Medical Services Optimization Using Differential Evolution

F.-C. Pop[1], M. Cremene[1], M.-F. Vaida[1], and A. Serbanescu[2]

[1] Faculty of Electronics, Telecommunications and IT, The Technical University of Cluj-Napoca, Romania
[2] The Faculty of Dental Medicine, "Iuliu Hatieganu" University of Medicine and Pharmacy, Cluj-Napoca, Romania

Abstract— **This paper proposes a method to compose and optimize medical services as business workflows. Such a workflow consists in a set of abstract services, and for each abstract service there are several concrete services. Since each medical service has different QoS (Quality of Service) parameters such as *response time*, *rating*, *distance* and *cost*, determining the optimal combination of concrete services that realize the abstract services of the business workflow is an NP hard problem. Recent proposals for solving NP optimization problems indicate the Genetic Algorithms (GA) as the best approach for complex workflows. But this problem usually needs to be solved at runtime, a task for which GA may be too slow. We propose a new approach, based on Differential Evolution (DE), that converges faster and it is more scalable and robust than the existing solutions based on Genetic Algorithms.**

Keywords— **Services, Composition, QoS, Optimization, Selection, Genetic Algorithms, Differential Evolution.**

I. Introduction

A. Background

The Service Oriented Architecture (SOA) model has become very popular in enterprise environments, where the complicated business logic is implemented by combining the functionality of various services. First, the business functions are defined. These functions represent the set of activities used to manage the assets of the organization in their various states. Then, the business functions are further decomposed into services, which implement the logic required to realize defined functions.

In software engineering, SOA defines how to discover and integrate disparate applications from different platforms into web-based applications. For example, one image processing application can be composed of several independent software components, each of them realizing a different function: enhancements, rotation, segmentation etc. and each of these components can be offered by a different *service provider*. Such a process that combines the functionality of multiple services is called *service composition*, and the resulted application is called a *composite service*.

In medicine, a service provider could be, for instance, a dental office, which offers various dental treatment services. A composite medical service can then be defined as any medical activity that requires the patient to benefit from two or more different medical services.

The patient (or the user of a service) is called a *service consumer*. A contract (formal or informal) is defined between the service provider and the service consumer to specify the level of service. This contract is called the Service Level Agreement (SLA). For example, the SLA between the dentist and the patient for a dental implant service may include the amount of time the implant is guaranteed to last, the cost of the medical procedure or the average rate of success. Such attributes represent the Quality of Service (QoS) properties of a service.

Two services that provide the same functionality often have different QoS properties. For example, many clinics offer a similar range of medical tests, but promote different prices and require different amounts of time to deliver the results. One may be cheaper, but require longer time to provide the results than a more expensive service.

B. Motivating Example

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 series of clinical tests is depicted in Fig. 1. This workflow consists of the following activities:

- *S1.* *Assisted General Diagnosis* for reading the patient's symptoms and classifying them in one of the 3 (example) categories: *Heart Disease Symptoms*, *Digestive System Symptoms* or *Other Symptoms*. According to the assigned category, the patient is then scheduled for specific medical tests.
- *S2.* *Cholesterol Test* for measuring the cholesterol level,
- *S3.* *Cardiac Exam* for investigating signs of any cardiovascular pathology,
- *S4.* *Endoscopy,* where the digestive tract is investigated,
- *S5.* *Physician Consultation* for having a physician examine the patient's symptoms and the results of the scheduled investigations,
- *S6.* *Send Test Results* that ensures the delivery of the patient's investigations result to his home and/or the location of his medical records.

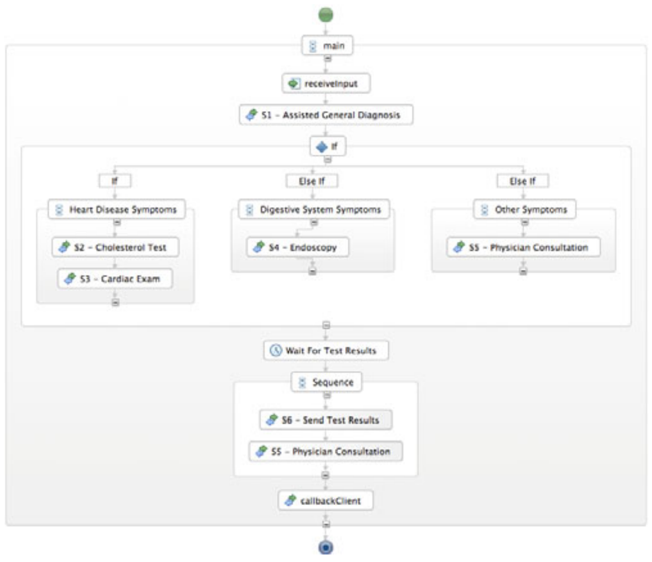

Fig. 1 An abstract process containing several clinical tests

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 parameters: *response time (t), rating (r), distance (d)* and *cost (c)*.

In software engineering, the *response time* (t) is a measure for the performance of a service. It represents the round-trip time between sending a request and receiving the response. In the medical world, the response time represents the duration for which a patient benefits from a medical service. The user *rating* (r) represents the score the patient uses to reward a medical service after accessing it. The *distance* (d) is a numerical description of how far apart are the patient and the medical service. The *cost* (c) is the price to pay for using each service.

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

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 choose 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.

Numerous existing proposals for NP optimization problems indicate Genetic Algorithms (GA) as the preferred approach. The particularity of QoS optimization is that it's usually done at runtime, where a fast algorithm is preferred. 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). According to the experimental results this method proved to outperform the GA in terms of convergence time and scalability.

C. *Outline*

The next section presents some of the existing NP optimization solutions. Section three contains some introductory aspects about Differential Evolution. Section four presents the proposed approach based on DE. In section five we show some numerical experiments and we compare the proposed method with the existing approaches. The last section contains the conclusions and future work.

II. Related work

The NP optimization 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, 3, 7, 14, 18] 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 a linear integer programming [16] 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 offers 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 algorithms* as the best approach for large search spaces: complex composite services with numerous abstract services and numerous concrete services. One of the main advantages 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 choose a DE-based approach.

III. Differential evolution

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 $p_{i1}(t)$ and $p_{i2}(t)$) to a third chosen vector (base vector $b_{i3}(t)$) as follows:

$$\mathrm{m_i(t)} = b_{i3}(t) + \mathrm{F} \cdot (p_{i1}(t) - p_{i2}(t)) \qquad (1)$$

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 $F \in \Re^+$ is the 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.

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 $f : D \to \Re$, where D is a discrete domain, DE can't be used in its canonical form.

Several methods to apply differential evolution for discrete variables were discussed in the literature [1, 6, 9, 17], two of which are discussed below: *TruncDE* and *XueDE*.

TruncDE was proposed by Lampinen and Zelinka [6] for applying DE to integer-valued problems. They maintain floating-point variables for internal DE computations, and truncate the values when evaluating the fitness function $f(y_i)$, where

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

$x_i \in D$ 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.

Xue et al. [17] 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:

$$y_i = \begin{cases} x_{best_j} & r \leq p_g \\ rand(\Omega_j) & p_g < r \leq p_g + p_m \\ x_{a_j} & p_g + p_m < r \leq p_g + p_m + p_c \\ x_j & \textit{otherwise} \end{cases} \qquad (3)$$

where r is a random number, x_{bestj} 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.

IV. Proposed approach

A. Services Technologies

Several technologies for creating and executing business workflows (such as the one depicted in Fig. 1.) exist: WS-BPEL (Web Services Business Process Execution Language) [10], WSCI (Web Service Choreography Interface), and others.

The most widely used standard for composing services, WS-BPEL, was chosen as service model. In WS-BPEL, a business process (workflow) consists in a set of activities that are executed according to some control structures. Such control structures include: *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.

B. Genotype

Let $S_A=\{S_{A1}, S_{A2}, .., S_{Am}\}$ be the set of abstract services from a business workflow and $S_{Ci}=\{S_{Ci,1}, S_{Ci,2}, .., S_{Ci,n}\}$ the set of concrete services that can realize the abstract service S_{Ai} and $Q_{i,j}=(t, r, a, c)$ the vector of QoS properties (*response time - t, rating - r, distance - d and cost - c*) for $S_{Ci,j}$.

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 encoding is depicted in Fig. 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.

C. 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.

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* executes several activities in parallel, the total response time is given by the maximum response time of all executed activities.

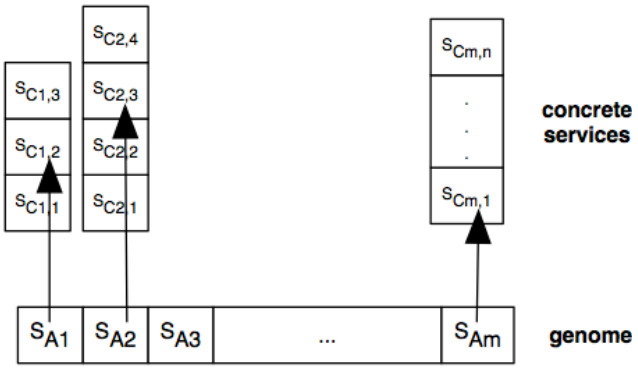

Fig. 2 Genome encoding [2]

Table 1 QoS Aggregation

Control struct. / QoS Property	Flow	Sequence	Switch	While
Response Time (T)	$\max_{i\in 1..m}(t_i)$	$\sum_{i=1}^{m} t_i$	$\sum_{i=1}^{m} p_i \cdot t_i$	$k \cdot t$
Rating (R)	$\prod_{i=1}^{m} r_i$	$\prod_{i=1}^{m} r_i$	$\sum_{i=1}^{m} p_i \cdot r_i$	r^k
Distance (D)	$\sum_{i=1}^{m-1} d_i$	$\sum_{i=1}^{m-1} d_i$	$\sum_{i=1}^{m-1} p_i \cdot d_i$	$k \cdot d$
Cost (C)	$\sum_{i=1}^{m} c_i$	$\sum_{i=1}^{m} c_i$	$\sum_{i=1}^{m} p_i \cdot c_i$	$k \cdot c$

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) similar to the one proposed by Canfora et al. [2]:

$$F(y) = \frac{w_1 \cdot R}{w_2 \cdot T + w_3 \cdot D + w_4 \cdot C} \tag{4}$$

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

V. Numerical experiments and evaluation

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

1. *TruncDE* - the DE algorithm based on Lampinen and Zelinka's proposal [6] with the parameters: scaling factor $F = 0.95$, jitter $F\_NOISE = 0.001$ and crossover constant $Cr = 0.95$. 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.
2. *XueDE* - the DE algorithm proposed by Xue et al. [17] with the following parameters: *DE/best/1/bin* strategy, scaling factor $F = 0.9$, jitter $F\_NOISE = 0.25$ and crossover constant $Cr = 0.95$. The probabilities for the conditional operator in equation (3) are: greedy probability $p_g = 0.1$, mutation probability $p_m = 0.65$ and crossover probability $p_c = 0.2$.
3. *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 $p_m = 0.01$.

For all these algorithms, the population was limited to 100 individuals, which were evolved for 1000 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. Each scenario ran 100 times and the results were averaged. All algorithms were implemented using ***ECJ*** version 20 [8].

Figures 3 – 7 show most significant numerical results for two of the considered test scenarios.

A case with a business workflow consisting in $m=10$ abstract services, each of them having $n=10$ concrete alternative services was evaluated. The results are depicted in Fig. 3. Within the first 80 generations all algorithms find a very good individual. Then, the best fitness of the population increases at a very slow rate. The fastest algorithm for this scenario is *TruncDE.*

A more complex scenario, involving a business workflow consisting in $m=20$ abstract services, each of them having $n=40$ alternatives is presented in Fig. 4. We notice that when increasing the complexity of the problem, *XueDE* becomes the fastest algorithm to converge, while *TruncDE* is the slowest. *IntGA*'s performance is above average, being comparable to the best *DE* in every scenario.

Since our aggregate fitness function *(4)* is composed of several objectives, some requiring to be maximized, others requiring to be minimized, we present the evolution of the objectives represented by the distance, cost and rating for the second scenario in Fig. 5 – 7.

These results show that the proposed DE approach (*TruncDE* and *XueDE*) outperforms the genetic algorithm proposed by Canfora et al. *IntGA* [2]) for solving the NP-hard problem of QoS-based service optimization.

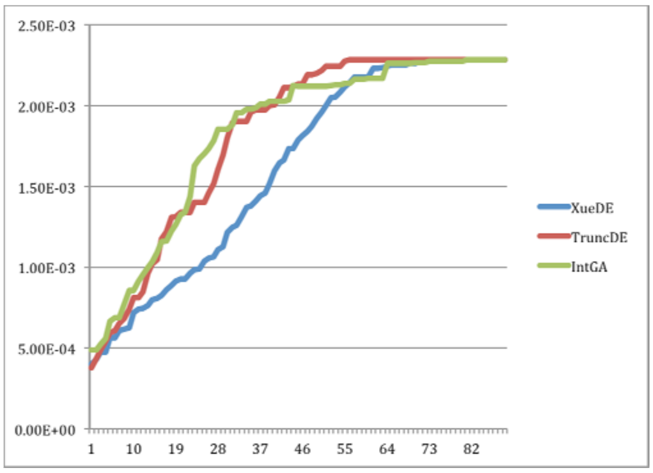

Fig. 3 The evolution of the best fitness over 90 generations for m=10 abstract services and n=10 concrete services

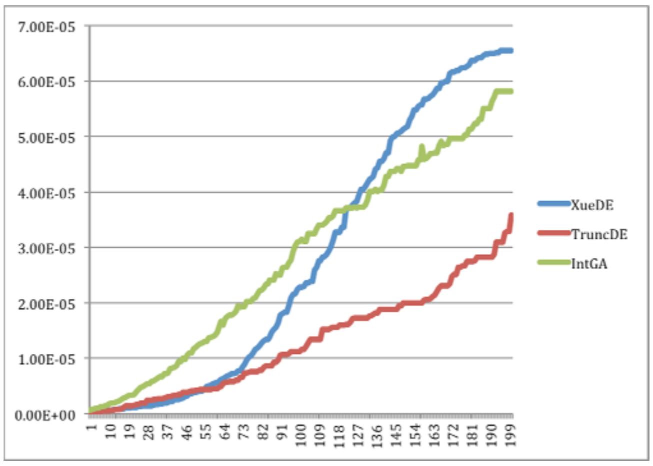

Fig. 4 The evolution of the best fitness over 200 generations for $m=20$ abstract services and $n=40$ concrete services

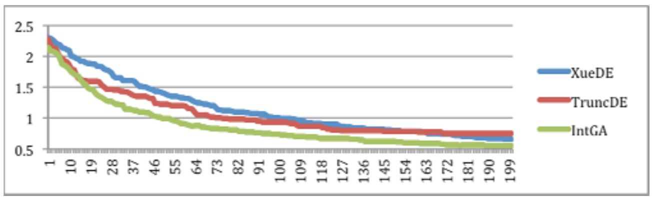

Fig. 5 Distance minimization during 200 generations for $m=20$ abstract services and $n=40$ concrete services

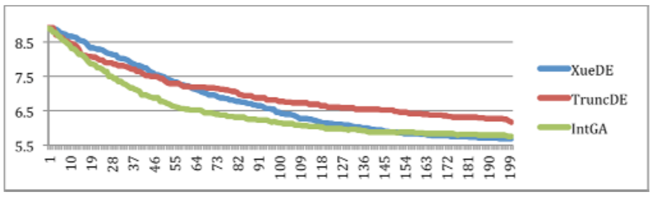

Fig. 6 Cost minimization during 200 generations for *m=20* abstract services and *n=40* concrete services

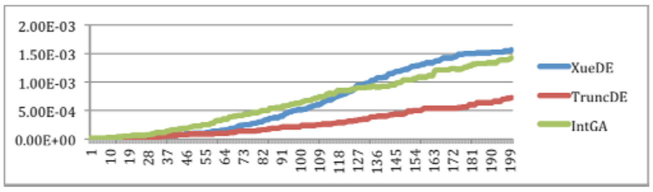

Fig. 7 Rating maximization during 200 generations for *m=20* abstract services and *n=40* concrete services

VI. Conclusion

This paper proposes a method to compose and optimize medical services as business workflows. Such a workflow consists in a set of abstract services, and for each abstract service there are several concrete services. Since each medical service has different QoS parameters such as *response time*, *rating*, *distance* and *cost*, determining the optimal combination of concrete services that realize the abstract services of the business workflow is an NP hard problem.

To solve this problem, we propose a new solution, based on Differential Evolution. We implemented two Discrete DE algorithms from the literature *TruncDE* [6] and *XueDE* [17] which we adapted to solve the services QoS optimization problem. We compare these algorithms with the genetic algorithm proposed by Canfora et al. in [2] – *IntGA*.

The results show that the approach based on DE outperforms the genetic algorithms. *TruncDE* proved to be suited for scenarios of low complexity (up to 15 abstract services, each of them having up to 40 alternatives), while *XueDE* was superior for scenarios of medium and high complexity. The performance of *IntGA* was average, but it was not the slowest to converge in any of the test scenarios.

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.

Acknowledgment

This project was supported by the national project code TE 252 financed by CNCSIS-UEFISCSU.

References

1. 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.
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.
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 - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 11:341-359, 1997.
13. T. Tusar and B. Filipic. 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.
15. T. Weise, S. Bleul, D. Comes, and K. Geihs. Different approaches to semantic web service composition. In
16. 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.
17. 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.
18. L. Alboaie, T. Barbu. Reputation System User Classification Using a Hausdorff-Based Metric. Computational Intelligence for Modelling Control & Automation, 2008 International Conference on, pp. 1035-1040, Dec. 2008