QoS-Based Service Optimization using Differential Evolution

Florin-Claudiu Pop
Technical University of Cluj-Napoca, Romania
florin.pop@com.utcluj.ro

Denis Pallez
University of Nice Sophia-Antipolis, France
denis.pallez@unice.fr

Marcel Cremene
Technical University of Cluj-Napoca, Romania
cremene@com.utcluj.ro

Andrea G. B. Tettamanzi
Università degli Studi di Milano, Italy
andrea.tettamanzi@unimi.it

Mihai Suciu
Babes Bolyai University of Cluj-Napoca, Romania
mihai.suciu@ubbcluj.ro

Mircea Vaida
Technical University of Cluj-Napoca, Romania
mircea.vaida@com.utcluj.ro

ABSTRACT

The aim of our research is to find an efficient solution to the services QoS optimization problem. This NP-hard problem is well known in the service-oriented computing field: given a business workflow that includes a set of abstract services and a set of concrete service implementations for each abstract service, the goal is to find the optimal combination of concrete services. The majority of recent proposals indicate the Genetic Algorithms (GA) as the best approach for complex workflows. We propose a new approach, based on Differential Evolution (DE) and a new genome encoding. The results show that proposed algorithms converge faster than the existing ones based on Genetic Algorithms with integer genome encoding.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Algorithms, Experimentation, Performance

Keywords

Services, Composition, QoS, Optimization, Selection,Genetic Algorithms, Differential Evolution

1. INTRODUCTION

Service Oriented Architecture (SOA) implementations have become more and more popular, diverse and widespread in enterprise distributed environments. This fact is due to their technical advantages over more traditional methods of distributing computing. These advantages include: delivering application functionality as services across several platforms, providing location independence, authentication and authorization support and dynamic search and connectivity to other services.

New services can be created by combining the functionality of existing services. This process is called *service composition* and the resulted services are called *composite services*. An important requirement for a composite service is to be able to efficiently select and integrate, at runtime, heterogeneous services from different providers.

Quality of Service (QoS) properties are part of the Service Level Agreement (SLA) between a service provider and a service requester. Two services that provide the same functionality may have different QoS properties. For example, one may be cheaper, but have a higher response time, while the other may be more expensive, but have a lower response time.

Given a composite service described by a business workflow that includes a set of abstract services, where each abstract service can be realized by several concrete services, the QoS optimization problem is to find the optimal combination of concrete services (having the best QoS). This problem is NP-hard.

Numerous existing proposals for this problem indicate Genetic Algorithms (GA) as the preferred approach. Services QoS optimization is usually done at runtime, where a fast algorithm is preferred, but GA may be slow for such a task. This fact and also the need to improve the accuracy and the exploration of the solutions space motivated us to propose a new approach, based on Differential Evolution (DE). We also propose a new genome encoding for solving a discrete problem using a continuous function optimizer, like DE. Comparative experiments are conducted. The results show that proposed algorithms converge faster than the existing ones based on Genetic Algorithms with integer genome encoding.

The next section presents the problem statement and some of the existing solutions. Section three gives some details about Differential Evolution. Section four presents the proposed approach based on DE. In section five we show some numerical experiments and we compare the results obtained using different DE and GA-based evolutionary algorithms. Some statistical validation tests were also performed. The last section contains the conclusion and future work.

GECCO'11, July 12–16, 2011, Dublin, Ireland.

2. SERVICES QOS OPTIMIZATION

This section presents briefly the most important service technologies, the QoS problem representation and depicts some of the solutions that were proposed for this problem.

2.1 Services technologies

Several technologies for service composition exist: WS-BPEL (Web Services Business Process Execution Language)[10], WSCI (Web Service Choreography Interface), and others. When composing a service, two types of requirements need to be considered [1]: a) functional requirements (FR), which specify the service behavior and b) non-functional requirements (NFR), which refer to the quality of a service (QoS). But the orchestration and choreography technologies concern only the functional requirements of a composite service. That is why we need to add dedicated mechanisms for handling the QoS requirements.

The most widely used standard for composing services, WS-BPEL, was chosen as service model. In WS-BPEL, a composite service is a business process (workflow) that consists in a set of activities. Such a workflow may include control structures like: *flow, sequence, switch* and *while*. *Flow* is used to define concurrent activities. A flow completes when all its activities did complete. A *sequence* is a set of activities that are executed one after the other. *Switch* selects between any number of *case* branches based on a condition. *While* is used to create conditional loops.

2.2 Problem statement

A composite service can be described as a process that involves the execution of several activities according to a workflow. An example workflow for a *flight booking process* is depicted in figure 1.

This workflow consists of the following activities: *Find Nearest Airport* for identifying the airport that is closest to the desired departure or destination location, *Propose Flight* for finding all flights that match a certain criteria, *Book Flight* for the actual purchase of the flight tickets and *Delivery* for mailing the tickets and the receipt.

Executing an activity means invoking a service. For each activity, which is assimilated to an *abstract service* (S1, S2, ... in fig. 1), several *concrete services* exist. Each concrete service has different QoS properties. For describing the QoS we use the following (widely used) parameters: *response time, reliability (r), availability (a)* and *cost (c)*.

The *response time (t)*, sometimes called *latency* is a measure for the performance of a service. It represents the round-trip time between sending a request and receiving the response. *Reliability (r)*, usually measured as the number of failures per unit of time (month or year), represents the capacity to ensure reliable message delivery for services. The probability of a service to be ready for immediate use is called *availability* (a). The *cost* (c) is the price to pay for each service request.

The QoS of the composite service is obtained by aggregating the QoS of the component services. The aggregation rules are described in the section 4.2.

Given m abstract services and n concrete services for each abstract service, there are n^m possibilities. The search space is a discrete one since for each abstract service we need to chose one concrete service and any combination is possible. We have a combinatorial optimization problem here. An exhaustive search algorithm is very inadequate because the solution should be found at runtime. Finding the solution with the optimal QoS is an NP-hard problem.

2.3 Related work

The problem stated previously is well known in domains like Service Oriented Computing (SOC) and Search-based Software Engineering (SBSE). We found it discussed in [2], [14], [7], [3] and other papers. In the literature, various solutions are proposed based on different approaches such as: integer programming (greedy algorithms), genetic algorithms and hill climbing algorithms. In this section we present what we considered the most relevant of these proposals.

Genetic algorithms versus linear programming. G. Canfora et al. [2] have compared an integer linear programming [17] based algorithm with a genetic algorithm. As a case study, they considered a workflow containing 8 distinct abstract services. The number of available concrete services per abstract service was set to: 5, 10, 15, 20 and 25. The comparison was based on the convergence time that was considered proportional to the CPU user time.

The authors used an elitist GA where only the best 2 individuals are copied to next generations, a crossover probability of 0.7, a mutation probability of 0.01 and a population of 100 individuals. The selection mechanism adopted was the roulette wheel selection.

Their conclusion was that, in contrast with linear integer programming (the widely adopted approach at the moment), GA is able to deal with QoS attributes having non-linear aggregation functions. Also, GA can scale-up when the number of concrete services per abstract service increases. When the workflow size and the number of concrete services per abstract service are limited and there is no need to use non-linear aggregation functions, integer programming is however preferable.

A genetic algorithm for services deployment optimization. Yves Vanrompay et al. [14] also propose to use genetic algorithms for mobile service composition and deployment. In this case, the problem is formulated slightly different: there is a system consisting of several nodes on which a composite service can be deployed in a distributed manner. The goal is to deploy the composite service onto a set of connected nodes in a way that the allocation meets the given QoS constraints and minimizes the communication cost between the nodes. A set of constraints are added to the problem model for specifying if a certain component can be deployed on a specific node. The authors prove that GAs provide a scalable mechanism which offers improvements over relevant solutions.

Genetic algorithms versus greedy algorithms. Liu Xiangwei et al. [7] also suggest that genetic algorithms are a good approach for semi-automatic service composition. The paper presents an independent global constrains-aware Web service composition method based on extended Color Petri net (eCPN) and a genetic algorithm (GA). The authors compared the genetic algorithm with a greedy algorithm and the conclusion was that GA has higher execution efficiency and success rate.

Weise et al. [15] also compare genetic algorithms with greedy algorithms and conclude that GAs offer a good exploration of the solutions space but they are slower than the greedy algorithm. Other advantages of the genetic algorithms approach are the generality and the extensibility.

The large majority of existing proposals indicate *genetic*

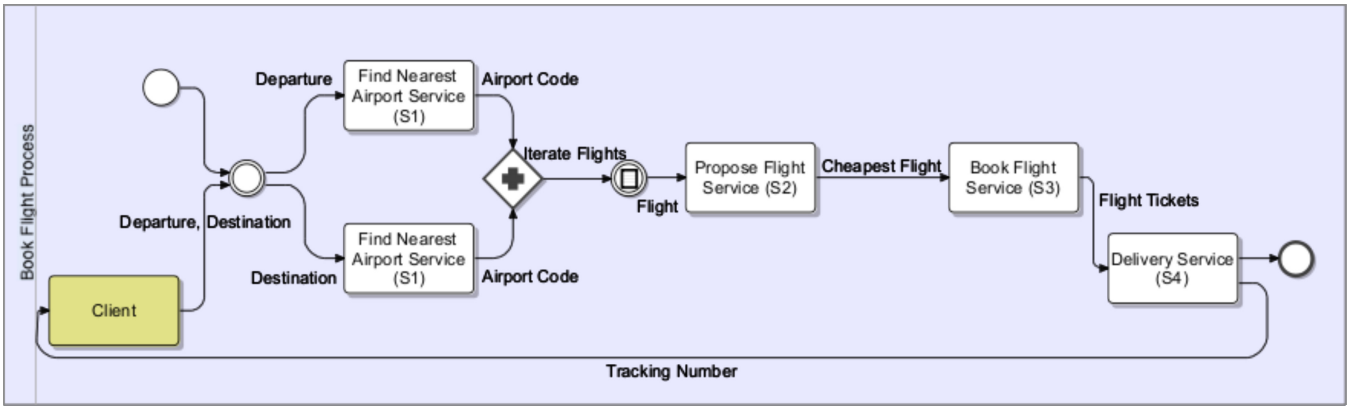

Figure 1: A flight booking abstract process

algorithms as the best approach for large search spaces: complex composite services with numerous abstract services and numerous concrete services. One of the main advantage of the GA is scalability.

Some existing research, as for instance Tusar and Filipic [13], show that for some general optimization problems, the algorithms based on Differential Evolution (DE) [11] performed significantly better than the corresponding genetic algorithms. This fact motivated us to chose a DE-based approach.

The next section introduces Differential Evolution and some of its discrete variants.

3. DIFFERENTIAL EVOLUTION

This section presents briefly some of the most important aspects about differential evolution.

3.1 Differential Evolution for continuous domains

The DE algorithm was introduced by Storn and Price [11]. DE is a population based, stochastic, and continuous function optimizer [12] where distance and direction information from the current population is used to guide the search process [4]. DE is known to be able to handle non-differentiable, nonlinear, and multimodal objective functions, to be easy to use, and to converge consistently to the global optimum in consecutive, independent trials.

Essentially, for each individual of the population (*target* vector $x_i(t)$), a *mutant* vector $m_i(t)$ is first generated by adding the weighted difference (*difference* vector) between two randomly chosen vectors (*parameter* vectors $p_{i_1}(t)$ and $p_{i_2}(t)$) to a third chosen vector (*base* vector $b_{i_3}(t)$) as follow:

$$m_i(t) = b_{i_3}(t) + F \cdot (p_{i_1}(t) - p_{i_2}(t)), \qquad (1)$$

where $i \neq i_1 \neq i_2 \neq i_3$; i_1, i_2 are randomly and uniformly chosen between 1 and the population size and $F \in \mathbb{R}^+$ is scaling factor, controlling the amplification of the differential variation.

Secondly, one child, called *trial* vector, is obtained by crossover of the mutant vector and the target vector. Finally, the target vector is replaced by the best of either the trial or target vector.

Depending on how the base vectors are selected, on how many differences contribute to the differential and on the type of crossover used, there are several strategies that can be adopted for DE. A short notation is used for each strategy. For example, one strategy can be *DE/rand/1/bin*. This means that the base vectors are randomly selected from the population, one difference vector is considered for generating the new vector and uniform crossover is used, based on a binomial distribution. See [12, 4] for a more detailed explanation.

One issue in using Differential Evolution derives from the fact that DE was originally proposed to solve problems defined in a continuous domain and the problem we want to solve is discrete. Since the objective functions we want to optimize are of the form $f : D \rightarrow \mathbb{R}$, where D is a discrete domain, DE cannot be used in its canonical form.

Several methods to apply differential evolution for discrete variables were proposed in the literature. Some of these methods are discussed below.

3.2 Discrete Differential Evolution

Most Discrete Differential Evolution (DDE) approaches fall into one of the following categories:

a) methods that use a mapping between the discrete domain and the continuous domain, allowing an unmodified DE strategy to be run. Two solutions from this category are discussed: *TruncDE* and *BinDE*.

b) methods that modify the original DE algorithm to evolve discrete variables. In DE, the basic idea is to adapt the search step along the evolutionary process. At the beginning, when the individuals are spread all over the search space, the search step is big. With each generation, the population converges towards a smaller region and the search step becomes smaller. Based on these premises, modified versions of DE for discrete domains replace the canonical mutation operator with a discrete one, which resembles the DE concept. Two such approaches are presented: *XueDE* and *HammingDE*.

1. TruncDE. Lampinen and Zelinka [6] propose a method to apply DE for integer-valued problems. They maintain floating-point variables for internal DE computations, and truncate the values when evaluating the cost-function $f_{cost}(y_i)$, where

$$y_i = \begin{cases} x_i & for\ continuous\ variables \\ INT(x_i) & for\ discrete\ variables \end{cases} \qquad (2)$$

$x_i \in \mathbb{R}$ and *INT* is a function that converts a floating-point number to an integer by truncation.

For finite discrete domains, the authors propose that instead of attributing the actual discrete values to x_i, this should store the index of the discrete value in the corresponding subset of values. Then, this problem can be handled as an integer problem.

For problems that require discrete variables and ordered sequence, rather than relative position indexing, the solution proposed by Onwubolu and Davendra [9] - based on a more elaborate transformation - can be used.

2. BinDE is inspired by the binary particle swarm optimization (BinPSO) method developed by Kennedy and Eberhart [5]. The main idea of BinDE consists in interpreting the floating-point DE individuals as probabilities that the corresponding component of the solution vector is bit 0 or 1.

$$y_{i,j} = \begin{cases} 0, & f(x_{i,j}(t)) > 0.5 \\ 1, & f(x_{i,j}(t)) \leq 0.5 \end{cases} \quad (3)$$

where $x_{i,j}$ (j = 1,..,n_x) are the floating-point components of the x_i individual, n_x is the dimension of the binary-valued problem and:

$$f(x) = \frac{1}{1 + e^{-x}}$$

3. XueDE. Xue et al. [16] replace the mutation operator of DE with a conditional operator based on three probabilities: greedy probability p_g, mutation probability p_m and crossover probability p_c. A new individual is generated with the following rule:

$$y_i = \begin{cases} x_{best_j} & r \leq p_g \\ rand(\Omega_j) & p_g < r \leq p_g + p_m \\ x_{a_j} & p_g + p_m < r \leq p_g + p_m + p_c \\ x_j & otherwise \end{cases} \quad (4)$$

where r is a random number, x_{best_j} is the individual with the highest fitness value from the population, Ω_j contains all the possible values for allele j, x_{a_j} is a randomly selected individual from parent population that is distinct with x_j.

4. HammingDE. Zhang et al. [18] adopt a binary encoding scheme for the discrete DE strategy. An individual is represented as a binary string, composed of the binary sub-strings obtained by encoding the components of the solution. The difference of two binary individuals is defined as a normalized Hamming distance between them, and is used to obtain the mutation rate p_r. This value is used to randomly disturb a third individual: implement the uniform mutation operation whose probability is p_r.

4. PROPOSED APPROACH

This section describes the proposed method based on Differential Evolution. To find the best approach, we explore 3 DE variants: *TruncDE* [6], *XueDE* [16], which were discussed in the previous section, and a new method that we introduce in this section, called *LongDE*.

The next two sub-sections present the considered genotype for each method and the fitness assignment.

4.1 Genotype

Let S_A={S_{A_1}, S_{A_2}, .., S_{A_m}} be the set of abstract services from a business workflow and S_{C_i}={$S_{C_{i,1}}$, $S_{C_{i,2}}$, .., $S_{C_{i,n}}$} the set of concrete services that can realize the abstract service S_{A_i} and $Q_{i,j}$=*(t, r, a, c)* the vector of QoS properties (*response time - t, reliability - r, availability - a* and *cost - c*) for $S_{C_{i,j}}$.

For the problem of services QoS optimization, the genome is usually encoded as a vector of integers: the ordinal value represents the identity of the abstract service and the cardinal value corresponds to the concrete service or to the execution node.

The genome we use for *TruncDE* and *XueDE* is depicted in figure 2 and was initially proposed in [2]. It consists in an array of integer values and has the length equal to the number of abstract services in S_A. Each gene stores the index of the concrete service that realizes the corresponding abstract service.

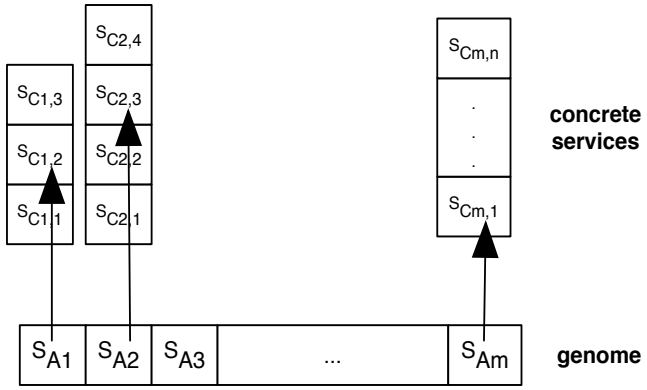

Figure 2: Genome encoding for *TruncDE* and *XueDE* [2]

Almost all DE discretization approaches involve either a bi-directional transformation from the discrete domain D to the continuous domain $\mathbb{R}$ (e.g. *TruncDE*) or attempt to modify the canonical form of DE (e.g. *XueDE*).

In this paper, we consider a new genome that facilitates discretization without changing the original DE algorithm or requiring any data transformation.

Figure 3.*a* depicts a representation of the proposed genome. A gene encodes each concrete service in S_{C_i} that can realize the abstract service S_{A_i}. The value stored in the gene represents the preference for choosing the corresponding concrete service.

For example, if the business process consists in 2 abstract services S_{A_1} and S_{A_2}, and for S_{A_1} there are 3 alternatives ($S_{C_{1,1}}$, $S_{C_{1,2}}$ and $S_{C_{1,3}}$) and for S_{A_2} there are 2 alternatives ($S_{C_{2,1}}$ and $S_{C_{2,2}}$), then the genome would be similar to the one depicted in figure 3.*b*. The greater the value of the allele, the most likely the corresponding concrete service will be selected. In our example, the preferred alternative for S_{A_1} is $S_{C_{1,2}}$, while for S_{A_2} is $S_{C_{2,1}}$.

The preferences values are initialized randomly between 0 and 1. On each generation, these preferences are updated according to the DE algorithm (see section 3.1).

For convenience, we will refer to the DE approach that uses the proposed genome with the name *LongDE*. The total

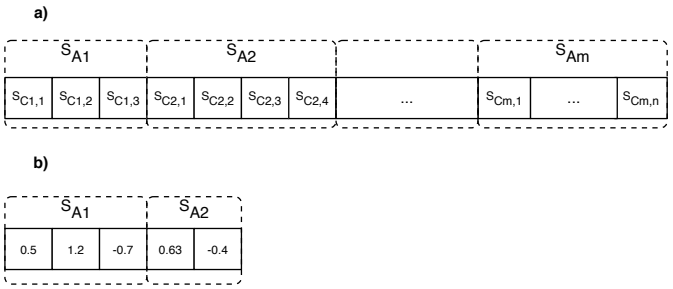

Figure 3: Solution encoding for *LongDE* - a) proposed genome, b) an example for two abstract services

number of genes required for *LongDE's* genome is given by the formula:

$$L = \sum_{i=1}^{m} n_i \tag{5}$$

where m is the number of abstract services and n_i is the number of alternatives for the abstract service i.

If n_i and m are about 10, as in a typical scenario, the genome length will be $L_{LongDE} = m \times n = 100$. In the case of *TruncDE* (or *XueDE*) the genome length is $L_{TruncDE} = m = 10$ for the same problem. Since L_{LongDE} is about 10 times bigger than $L_{TruncDE}$, *LongDE* will operate on a search space more complex than the one of *TruncDE* or *XueDE*. We study the behavior of this genotype in the next section.

4.2 Fitness assignment

The fitness is assigned to a composite service function of its QoS attributes. But the composite service QoS is not given. Thus, it is necessary to compute the QoS of a composite service starting from the QoS of the concrete services called by that composite service. This operation is called QoS aggregation [2].

The aggregation operations depend on the composite service architecture. Table 1 shows how the aggregate QoS is computed for each control structure.

For *flow* and *sequence* the QoS vector for individual services is sufficient to evaluate the aggregate QoS. For example, since *flow* means executing several activities in parallel, the total response time (R) is given by the maximum response time of all executed activities.

In case of the *switch* construct, the BPEL process needs to be monitored at runtime during multiple executions, to determine the probabilities p_i associated to each case branch, $\sum_{i=1}^{m} p_i = 1$. p_i represents the probability to select case branch i. In case of the *while* loop, the average number of iterations k is also determined during monitoring.

To evaluate the quality of each potential solution, we consider an aggregate objective function (AOF):

$$F(y) = w_1 \cdot R + w_2 \cdot A + \frac{w_3}{T} + \frac{w_4}{C} \tag{6}$$

where w_i are the weights that correspond to the importance of each QoS property to the user and R, A, T, C are the aggregate QoS values for the business workflow.

QoS Property	Flow	Sequence	Switch	While
Response Time (T)	$\max_{i \in 1..m} \{t_i\}$	$\sum_{i=1}^{m} t_i$	$\sum_{i=1}^{m} p_i \cdot t_i$	$k \cdot t$
Reliability (R)	$\prod_{i=1}^{m} r_i$	$\prod_{i=1}^{m} r_i$	$\sum_{i=1}^{m} p_i \cdot r_i$	r^k
Availability (A)	$\prod_{i=1}^{m} a_i$	$\prod_{i=1}^{m} a_i$	$\sum_{i=1}^{m} p_i \cdot a_i$	a^k
Cost (C)	$\sum_{i=1}^{m} c_i$	$\sum_{i=1}^{m} c_i$	$\sum_{i=1}^{m} p_i \cdot c_i$	$k \cdot c$

Table 1: QoS Aggregation

5. NUMERICAL EXPERIMENTS AND EVALUATION

In order to test our solution, we implemented the following algorithms:

- *TruncDE* - the DE algorithm based on Lampinen and Zelinka's proposal [6] (see 3.2) with the parameters: scaling factor $F = 0.65$, jitter $F_{NOISE} = 0.1$ and crossover constant $Cr = 0.92$. The strategy used for Differential Evolution is *DE/best/1/bin*. This means that the base vector is the best vector from the population, one difference vector is considered for generating the new vector and uniform crossover is used, based on a binomial distribution.

- *XueDE* - the DE algorithm proposed by Xue et al. [16] with the following parameters: *DE/best/1/bin* strategy, scaling factor $F = 0.95$, jitter $F_{NOISE} = 0.25$ and crossover constant $Cr = 0.95$. The probabilities for the conditional operator in equation (4) are: greedy probability $p_g = 0.1$, mutation probability $p_m = 0.65$ and crossover probability $p_c = 0.2$.

- *IntGA* - the GA algorithm proposed by Canfora et al. [2] with the parameters: *uniform* crossover where one parent is selected using *tournament selection* and the second parent is selected using *roulette-wheel selection*, the tournament size is *5*. The mutation probability suggested in [2] is $p_m = 0.01$.

- *LongDE* - the DE algorithm based on the proposed genotype, with the parameters: *DE/best/1/bin* strategy, scaling factor $F = 0.75$, jitter $F_{NOISE} = 0.15$ and crossover constant $Cr = 0.9$.

- *LongGA* - a genetic algorithm similar to IntGA with the difference that is uses the long genome that we use for *LongDE*. The parameters used for testing *LongGA* are: *two-point* crossover where the parents are selected using *tournament selection*, the tournament size is *5* and the mutation probability is $p_m = 0.025$.

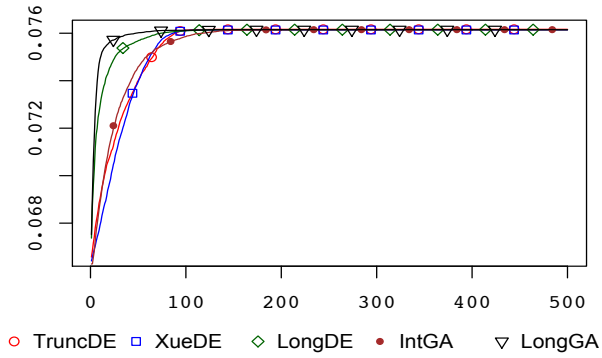

Figure 4: The evolution of the best fitness over 500 generations for m=10 abstract services and n=20 concrete services

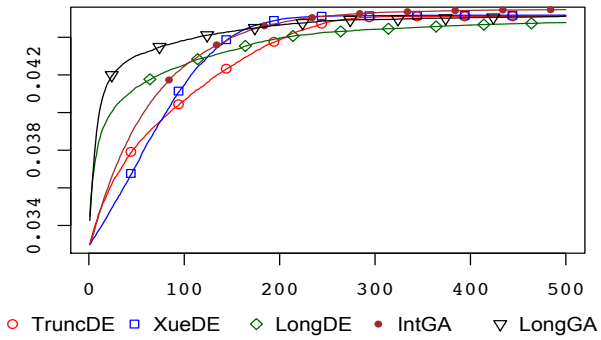

Figure 5: The evolution of the best fitness over 500 generations for m=20 abstract services and n=40 concrete services

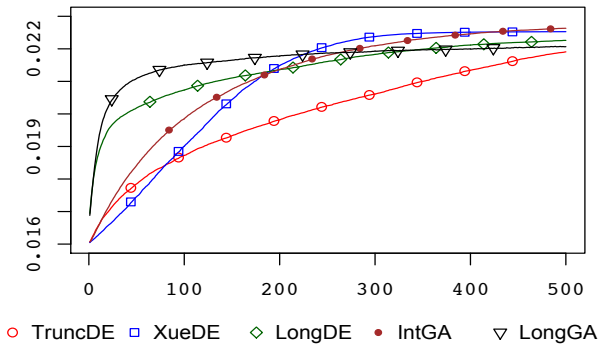

Figure 6: The evolution of the best fitness over 500 generations for m=40 abstract services and n=40 concrete services

For all these algorithms, the population was limited to 100 individuals, which were evolved for 500 generations.

We conducted experiments for 25 scenarios that include all combinations of $m \in \{10, 20, 30, 40, 50\}$ abstract services and $n \in \{10, 20, 30, 40, 50\}$ concrete services. The data for the considered scenarios was generated based on a normal distribution. Each scenario ran 100 times and the results were averaged.

All algorithms were implemented using **ECJ** [8] version 20.

Figures 4-6 show the evolution of the best fitness of the population for three most significant test scenarios.

A case with a business workflow consisting in $m=10$ abstract services, each of them having $n=20$ concrete alternative services was evaluated. The results are depicted in figure 4. We notice that all algorithms converge within 100 generations, but *LongGA* and *LongDE* are the first.

A more complex scenario, involving a business workflow consisting in $m=20$ abstract services, each of them having $n=40$ alternatives (figure 5) was tested. To converge, all algorithms require about 200 generations. *LongGA* has the best performance, while *TruncDE* is the last one that converges. *LongDE* quickly finds good individuals, but then the fitness increases at a slow rate.

For our last experiment we used a business workflow that generates a very complex optimization problem: $m=40$ abstract services, each of them having $n=40$ alternatives (figure 6). The results were similar to the previous scenarios, with the following differences: in long term (300 generations) *LongGA* and *LongDE* are slightly surpassed by *XueDE* and *IntGA*. Also, as the complexity increases, *TruncDE*'s rate of convergence decreases.

Of more practical interest is the time required to find a good solution. For this purpose, we measured the time necessary for each algorithm to find an individual that has a fitness value over a predefined threshold. Results are shown in figure 7. We notice that *LongGA* is the fastest for the majority of the considered scenarios, while *IntGA* is the slowest.

Statistical experiments were conducted using the data from the first 500 generations of the 25 considered scenarios in order to test the significance of the results. For each generation, the best and the mean fitness was considered. These values, from a series of 100 test runs were averaged. The mean (μ) and the standard deviation (σ) of the best and the mean fitness for 9 most significant scenarios are shown in table 2.

6. CONCLUSION

This paper proposes a new solution, based on Differential Evolution, for the well known NP-hard problem of composite services optimization based on QoS properties. To solve this problem, we implement two Discrete DE algorithms from the literature (*TruncDE* [6] and *XueDE* [16]), then we propose a new DE variant – which we call *LongDE*, based on a genotype that facilitates discretization. We compare these algorithms with the genetic algorithm proposed by Canfora et al. in [2] – *IntGA* – and a modified version of it that uses the proposed genotype – *LongGA*.

The results show that the algorithms based on the proposed genotype outperform algorithms for scenarios of low

m/n		TruncDE mean	TruncDE best	XueDE mean	XueDE best	IntGA mean	IntGA best	LongDE mean	LongDE best	LongGA mean	LongGA best
10/10	μ	7.85E-02	7.85E-02	7.86E-02	7.86E-02	7.86E-02	7.86E-02	7.81E-02	7.86E-02	7.66E-02	7.81E-02
	σ	1.91E-04	1.91E-04	9.29E-05	9.29E-05	1.24E-05	0.00E+00	1.15E-04	0.00E+00	7.76E-04	7.95E-04
10/20	μ	7.54E-02	7.54E-02	7.54E-02	7.54E-02	7.54E-02	7.54E-02	7.50E-02	7.54E-02	7.42E-02	7.54E-02
	σ	1.88E-05	1.88E-05	8.13E-05	7.94E-05	1.22E-06	0.00E+00	1.06E-04	0.00E+00	2.91E-04	0.00E+00
10/40	μ	9.00E-02	9.00E-02	8.96E-02	8.98E-02	8.81E-02	8.95E-02	8.92E-02	9.01E-02	8.74E-02	9.00E-02
	σ	2.57E-04	2.52E-04	4.94E-04	4.68E-04	8.23E-04	5.34E-04	2.48E-04	4.00E-05	5.65E-04	3.22E-04
20/10	μ	3.92E-02	3.92E-02	3.93E-02	3.93E-02	3.87E-02	3.92E-02	3.90E-02	3.93E-02	3.80E-02	3.88E-02
	σ	1.48E-04	1.48E-04	3.92E-05	3.92E-05	4.25E-04	1.92E-04	5.27E-05	0.00E+00	4.24E-04	4.60E-04
20/20	μ	3.77E-02	3.77E-02	3.77E-02	3.77E-02	3.77E-02	3.77E-02	3.75E-02	3.77E-02	3.71E-02	3.77E-02
	σ	7.98E-06	7.98E-06	4.44E-05	4.40E-05	2.13E-05	0.00E+00	4.38E-05	0.00E+00	1.48E-04	0.00E+00
20/40	μ	4.47E-02	4.47E-02	4.46E-02	4.47E-02	4.38E-02	4.43E-02	4.46E-02	4.50E-02	4.31E-02	4.47E-02
	σ	3.30E-04	3.29E-04	3.23E-04	3.16E-04	4.14E-04	2.17E-04	1.36E-04	7.56E-05	7.52E-04	7.69E-04
40/10	μ	1.96E-02	1.96E-02	1.96E-02	1.96E-02	1.84E-02	1.90E-02	1.95E-02	1.96E-02	1.87E-02	1.92E-02
	σ	6.55E-05	6.58E-05	2.04E-05	2.04E-05	1.58E-04	2.27E-04	1.93E-05	0.00E+00	2.61E-04	2.67E-04
40/20	μ	1.81E-02	1.84E-02	1.88E-02	1.88E-02	1.87E-02	1.88E-02	1.87E-02	1.88E-02	1.85E-02	1.88E-02
	σ	1.09E-04	1.05E-04	1.83E-05	1.81E-05	2.94E-05	2.89E-05	1.73E-05	1.87E-06	7.59E-05	1.54E-05
40/40	μ	2.13E-02	2.17E-02	2.23E-02	2.23E-02	2.16E-02	2.20E-02	2.22E-02	2.24E-02	2.10E-02	2.18E-02
	σ	2.54E-04	1.82E-04	1.31E-04	1.14E-04	3.75E-04	2.41E-04	8.66E-05	8.10E-05	4.09E-04	4.91E-04

Table 2: The mean (μ) and standard deviation (σ) of the best and the mean fitness for each test scenario

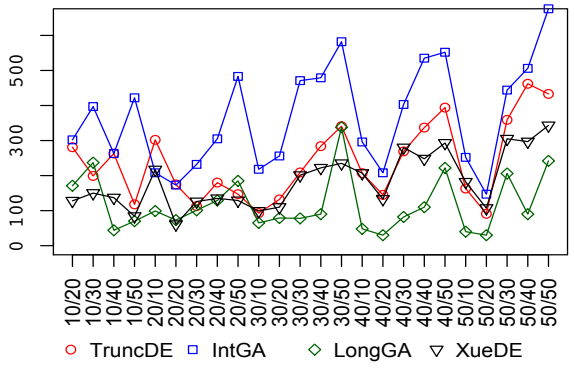

Figure 7: Time (in milliseconds) required to reach the specified fitness threshold for each scenario

and medium complexity: up to 40 abstract services, each of them having up to 40 concrete implementations. Since nowadays a typical composite service contains no more than 10 abstract services, *LongDE* or *LongGA* are the fastest options for optimization based on QoS properties.

Of all implemented DE variants, *XueDE* proved to be the most robust, having a constant behavior relative to variations of the problem complexity, the changes of evolutionary parameters (population size, mutation probability, etc.) or the QoS properties distribution. While *IntGA*'s convergence is very similar to the one of *XueDE*, the later is faster.

As future work, we intend do some more comparative experiments with other meta-heuristics such as: hill-climbing, simulated annealing and others. Another future direction is to develop a solution based on multi-objective optimization algorithms.

Acknowledgments

This project was supported by the national project code TE 252 financed by the Romanian Ministry of Education and Research CNCSIS-UEFISCSU.

Special thanks for professor D. Dumitrescu from Babes Bolyai University of Cluj-Napoca, Romania and Jean-Yves Tigly from University of Nice Sophia-Antipolis, France, for their useful contributions to this paper.

7. REFERENCES

[1] S. Andreozzi, D. Montesi, P. Ciancarini, and R. Moretti. Towards a model for quality of web and grid services. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 271–276, Washington, DC, USA, 2004. IEEE Computer Society.

[2] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO'05, pages 1069–1075, New York, NY, USA, 2005. ACM.

[3] D. Comes, H. Baraki, R. Reichle, M. Zapf, and K. Geihs. Heuristic approaches for qos-based service selection. In *ICSOC 2010, Lecture Notes in Computer Science*, 2010.

[4] A. P. Engelbrecht. *Computational Intelligence: An Introduction*. John Wiley and Sons, 2nd edition, 2007.

[5] J. Kennedy and R. Eberhart. A discrete binary version of the particle swarm algorithm. In *Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and*

Simulation'., 1997 IEEE International Conference on, volume 5, pages 4104 –4108 vol.5, Oct. 1997.

[6] J. Lampinen and I. Zelinka. *Mechanical engineering design optimization by differential evolution*, pages 127–146. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[7] X. Liu, Z. Xu, and L. Yang. Independent global constraints-aware web service composition optimization based on genetic algorithm. *Intelligent Information Systems, IASTED International Conference on*, 0:52–55, 2009.

[8] S. Luke. Ecj - a java-based evolutionary computation research system, 2010.

[9] G. Onwubolu and D. Davendra. Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171(2):674 – 692, 2006.

[10] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, April 2007.

[11] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, March 1995.

[12] R. Storn and K. Price. Differential evolution - simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[13] T. Tušar and B. Filipič. Differential evolution versus genetic algorithms in multiobjective optimization. In *Proceedings of the 4th international conference on Evolutionary multi-criterion optimization*, EMO'07, pages 257–271, Berlin, Heidelberg, 2007. Springer-Verlag.

[14] Y. Vanrompay, P. Rigole, and Y. Berbers. Genetic algorithm-based optimization of service composition and deployment. In *Proceedings of the 3rd international workshop on Services integration in pervasive environments*, SIPE'08, pages 13–18, New York, NY, USA, 2008. ACM.

[15] T. Weise, S. Bleul, D. Comes, and K. Geihs. Different approaches to semantic web service composition. In *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pages 90–96, Washington, DC, USA, 2008. IEEE Computer Society.

[16] F. Xue, A. Sanderson, and R. Graves. Multi-objective differential evolution and its application to enterprise planning. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 3535 – 3541 vol.3, 2003.

[17] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30:311–327, May 2004.

[18] M. Zhang, S. Zhao, and X. Wang. Multi-objective evolutionary algorithm based on adaptive discrete differential evolution. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 614–621, Piscataway, NJ, USA, 2009. IEEE Press.