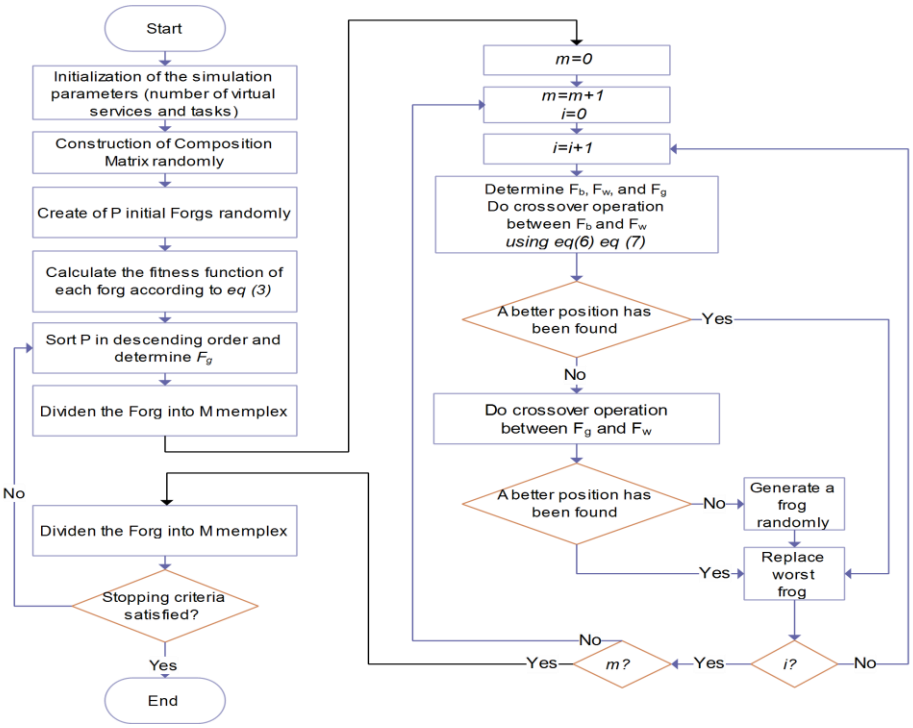

Fig 7: Flowchart of SFLA

The first test was performed with a series of tasks (10, 50, 100, 500, 1000), for each task we have three series of candidate services (10, 50, 100), the performance comparison result as shown in Fig.9 demonstrates that the population sizes have a significant factor on the execution time, thus the SFLA in all cases, required lower time for the optimization when compared to GA approximately with a speedup to 36%.

6.3 Computation time with respect to the number of virtual services

In another separate run, a simulation is performed, by a varying number of candidate virtual services in which the number of tasks is (10, 20 and 30) as shown in Figure.10. It is obvious that the execution time of the two methods increases when the population size increases and the overall execution time obtained by SFLA is always lower than GA.

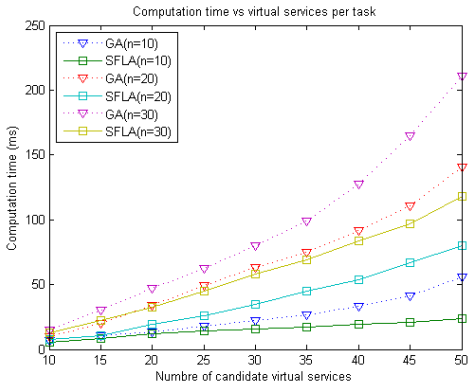

Fig. 10. Computation time vs virtual services per task

Moreover, we noticed that the computation time gap between GA and SFLA increases linearly with increasing the number of candidate virtual services, which proves also that the local search used by SFLA is more efficient than the one used by GA. This clearly demonstrates the effectiveness of SFLA compared to GA whether in the local search or in the global optimization.

6.4 Computation time with respect to the number QoS constraints

In the third experiment as shown in Figure.11, we measured the performance of the different methods with respect to the number of QoS criteria. An expected result with a very constrained problem is that it is very probale that GA and FLSA methods will need more and more iteration until a solution is found.

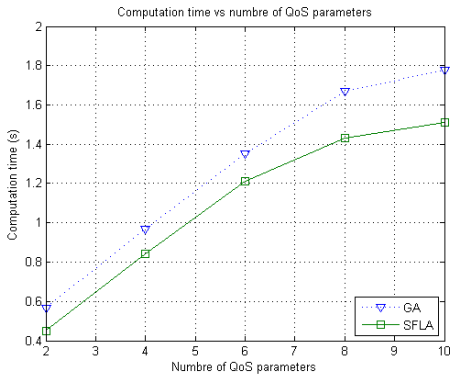

Fig. 11. Computation time vs number of QoS parameters

According to this simulation results SFLA proves to be efficient in the area of QoS-aware cloud-service composition, and it can't only find an optimal solution, but has also a better convergence speed. Thus, we deduce that SFLA is more adapted for QoS-based service composition.

7. CONCLUSION

This paper, addresses the problem of QoS-based cloud service composition with end-to-end QoS. It suggests a universal

composition model which starts by defining an abstract component that represents the basic and the most important composition structures commonly used in the service composition standards. The proposed composition matrix allows this model to support any type of functional composition which can represent all possible composition structures. To find the composite service with an optimal QoS, two evolutionary-based research methods have been presented (GA and SFLA). A description of each method has been presented. Impact of significant parameters in evolutionary algorithms such as the population size, the number of candidate services per task and the number criteria has tested. The experimental results show that the SFLA-based method in all scenarios performs better than GA either in term of success rate or in term of computational time.

8. REFERENCES

[1] L. Min, Z. Liang-Jie, and L. Fengyun, "An Insuanrance Model for Guaranteeing Service Assurance, Integrity and QoS in Cloud Computing," Proceedings of the 8th IEEE International Conference Web Services (ICWS 2010), 2010, pp. 584-591.

[2] A. Younes, M. Essaaidi, A. El Moussaoui, A. Bendahmane, “Grid computing middleware information systems: Review and synthesis study”, in International Conference on Multimedia Computing and Systems, 2009. ICMCS '09, 2-4 April 2009.

[3] S. Distefano, A. Puliafito, M. Rak and S. Venticinque, “QoS Management in Cloud @ Home Infrastructures”, International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, IEEE (2011).

[4] O. Kayed Qtaish, Z. BtJamaludin, M.Mahmuddin, “Multi-Path QoS-Aware Service Composition”, International Journal of Engineering Research and Applications (IJERA), ISSN: 2248-9622, Vol. 2, Issue 2, Mar-Apr 2012, pp.1075-1085.

[5] Min Liu, Mingrui Wang, Weiming Shen,Nan Luo, Junwei Yan, “A quality of service (QoS)-aware execution plan selection approach for a service composition process”, Future Generation Computer Systems, 28 (2012) 1080–1089

[6] Yu, T., Zhang, Y., and Lin, K.-J. 2007. “Efficient algorithms for Web services selection with end-to-end QoS constraints”, ACM Trans. Web 1, 1, Article 6 (May 2007),

[7] M. AllamehAmiri, V. Derhami, M. Ghasemzadeh, “QoS-Based web service composition based on genetic algorithm”, Journal of AI and Data Mining, Vol. 1, No.2, 2013, 63-73

[8] ISO8402: Quality management and quality assurance vocabulary, 1994.

[9] ITU-T Recommendation E.800, Terms and Definitions Related to Quality of Service and Network Performance Including Dependability, 09/2008.

[10] F. lecue, A. delteil, A. leger, "Optimizing causal link based Web service composition ", in ECAI, 2008 45-49

[11] 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.

[12] A. Younes, M. Essaaid,A. ElMoussaoui, A. Bendahmane, "A fuzzy MADM approach for grid services composition," Multimedia Computing and Systems (ICMCS), 2011 International Conference, ICMCS.2011

[13] M. Eusuff and K. Lansey, “Optimization of water distribution network design using the shuffled frog leaping algorithm,” Journal of Water Resources Planning and Management, vol. 129, no. 3, pp. 210–225, 2003

[14] M. Alrifai, D. Skoutas, T. Risse, “Selecting skyline services for QoS-based web service composition”, in WWW 2010, pp 11-20

[15] Y. Zongkai, S. Chaowang,L. Qingtang, Z. Chengling,“A Dynamic Web Services Composition Algorithm Based on the Combination of Ant Colony Algorithm and Genetic Algorithm”, Journal of Computational Information Systems, 6:8(2010) 2617-2622.

[16] M. Jaeger and G. Muhl. “QoS-Based Selection of Services: The Implementation of a Genetic Algorithm”, In KiVS 2007 Workshop: Service-Oriented Architectures und Service-Oriented Computing (SOA/SOC), Bern, Switzerland, pages 359{370, 2007

[17] FERCHICHI, S.E, K. LAABIDI, S. ZIDI, “Genetic Algorithm and tabu Search for Feature Selection”, Studies in Informations and Control, Vol. 18, No. 2, (2009)

[18] R. Wang, C.-H. Chi, and J. Deng, “A Fast Heuristic Algorithm for the Composite Web Service Selection”, Advances in Data and Web Management, 5446 (Heidelberg: Springer Berlin 2009) 506-518.

[19] A. Klein, F. Ishikawa, and S. Honiden. “Efficient, Heuristic Approach with Improved Time Complexityfor QoS-aware Service Composition”, In IEEE, International Conference on Web Services (ICWS 2011), pages 436{443, 2011.

[20] Ravi Khadka, Bramhananda Sapkota, Luís Ferreira Pires, Marten van Sinderen, Slinger Jansen, Model-driven approach to enterprise interoperability at the technical service level, Computers in Industry, Volume 64, Issue 8, October 2013, Pages 951-965.

[21] Xue-hui, YANG Ye, LI Xia ,”Solving TSP with Shuffled Frog-Leaping Algorithm”, Eighth International Conference on Intelligent Systems Design and Applications , ISDA, Kaohsiung, Taiwan, November 26-28, 2008 (ISBN: 978-0-7695-3382-7).

[22] Antariksha Bhaduri, “A Clonal Selection Based Shuffled Frog Leaping Algorithm”, 4th Annual IEEE Conference. International Advance Computing Conference, IACC, Patiala, India, March 2009 (ISBN: 978-1-4244-2927-1).

[23] Haifeng Li, Qing Z, Xiaoxia Y, Linrong X, “Geo-information processing service composition for concurrent tasks: A QoS-aware game theory approach”, Computers & Geosciences, 47, (2012) 46-56.