Computing and Informatics, Vol. 33, 2014, 1047–1064

EXPLORING THE SELECTION OF THE OPTIMAL WEB SERVICE COMPOSITION THROUGH ANT COLONY OPTIMIZATION

Viorica Rozina Chifu*, Ioan Salomie*, Cristina Bianca Pop*
Alexandru Nicolae Niculici, Dumitru Samuel Suia

Department of Computer Science
Technical University of Cluj-Napoca
26-28 Baritiu Street, Cluj-Napoca
Romania
e-mail: {`Viorica.Chifu, Ioan.Salomie, Cristina.Pop`}`@cs.utcluj.ro`

Abstract. This paper presents an ant-inspired method for selecting the optimal or a near optimal solution in semantic Web service composition. The proposed method adapts and enhances the Ant Colony Optimization meta-heuristic and considers as selection criteria the QoS attributes of the services involved in the composition as well as the semantic similarity between them. To improve the performance of the proposed selection method a 1-OPT heuristic is defined which expands the search space in a controlled way so as to avoid the stagnation on local optimal solutions. The ant-inspired selection method has been evaluated on a set of scenarios having different complexities and comparatively analyzed with a cuckoo-inspired and a bee-inspired selection method.

Keywords: Ant Colony Optimization, optimal Web service composition, semantic quality, QoS, Web service composition selection

Mathematics Subject Classification 2010: 68T20, 68W25

* corresponding authors

1 INTRODUCTION

Web service composition has emerged as a technique in the context of component-based architectures as it provides the means for satisfying complex service requests by reusing existing atomic/composite services. By composing existing services, in contrast with building new composite services from scratch, the development costs are significantly reduced. Due to the large number of services providing similar functionality that may be involved in Web service composition, the selection of the composition solution that satisfies best the non-functional user requirements may be modeled as an optimization problem. Some of the most suitable search strategies applicable in solving such optimization problems are the bio-inspired meta-heuristics as they identify the optimal or a near-optimal solution in a short time and without processing the entire search space. The bio-inspired meta-heuristics (e.g. Ant Colony Optimization, Particle Swarm Optimization, Bee Colony Optimization) model the collective intelligence of insects, birds and animals that enables them to solve complex life problems (e.g. foraging, migration).

In this paper we explore how the Ant Colony Optimization (ACO) meta-heuristic [3] can be adapted to identify the optimal or a near optimal solution in semantic Web service composition. The methodology used for adapting ACO includes the following steps: modeling the ACO entities, relationships and processes to fit the problem of selecting the optimal composition solution and adapting an algorithm proposed by the ACO meta-heuristic for solving the service selection problem. In addition, for improving the performance of the traditional ACO algorithm we define a 1-OPT heuristic which expands the search space in a controlled way so as to avoid the stagnation on local optimal solutions. By using this methodology we have developed an ant-inspired selection method which is applied on an Enhanced Planning Graph (EPG), dynamically built according to the user request. The EPG extends the classical planning graph [12] structure with the concepts of cluster of services and cluster of service input/output parameters. To identify the optimal or a near optimal solution encoded in the EPG we define a fitness function which uses the QoS attributes and the semantic quality as selection criteria. The ant-inspired selection method has been evaluated on a set of scenarios having different complexities and comparatively analyzed with a cuckoo-inspired and a bee-inspired selection method.

The paper is organized as follows. Section 2 presents related work. Section 3 presents the ant-inspired selection method, while Section 4 evaluates its performance. The paper ends with conclusions.

2 RELATED WORK

This section presents the state of the art in Ant Colony Optimization-based methods for selecting the optimal or a near-optimal solution in Web service composition. Most of these approaches model Web service composition as an abstract graph of tasks, each task having associated a set of concrete services. The services are described by their QoS attributes.

In [9], authors propose an ACO-based method for selecting the optimal Web service composition solution. In each iteration, each ant builds a composition solution starting from the graph origin by probabilistically choosing candidate services to be added to its current partial solution. The probability of choosing a candidate service depends on the pheromone level associated to the edge in the abstract workflow connecting the current service to the candidate service and on heuristic information (the weighted sum of normalized QoS attributes values). The importance of the two components is given by two parameters experimentally tuned according to the specific optimization problem. The pheromone laid by an ant is defined as a k-tuple, where a tuple element represents the pheromone quantity associated to a problem objective such as response time, cost, etc. After an ant completes the construction of a composition solution, the pheromone associated to each edge part of the solution is updated. The update is a two-step process which consists of decreasing the pheromone level associated to each edge by means of an evaporation strategy and of increasing the pheromone level associated to each edge part of a promising solution. The pheromone evaporation strategy is applied only after a predefined number of iterations, to avoid the situation in which the pheromone level increases too much. The pheromone level associated to the edge parts of promising solutions is increased proportionally with the value of a utility function. The function measures how close is the quality of the current solution to the quality of the optimal solution in terms of the QoS attributes.

In [7], the heuristic information used in the ant's probabilistic decision is inversely proportional to the square root of the sum of the considered QoS attributes square values. In this approach, initially, all pheromones have the same constant value which is continuously updated after each ant builds its solution. The update is performed in a single step and depends on a pheromone evaporation rate, the current pheromone level and a chaos operator. The chaos operator aims to improve the convergence speed of the algorithm.

In [10], authors propose an algorithm for selecting the optimal solution in Web service composition which combines Ant Colony Optimization (ACO) and genetic algorithms. ACO is used to find the optimal solution, while the genetic algorithm is used to identify the optimal values of ACO's adjustable parameters. In an iteration, each ant builds a composition solution by choosing probabilistically a new service to add to its partial solution. The choice of a new service depends on the pheromone concentration, on a heuristic factor and on whether the candidate service is tabu or not. In the genetic algorithm, a chromosome represents a set of adjustable parameters combination for ACO. The steps of the genetic algorithm for identifying the optimal configuration of adjustable parameters are as follows:

1. a set of chromosomes is randomly generated,
2. four chromosomes are randomly selected for the next steps,
3. the fitness of the chosen chromosomes is computed as the fitness of the optimal solution obtained by running the ACO-based selection algorithm for each chromosome,

4. the best two chromosomes are selected and crossover and mutation operators are applied on them,
5. the two chromosomes resulted from step 4 replace the worst two chromosomes identified in step 2 [10].

Similar to [10], in the approach presented in [11], each ant builds a composition solution starting from the graph origin by probabilistically choosing candidate services to be added to its partial solution. In this case, the heuristic information used in the probabilistic decision is inversely proportional to the weighted sum of the considered QoS attributes values. Each ant maintains a history with the visited abstract services and the chosen concrete service associated to each visited abstract service. The pheromone laid by an ant on a graph edge is a numerical value reflecting the global QoS value of a service. The pheromone evaporation is performed by decreasing the pheromone level of an edge based on an evaporation rate and on the former pheromone level associated to that edge. After the evaporation is completed, the pheromone level associated to the edges of promising solutions is increased by adding a value which is inversely proportional to the QoS value of the selected service.

3 THE ANT-INSPIRED SELECTION METHOD

This section presents the ant-inspired method for identifying the optimal or a near-optimal composition solution best satisfying a user request in terms of QoS and semantic quality. The ant-inspired method adapts and enhances the Ant Colony Optimization (ACO) meta-heuristic [3]. The search space of the ant-inspired method is modeled as an Enhanced Planning Graph (EPG) we previously introduced in [8]. The EPG is a multi-layered graph in which each layer consists of a set of services organized in service clusters and a set of input/output service parameters organized in parameter clusters. A service/parameter cluster groups similar services/parameters, the similarity being established based on the semantic matching between the service /parameter descriptions.

3.1 Overview of Ant Colony Optimization

ACO is a meta-heuristic which relies on a set of artificial ants which communicate with each other to solve optimization problems. The behavior of artificial ants is modeled according to the behavior of real ants in nature, which search for the shortest route to a food source and communicate indirectly with each other by means of the pheromone they lay on their route. The meta-heuristic consists of three steps which are repeated until a stopping condition is fulfilled (see Figure 1).

Before these steps are iteratively performed, ants are initialized by setting them in a position of the search space. Then, (1) in the first step each artificial ant builds a solution by deciding probabilistically to add a new solution component to its partial solution, (2) in the second step, a local search is performed optionally

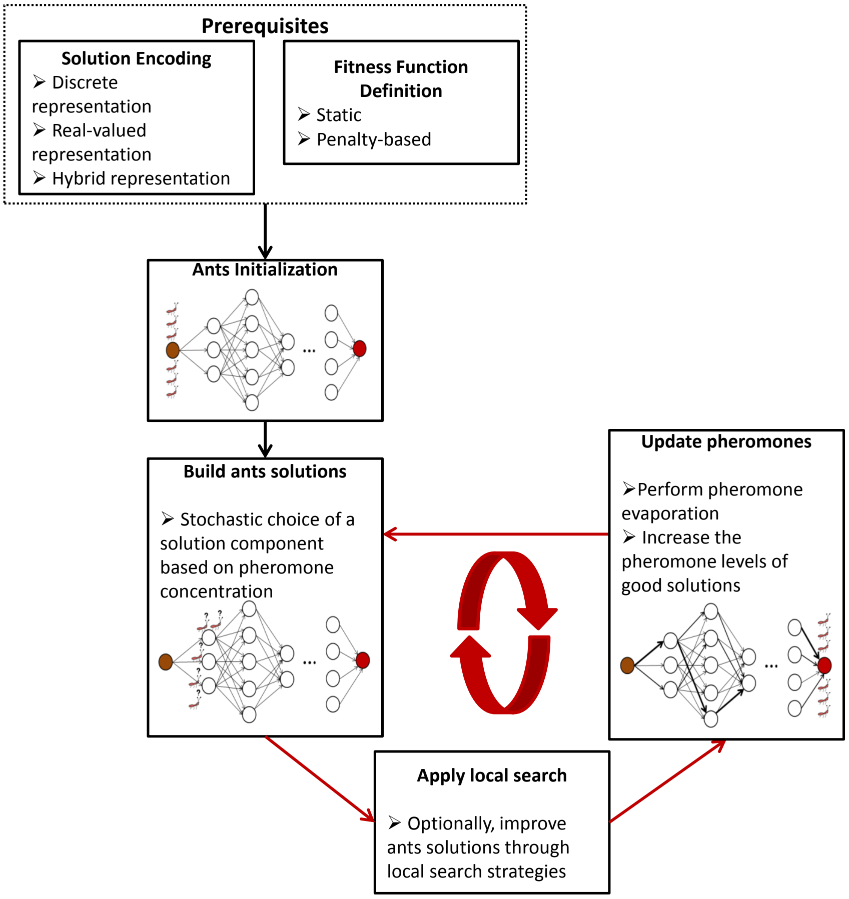
Prerequisites
Solution Encoding
➢ Discrete representation
➢ Real-valued representation
➢ Hybrid representation
Fitness Function Definition
➢ Static
➢ Penalty-based
Ants Initialization
Build ants solutions
➢ Stochastic choice of a solution component based on pheromone concentration
Update pheromones
➢Perform pheromone evaporation
➢ Increase the pheromone levels of good solutions
Apply local search
➢ Optionally, improve ants solutions through local search strategies

Figure 1. The steps of the Ant Colony Optimization meta-heuristic

by each ant aiming to improve their associated solutions, (3) in the third step, the pheromone level of each solution is updated (increased or decreased) based on the quality of each solution which is evaluated with a fitness function [13].

3.2 Mapping the Behavior of Ants to the Problem of Selecting the Optimal Service Composition

The first step in adapting the ACO meta-heuristic to select the optimal Web service composition implies modeling the ACO entities, relationships and processes to fit the considered problem. Consequently, just as in ACO, in the optimal service composition selection problem we have a number of artificial ants that cooperate with each other by indirectly exchanging information (i.e. the pheromone in nature) to identify the optimal or a near-optimal service composition solution (i.e. the food source in nature). In our case, the search space (i.e. the environment in which real ants live) encoding the composition solutions is represented by the EPG.

We formally define an artificial ant as follows:

$$ant = (sol, score) \tag{1}$$

where *sol* is a composition solution associated to the artificial ant, and *score* is the quality of *sol*. A composition solution consists of a set of services such that exactly one service is selected from each cluster of each layer from the EPG. To evaluate the score of a composition solution, we define a fitness function QF which considers the QoS attributes of the associated services as well as the semantic quality of the connections between these services:

$$QF = \frac{w_{QoS} * QoS(sol) + w_{Sem} * Sem(sol)}{w_{QoS} + wSem} \tag{2}$$

where

- $QoS(sol)$ is the QoS score of the composition solution sol [8];
- $Sem(sol)$ is the semantic quality score of the composition solution sol [8];
- w_{QoS} and w_{Sem} are the weights corresponding to user preferences related to the relevance of QoS and semantic quality.

When building a composition solution, each ant is guided by the pheromone-based information associated to each service encoded in the EPG.

3.3 The Ant-Inspired Selection Algorithm

A prerequisite of the ant-inspired selection algorithm is to establish the number of ants that will be used in the search process so as to obtain the optimal solution in a short time interval and without processing the entire search space. We have defined the number of ants as being dependent of the total number of composition solutions encoded in the EPG:

$$noAnts = Round\left(\sqrt[n]{noSol}\right) \tag{3}$$

where

- $noSol$ is the number of possible composition solutions computed by multiplying the numbers of services in each cluster of the EPG;
- $n \in N^*$ is experimentally determined.

In our experiments we have considered several formulas for computing the number of ants and the experimental results demonstrated that the most feasible one is Equation (3). Another formula for computing the number of ants that we have considered applied fixed division factors on the number of solutions. Because the solution domain grows exponentially, scaling the number of ants linearly will make the algorithm suboptimal time-wise independent of the EPG size. This is why we reached the conclusion that the number of ants must be scaled exponentially so we have applied a higher order root on the number of solutions.

The inputs of the ant-inspired selection algorithm (Algorithm 1) are as follows:

1. the EPG structure resulted from the Web service composition process,
2. the weights w_{Sem} and w_{QoS} which state the relevance of a solution's semantic quality compared to its QoS quality, and
3. a number *noAnts* (see Equation (3)) of artificial ants used in the search for the best composition solution.

Algorithm 1 Ant-inspired_Web_Service_Selection

Input: EPG – the enhanced planning graph; w_{QoS}, w_{Sem} – weights for the QoS attributes and the semantic quality; *noAnts* – number of ants;

Output: *Sol* – an ordered set of the best composition solutions found in each algorithm iteration;

```
begin
  Initialize_Pheromone_Levels($EPG, \tau_0$)
  while (!Stopping_Condition()) do
    Reset($Ants, noAnts$)
    $Ants$ = Generate_Solutions($EPG, Ants$)
    $Ants$ = 1-OPT($Ants$)
    $sol_{Best}$ = Get_Best_Solution($Ants, sol_{Best}$)
    $Sol = Sol \bigcup sol_{Best}$
    $Ants$ = Global_Pheromone_Update($Ants$)
  end while
  return $Sol$
end
```

Due to the iterative nature of the selection algorithm, its output consists of an ordered set *Sol* of high quality composition solutions, each added solution being the best one obtained in its iteration. The first position in *Sol* corresponds to the best composition solution obtained so far. Before performing the actual search, the pheromone for each service in the EPG is set to an initial value, τ_0, experimentally determined (*Initialize_Pheromone_Levels*). Then, a number of iterations are performed until a stopping condition is fulfilled (*Stopping_Condition*).

As stopping criterion for the ant-inspired selection method we considered the number *noStagnations* of consecutive iterations in which the algorithm stagnates on the same optimal solution. Based on a set of preliminary experimental results we have defined the following formula for computing the number of stagnations:

$$noStagnations = Round\left(\sqrt[m]{noSol}\right). \tag{4}$$

In each iteration, the artificial ants are repositioned in the origin of the EPG (*Reset*) from where they start building solutions in an incremental way (*Generate_Solutions*).

An ant with a current partial solution sol_p selects its next service s_k from the set of possible services contained in a cluster sc EPG as follows [6]:

$$s_k = \begin{cases} s_i \in sc | \forall s_j \in sc, \tau(s_i) * [QF(sol'_p)]^\beta > \tau(s_j) * [QF(sol''_p)]^\beta, & if\ q \leq q_0 \\ Rand(sc), & \text{otherwise} \end{cases}$$

where

- $\tau(s)$ is the pheromone level of the service s;
- $QF(sol'_p)$ is the quality of the partial solution sol_p to which a service s_i is added;
- β weights the relative importance of the quality value compared to the pheromone level;
- $q \in [0, 1]$ is a random value, while $q_0 \in [0, 1]$ is an experimentally established threshold that influences how the next service is chosen;
- $Rand(sc)$ is a function used to select the next service based on the following probability [6]:

$$P(s) = \frac{\tau(s) * [QF(sol_p^s)]^\beta}{\sum_{s' \in SC} \tau(s') * [QF(sol_p^{s'})]^\beta}. \tag{5}$$

After an ant chooses a service and adds it to its solution, it (locally) updates the pheromone of the service according to the following formula [6]:

$$\tau(s) = (1 - \rho) * \tau(s) + \rho * \tau_0 \tag{6}$$

where

- $\tau(s)$ is the pheromone level of the service s;
- $\rho \in (0, 1]$ represents the pheromone evaporation rate determined experimentally;
- τ_0 represents the initial pheromone level, which is a small constant value used for keeping the pheromone level from dropping too low.

After each ant has associated a new solution, a 1-OPT local search heuristic is applied (*1-Opt*) on each of the solutions to prevent local optimum stagnancy and help a faster convergence to the global optimum. The 1-OPT heuristic is inspired from the general K-OPT graph heuristics which remove k edges from the solution graph and find the best possible replacements for those edges. Our approach takes out one service from each layer of the current solution and finds its best replacement with another service from its cluster based on the fitness function. The next step identifies the global best solution (*Get_Best_Solution*). Finally, the global pheromone update is performed according to Equation (7) (*Global_Pheromone_Update*):

$$\tau(s) = (1 - \alpha) * \tau(s) + \alpha * QF(sol) \tag{7}$$

where

- $\tau(s)$ is the pheromone level of the service s;
- $\alpha \in (0, 1]$ represents the pheromone decay coefficient, that is the rate at which the old pheromone evaporates compared to the addition of the new one;
- $QF(sol)$ is the quality of the solution evaluated with Equation (2).

Initially, we have applied the methods for global pheromone update used in the *Ant System* (AS) [5] and *Ant Colony System* (ACS) [4] algorithms and we have come to the conclusion that they all perform well (AS) or very well (ACS) on small graphs, but on larger ones local stagnancy becomes a major problem. In the case of AS, because the update is done for all solutions in an uncontrolled manner the amount of pheromone deposited on the services belonging to multiple solutions is very high. In time, this causes all the ants to pick the same high pheromone services and stagnate on the same solution composed by them, even if it is a local optimum. For ACS the fact that only one solution is used for the global update leads to very few services taking advantage of this update. The pheromone for these services will then drop very fast in the next iteration, causing the exploitation part of the algorithm to be useless for most of the ants. As a result, most of the work will be done by the exploration function, which, being random, can sometimes fail to find the optimal solution, especially for large graphs. Taking these aspects into consideration we have come up to a compromise between the two methods, which performs much better than both. Our method consists of applying the update rule from ACS by adding pheromone on the services from a number of the best solutions. The amount of pheromone added is directly proportional to the quality of the solution for which the update is made. The number of solutions subject to the pheromone update is a percentage of the number of ants, μ, experimentally determined. This method prevents local stagnancy because the number of services with pheromone updates increases, making exploitation more useful. Also, by using the ACS rule with small values of α, the pheromone levels for services belonging to multiple solutions will increase compared to the ones belonging to only one solution, but in a slower and controlled manner, preventing all services from being overused and diversifying the search. In addition, by applying the update rule for the solutions in order from worst to best will cause the best solution to have the greatest impact on the services belonging to multiple solutions. This causes the services belonging to the best solutions to have the best overall pheromone after the update, thus the algorithm converges to the global optimum.

4 PERFORMANCE EVALUATION

This section presents the methodology for evaluating the ant-inspired selection algorithm as well as an analysis of the experimental results obtained by running the proposed algorithm on different Enhanced Planning Graph topologies.

4.1 Evaluation Methodology

The convergence of an optimization algorithm towards the optimal solution is influenced by a set of adjustable parameters specific to the algorithm. We consider that a proper methodology for evaluating an optimization algorithm should consist of two steps: one step for establishing the optimal values of the adjustable parameters, and another step for evaluating the algorithm using the optimal configuration of the adjustable parameters. To establish the optimal values of the adjustable parameters the following three steps need to be addressed. In the first step, an exhaustive search in the composition model should be performed to identify the score of the optimal composition solution. This score is further used to identify the most appropriate configuration of the adjustable parameters which ensures that the optimal or a near-optimal composition solution is obtained without processing the entire search space. In the second step, an initial configuration of the adjustable parameters should be identified based on a set of preliminary assumptions. These assumptions will be validated by experiments. In the third step, the initial configuration of the adjustable parameters is fine-tuned iteratively to identify their optimal values. During these experiments, the execution time and the standard deviation of the optimal solution returned by the algorithm is compared with the execution time and the optimal solution returned by the exhaustive search. The last two steps should be repeated for composition requests having different complexities and the results should be analyzed to identify the optimal configuration of the adjustable parameters.

4.2 Experimental Results

We have tested our approach on three scenarios with different complexities. In Table I we present the information about the Enhanced Planning Graphs for each considered scenario, where:

1. *code* is the scenario code,
2. *graph configuration* illustrates the number of layers (given by the number of subsets), the number of clusters from each layer (given by the cardinality of each subset) and the number of services per cluster (given by the value of each element in a subset),
3. *noSol* is the total number of possible solutions for the considered scenarion,
4. fit_{Opt} is the optimal fitness value for the considered scenario,
5. *time* is the time measured in minutes and seconds required to identify the optimal solution through exhaustive search.

Tables 2–4 present some fragments (top 31 configurations out of 100 configurations for each scenario) of the best experimental results (average optimal fitness fit_{Opt}, average execution time T_{avg}, average standard deviation stD) obtained when varying the values of the adjustable parameters in the case of each scenario.

Code	*Graph configuration*	*noSol*	fit_{opt}	*Time (min:sec)*
S	{4 5 6} {6 4 6} {4 6 5}	2 073 600	6.456	3:8
M	{3 5 4 6} {6 4 6 5} {4 6}	6 220 800	7.482	15:57
L	{4 4 5 4} {5 4 5 4} {6 5 5}	19 200 000	7.577	56:18

Table 1. The configuration of EPG for each scenario

The algorithm's adjustable parameters are as follows: the number n used for computing the number of ants, the number m used for computing the number of stagnations, the relative importance β of the fitness function compared to the pheromone level, the threshold q_0 that influences how the next service is chosen, the pheromone decay coefficient α, the pheromone evaporation rate ρ, the initial pheromone level τ_0, and the percentage of ants whose solutions are subject to the global pheromone update μ. Each row includes the average experimental results obtained after running the algorithm for 100 times on the same configuration of adjustable parameters.

For the configurations from Table 2, the algorithm provides the optimal solution in most of the cases (the maximum standard deviation is 0.031) in a short time ($time \in [0.3, 1.06]$ seconds) for scenario S. The configuration that provides the best results is highlighted with grey.

For the configurations presented in Table 3, the algorithm provides near-optimal composition solutions in most of the cases (the maximum standard deviation is 0.106) in a short time ($time \in [0.64, 2.8]$ seconds) for scenario M. The configuration that provides the best results (best compromise between fitness and time) is highlighted with grey.

For the configurations presented in Table 4, the algorithm provides near-optimal composition solutions in most of the cases (the maximum standard deviation is 0.065) in a short time ($time \in [0.7, 4.06]$ seconds) for scenario L. The configuration that provides the best results (best compromise between fitness and time) is highlighted with grey.

In conclusion, based on the experimental results we can observe that for scenarios involving a smaller search space (see Table 2), the ant-inspired algorithm identifies the optimal solution in most of the cases in a short time, while for scenarios involving a larger search space (see Tables 3 and 4), the algorithm provides near-optimal solutions in most of the cases in a short time. However, a configuration of adjustable parameters that provides good results, in terms of fitness value and time, for all the considered scenarios is for $n = 4$, $m = 8$, $\beta = 2$, $q_0 = 0.3$, $\alpha = 0.05$, $\rho = 0.1$, $\tau_0 = 0.1$, and $\mu = 25\%$ (see line 13 in Tables 2, 3 and 4).

4.3 Comparative Analysis

To assess the performance of the ant-inspired selection algorithm we have compared it with two bio-inspired selection algorithms: the cuckoo-inspired algorithm [1] and the bee-inspired selection algorithm [2]. The three selection algorithms have been

#	n	m	β	q_0	α	ρ	τ_0	$\mu(\%)$	fit_{avg}	$T_{avg}(s)$	Std
1	4	6	2	0.3	0.05	0.05	0.1	40	6.456	1	0
2	4	6	2	0.3	0.05	0.15	0.1	40	6.456	0.98	0
3	4	6	2	0.3	0.1	0.05	0.1	100	6.456	1.06	0
4	4	6	2	0.3	0.1	0.1	0.1	25	6.456	1	0
5	4	6	2	0.3	0.15	0.05	0.1	1	6.452	1.02	0.004
6	4	6	2	0.3	0.15	0.05	0.1	40	6.456	0.96	0
7	4	6	2	0.3	0.15	0.1	0.1	1	6.456	1.02	0
8	4	7	2	0.3	0.1	0.1	0.1	40	6.456	0.7	0
9	4	7	2	0.3	0.1	0.1	0.1	100	6.455	0.84	0.001
10	4	7	2	0.5	0.1	0.15	0.1	100	6.456	0.66	0
11	4	7	2	0.5	0.15	0.15	0.1	1	6.456	0.64	0
12	4	8	2	0.3	0.05	0.1	0.1	1	6.456	0.66	0
13	4	8	2	0.3	0.05	0.1	0.1	25	6.456	0.62	0
14	4	8	2	0.3	0.1	0.05	0.1	1	6.456	0.64	0
15	4	8	2	0.3	0.1	0.05	0.1	25	6.454	0.62	0.002
16	4	8	2	0.3	0.15	0.1	0.1	100	6.444	0.74	0.012
17	5	6	2	0.3	0.15	0.05	0.1	100	6.456	0.54	0
18	5	6	2	0.3	0.15	0.15	0.1	33	6.454	0.5	0.002
19	5	6	2	0.4	0.1	0.1	0.1	33	6.448	0.48	0.008
20	5	6	2	0.5	0.1	0.15	0.1	33	6.456	0.46	0
21	5	7	2	0.3	0.05	0.15	0.1	25	6.454	0.38	0.002
22	5	7	2	0.3	0.15	0.1	0.1	25	6.450	0.36	0.006
23	5	7	2	0.4	0.1	0.05	0.1	1	6.425	0.36	0.031
24	5	7	2	0.4	0.15	0.15	0.1	40	6.456	0.34	0
25	5	7	2	0.5	0.05	0.1	0.1	33	6.452	0.32	0.004
26	5	7	2	0.5	0.1	0.1	0.1	1	6.441	0.32	0.015
27	5	7	2	0.5	0.15	0.1	0.1	33	6.452	0.32	0.004
28	5	8	2	0.3	0.05	0.1	0.1	100	6.456	0.38	0
29	5	8	2	0.3	0.1	0.1	0.1	100	6.456	0.38	0
30	5	8	2	0.5	0.15	0.05	0.1	33	6.438	0.3	0.018
31	5	8	2	0.5	0.15	0.15	0.1	100	6.456	0.3	0

Table 2. Fragment of the best experimental results for scenario S

comparatively evaluated according to the following criteria: the average number of processed solutions, the average percentage of explored search space, the average simulation time, and the average fitness value. The optimal configurations of adjustable parameters have been considered for each algorithm in the case of each scenario.

As can be seen in the average results presented in Tables 5, 6 and 7, the ant-inspired selection algorithm performs best, followed by the cuckoo-inspired algorithm and the bee-inspired algorithm for scenarios S and M. However, in the case of scenario L, the ant-inspired algorithm is closely outperformed in terms of time and explored search space by the cuckoo-inspired algorithm.

#	n	m	β	q_0	α	ρ	τ_0	$\mu(\%)$	fit_{avg}	$T_{avg}(s)$	Std
1	4	6	2	0.3	0.05	0.05	0.1	40	7.482	2.6	0
2	4	6	2	0.3	0.05	0.15	0.1	40	7.482	2.64	0
3	4	6	2	0.3	0.1	0.05	0.1	100	7.462	2.8	0.020
4	4	6	2	0.3	0.1	0.1	0.1	25	7.482	2.46	0
5	4	6	2	0.3	0.15	0.05	0.1	1	7.460	2.34	0.022
6	4	6	2	0.3	0.15	0.05	0.1	40	7.482	2.54	0
7	4	6	2	0.3	0.15	0.1	0.1	1	7.470	2.3	0.012
8	4	7	2	0.3	0.1	0.1	0.1	40	7.482	2.06	0
9	4	7	2	0.3	0.1	0.1	0.1	100	7.428	2.04	0.054
10	4	7	2	0.5	0.1	0.15	0.1	100	7.449	1.84	0.033
11	4	7	2	0.5	0.15	0.15	0.1	1	7.420	1.52	0.062
12	4	8	2	0.3	0.05	0.1	0.1	1	7.462	1.46	0.020
13	4	8	2	0.3	0.05	0.1	0.1	25	7.482	1.6	0
14	4	8	2	0.3	0.1	0.05	0.1	1	7.454	1.42	0.028
15	4	8	2	0.3	0.1	0.05	0.1	25	7.481	1.58	0.001
16	4	8	2	0.3	0.15	0.1	0.1	100	7.395	1.5	0.087
17	5	6	2	0.3	0.15	0.05	0.1	100	7.465	1.52	0.017
18	5	6	2	0.3	0.15	0.15	0.1	33	7.480	1.04	0.002
19	5	6	2	0.4	0.1	0.1	0.1	33	7.477	1.1	0.005
20	5	6	2	0.5	0.1	0.15	0.1	33	7.479	0.98	0.003
21	5	7	2	0.3	0.05	0.15	0.1	25	7.480	0.86	0.002
22	5	7	2	0.3	0.15	0.1	0.1	25	7.470	0.78	0.012
23	5	7	2	0.4	0.1	0.05	0.1	1	7.376	0.64	0.106
24	5	7	2	0.4	0.15	0.15	0.1	40	7.481	0.86	0.001
25	5	7	2	0.5	0.05	0.1	0.1	33	7.470	0.8	0.012
26	5	7	2	0.5	0.1	0.1	0.1	1	7.398	0.74	0.084
27	5	7	2	0.5	0.15	0.1	0.1	33	7.468	0.8	0.014
28	5	8	2	0.3	0.05	0.1	0.1	100	7.439	0.86	0.043
29	5	8	2	0.3	0.1	0.1	0.1	100	7.428	0.8	0.054
30	5	8	2	0.5	0.15	0.05	0.1	33	7.439	0.7	0.043
31	5	8	2	0.5	0.15	0.15	0.1	100	7.436	0.74	0.046

Table 3. Fragment of the best experimental results for scenario M

The good performance of the ant and cuckoo-inspired selection algorithms is due to the proper balance between exploration and exploitation which ensures that a solution close to the global optimal one is reached in most of the cases. The drawback of the bee-inspired selection algorithm is that it focuses more on exploitation rather than on exploration. As a result, the chances that it stagnates on a local optimum, not close to the global one, increase.

#	n	m	β	q_0	α	ρ	τ_0	$\mu(\%)$	fit_{avg}	$T_{avg}(s)$	Std
1	4	6	2	0.3	0.05	0.05	0.1	40	7.577	3.1	0
2	4	6	2	0.3	0.05	0.15	0.1	40	7.577	3.66	0
3	4	6	2	0.3	0.1	0.05	0.1	100	7.567	4.06	0.010
4	4	6	2	0.3	0.1	0.1	0.1	25	7.577	3.16	0
5	4	6	2	0.3	0.15	0.05	0.1	1	7.560	3.3	0.017
6	4	6	2	0.3	0.15	0.05	0.1	40	7.577	3.3	0
7	4	6	2	0.3	0.15	0.1	0.1	1	7.571	3.44	0.006
8	4	7	2	0.3	0.1	0.1	0.1	40	7.577	2.46	0
9	4	7	2	0.3	0.1	0.1	0.1	100	7.539	2.6	0.038
10	4	7	2	0.5	0.1	0.15	0.1	100	7.565	2.5	0.012
11	4	7	2	0.5	0.15	0.15	0.1	1	7.553	2.22	0.024
12	4	8	2	0.3	0.05	0.1	0.1	1	7.568	1.98	0.009
13	4	8	2	0.3	0.05	0.1	0.1	25	7.577	1.86	0
14	4	8	2	0.3	0.1	0.05	0.1	1	7.561	1.96	0.016
15	4	8	2	0.3	0.1	0.05	0.1	25	7.577	1.7	0
16	4	8	2	0.3	0.15	0.1	0.1	100	7.526	2.14	0.051
17	5	6	2	0.3	0.15	0.05	0.1	100	7.573	1.78	0.004
18	5	6	2	0.3	0.15	0.15	0.1	33	7.576	1.36	0.001
19	5	6	2	0.4	0.1	0.1	0.1	33	7.572	1.28	0.005
20	5	6	2	0.5	0.1	0.15	0.1	33	7.575	1.22	0.002
21	5	7	2	0.3	0.05	0.15	0.1	25	7.572	0.98	0.005
22	5	7	2	0.3	0.15	0.1	0.1	25	7.561	0.9	0.016
23	5	7	2	0.4	0.1	0.05	0.1	1	7.512	0.94	0.065
24	5	7	2	0.4	0.15	0.15	0.1	40	7.576	0.92	0.001
25	5	7	2	0.5	0.05	0.1	0.1	33	7.567	0.86	0.010
26	5	7	2	0.5	0.1	0.1	0.1	1	7.541	0.88	0.036
27	5	7	2	0.5	0.15	0.1	0.1	33	7.575	0.86	0.002
28	5	8	2	0.3	0.05	0.1	0.1	100	7.560	1.04	0.017
29	5	8	2	0.3	0.1	0.1	0.1	100	7.549	1.08	0.028
30	5	8	2	0.5	0.15	0.05	0.1	33	7.563	0.7	0.014
31	5	8	2	0.5	0.15	0.15	0.1	100	7.562	0.98	0.015

Table 4. Fragment of the best experimental results for scenario L

5 CONCLUSIONS

In this paper we have proposed a method for identifying the optimal solution in semantic Web service composition, inspired by the foraging behavior of ants. The proposed method optimizes the selection process without considering the entire search space and avoids the local optimum stagnancy problem. The search space is encoded as an Enhanced Planning Graph which is dynamically built for each user request. We have tested and evaluated our ant-inspired selection method on a set of scenarios involving Enhanced Planning Graphs with different complexities. First, we performed a series of experiments to adjust the parameters' values of the selection

Algorithm	*Explored search space (%)*	*Time (sec)*	*Fitness*	*Standard deviation*
Ant	0.008	0.3	6.456	0
Bee	0.0604	2.24	6.455	0.001
Cuckoo	0.129	3.9	6.455	0.001

Table 5. Comparison between Ant, Bee and Cuckoo inspired algorithms for scenario S

Algorithm	*Explored search space (%)*	*Time (sec)*	*Fitness*	*Standard deviation*
Ant	0.012	1.6	7.482	0
Bee	0.0304	4.26	7.477	0.005
Cuckoo	0.059	6.6	7.481	0.001

Table 6. Comparison between Ant, Bee and Cuckoo inspired algorithms for scenario M

Algorithm	*Explored search space (%)*	*Time (sec)*	*Fitness*	*Standard deviation*
Ant	0.004	1.7	7.577	0
Bee	0.0178	7.56	7.572	0.005
Cuckoo	0.003	1.25	7.577	0

Table 7. Comparison between Ant, Bee and Cuckoo inspired algorithms for scenario L

method so that the algorithm provides the optimal or a near-optimal composition solution in a small number of iterations and without processing the entire search space. Then, using the optimal values of the adjustable parameters we conducted a series of experiments to comparatively analyze the performance of our method with a cuckoo-inspired method and a bee-inspired method. Based on the obtained results, we conclude that the ant-inspired method performs best, very closely followed by the cuckoo-inspired method.

REFERENCES

[1] Chifu, V. R.—Pop, C. B.—Salomie, I.—Suia, D. S.—Niculici, A. N.: Optimizing the Semantic Web Service Composition Process Using Cuckoo Search. In: Brazier, F. M. T., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C. (Eds.): Intelligent Distributed Computing V, Proceedings of the 5th International Symposium on Intelligent Distributed Computing (IDC 2011), Delft, The Netherlands, Studies in Computational Intelligence, Vol. 382, 2011, pp. 93–102.

[2] Chifu, V. R.—Pop, C. B.—Salomie, I. et al.: Selecting the Optimal Web Service Composition Based on a Multi-Criteria Bee-Inspired Method. In: Proceedings of the 12th International Conference on Information Integration and Web-Based Applications & Services (iiWAS 2010), Paris, France, November 2010, pp. 40–47.

[3] Dorigo, M.—Di Caro, G.: The Ant Colony Optimization Metaheuristic. In: Corne, D. et al. (Eds.): New Ideas in Optimization. McGraw-Hill, UK, 1999, pp. 11–32.

[4] Dorigo, M.—Gambardella, L. M.: Ant Colonies for the Traveling Salesman Problem. Bio-Systems, Elsevier, 1997, pp. 4373–4381.

[5] Dorigo, M.—Maniezzo, V.—Colorni, A.: Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B, Vol. 26, 1996, No. 1, pp. 29–41.

[6] Gambardella, L. M.—Dorigo, M.: Solving Symmetric and Asymmetric TSPs by Ant Colonies. Proceedings of the International Conference on Evolutionary Computation, Japan, 1996, pp. 622–627.

[7] Li, W.—Yan-Xiang, H.: A Web Service Composition Algorithm Based on Global QoS Optimizing with MOCACO. In: Hsu, C. H., Yang, L. T., Park, J. H., Yeo, S. S. (Eds.): Algorithms and Architectures for Parallel Processing, 10th International Conference (ICA3PP 2010), Busan, Korea, May 2010, pp. 218–224.

[8] Pop, C. B.—Chifu V. R.—Salomie, I.—Dinsoreanu, M.: Immune-Inspired Method for Selecting the Optimal Solution in Web Service Composition. In: Lacroix, Z. (Ed.): Resource Discovery, Second International Workshop (RED 2009), Lyon, France, LNCS, Vol. 6162, 2010, pp. 1–17.

[9] Zhang, W.—Chang, C. K.—Feng, T.—Jiang, H.: QoS-Based Dynamic Web Service Composition with Ant Colony Optimization. In: Ahamed, S. I. et al. (Eds.): Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference (COMPSAC 2010), Seoul, Korea, July 2010, pp. 493–502.

[10] Yang, Z.—Shang, C.—Liu, Q.—Zhao, C.: A Dynamic Web Services Composition Algorithm Based on the Combination of Ant Colony Algorithm and Genetic Algorithm. Journal of Computational Information Systems, Vol. 6, 2010, No. 8, pp. 2617–2622.

[11] Wang, X. L.—Jing, Z.—Yang, H. Z.: Service Selection Constraint Model and Optimization Algorithm for Web Service Composition. Journal of Information Technology, Vol. 10, 2011, pp. 1024–1030.

[12] Russell, S.—Norvig, P.: Artificial Intelligence: A Modern Approach. Upper Saddle River, NJ, Prentice Hall/Pearson Education, 2003.

[13] Dorigo, M.—Birattari, M.—Stützle, T.: Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique. IEEE Computational Intelligence Magazine, Vol. 1, 2006, Iss. 4, pp. 28–39.

Viorica Rozina Chifu is a Ph. D. lecturer at the Department of Computer Science, Technical University of Cluj-Napoca, Romania. Her research areas of interest include ontologies and semantic web, AI planning, automatic web service composition.

Ioan Salomie is currently Professor of computer science at the Technical University of Cluj-Napoca, Romania, being in the past years invited Professor at Loyola College in Maryland (1996) and University of Limerick (2000–2004). His research interests focuse on service oriented distributed computing, context awareness and autonomic computing, bio-inspired computing and intelligent systems. He is the Head of Distributed Systems Research Laboratory, which is an active partner in relevant national and EU projects.

Cristina Bianca Pop is a Ph. D. student at the Department of Computer Science, Technical University of Cluj-Napoca, Romania. Her research areas of interest include biologically-inspired distributed systems, ontologies and semantic web, automatic web service composition.

Alexandru Nicolae Niculici is Master student at the Department of Computer Science, Technical University of Cluj-Napoca, Romania.

Dumitru Samuel Suia is Master student at the Department of Computer Science, Technical University of Cluj-Napoca, Romania.