A Bee-inspired Approach for Selecting the Optimal Service Composition Solution

Cristina Bianca POP, Viorica Rozina CHIFU, Ioan SALOMIE, Mihaela DINSOREANU, Mihaly FODOR, Irina CONDOR
Department of Computer Science, Technical University of Cluj-Napoca
26-28 Baritiu Street, Cluj-Napoca, Romania
{Cristina.Pop, Viorica.Chifu, Ioan.Salomie, Mihaela.Dinsoreanu}@cs.utcluj.ro,
{Mihaly Fodor, Irina.Condor}@student.utcluj.ro

Abstract — **This paper presents a bee colony optimization method for selecting the optimal solution in semantic Web service composition. The bee-inspired selection method uses an enhanced planning graph model and a matrix of semantic links to incrementally search the optimal solution. We use a multi-criteria function which evaluates whether a solution is optimal or not in terms of its QoS attributes and the quality of the semantic match between the services involved in the solution. The selection method was validated by making experiments on a set of semantic Web services from the trip planning domain.**

Index Terms — **bee colony optimization, enhanced planning graph, ontology, semantic Web service, service composition**

I. INTRODUCTION

Web Services have grown in importance for the past years because they provide business functionality to other applications using Internet connections in a modular and self-contained way. In the real world, a single Web-service is rarely used as the answer to a complex request and so, more services need to work together in order to accomplish the task. Manually discovering and composing Web services is a tedious work, if the growing number of available Web services and their diversity in terms of functionality or Quality of Service (QoS) is taken into account. Web Services can be described at a syntactic and at a semantic level. In the case of the syntactic description, the WSDL (interface-based description of a Web service) and SOAP (XML-based formalization of Web services inter-communication) standards are used. In the semantic Web view, Web services are described using ontology concepts which annotate the service inputs, outputs, preconditions and effects. As a result, computers can understand not only the syntax of Web services, but also their content and meaning thus favoring an automatic process of service discovery and composition. In case the number of services involved in composition is large, a lot of composition solutions may be obtained and the search for the optimal solution can be seen as an optimization process.

This paper presents a new technique for automatic Web service composition inspired by the behavior of bees. The proposed technique uses an enhanced AI planning graph model combined with a bee-inspired optimization algorithm to find the composition solution which satisfies the user request. The user request is described in terms of functional and non-functional requirements. The functional requirements are expressed using ontological concepts that annotate the provided inputs and requested outputs. The non-functional requirements represent weights associated to user preferences regarding the relevance of a solution's semantic quality compared to its QoS attributes. The bee-inspired optimization algorithm is used to select the optimal composition solution according to QoS attributes and semantic quality.

The paper is organized as follows. Section II presents the proposed Web service composition method, while section III details the bee-inspired selection method for identifying the optimal composition solution. Experimental results are presented in section IV. Section V surveys the related work of bio-inspired service composition methods. We end our paper with conclusions and future work proposals.

II. THE WEB SERVICE COMPOSITION METHOD

To obtain the optimal Web service composition solution we combine a bee-inspired selection method with an *Enhanced Planning Graph* (*EPG*) model and a *Matrix of Semantic Links* (*MSL*). In this section we briefly present the composition approach that we have proposed in [6] which aims at building the *EPG* and *MSL* models.

A. The Enhanced Planning Graph Model

We obtained the *EPG* model by mapping the classical AI planning graph problem [7] to semantic Web service composition (see Table 1) and by adding new structures.

TABLE I. CONCEPTS FROM MAPPING THE AI PLANNING GRAPH PROBLEM TO SEMANTIC WEB SERVICE COMPOSITION

AI planning graph concepts	**Web service composition concepts**
Action	Service operation described by an ontology concept
Precondition	Input parameter of a service operation described by an ontology concept
Effect	Output parameter of a service operation described by an ontology concept
Initial state	User provided input parameters expressed as ontology concepts
Goal state	User provided output parameters expressed as ontology concepts

The *EPG* construction is an iterative process. In each iteration, a new layer consisting of a tuple (A_i, L_i) is added to the graph where A_i represents a set of service clusters and L_i is a set of clusters of service output parameters. Layer 0 consists of a tuple (A_0, L_0) where A_0 is an empty set of services (actions) and L_0 contains the input parameters of the user request. For each layer $i > 0$, A_i consists of a set of clusters of services for which the input parameters are literals from L_{i-1}. A cluster of services groups services which provide the same functionality. The functionalities of the services belonging to the same cluster are annotated only with *is-a* related ontological concepts. The services which contribute in each step to the extension of the *EPG* are provided by a discovery process. The discovery process finds the appropriate Web services in a repository of services, based on the semantic matching between the services' inputs and the set of literals of the previous graph layer. The L_i set is constructed as a union of the L_{i-1} set and the sets of outputs of the services in A_i. Parameters of L_i (inputs and outputs) are grouped in clusters of literals where each cluster contains only *is-a* related ontological concepts. The construction of the *EPG* ends either when the user requested outputs are contained in the current set of literals or when the graph reaches a fixed point. Reaching a fixed point means that the sets of actions and literals are the same for the last two consecutive generated layers. In figure 1 we present the step-by-step construction of an *EPG*.

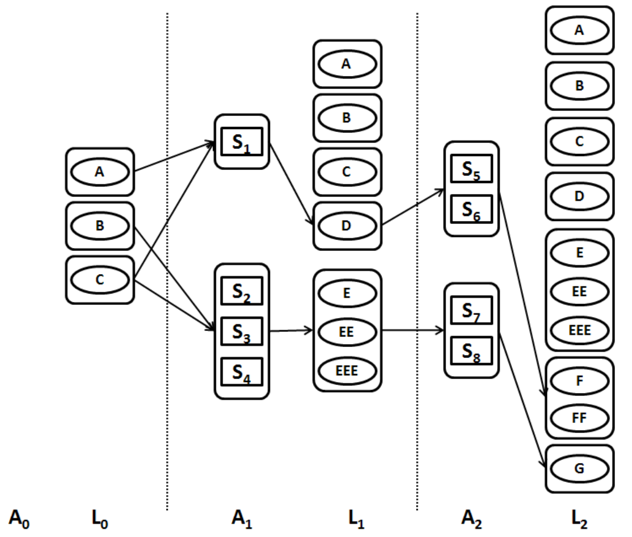

Figure 1. An example of step-by-step construction of the *EPG*.

In the figure, a service is represented with a rectangle and a literal with an oval. The services as well as the literals are grouped in clusters. The clusters are represented as rectangles with rounded corners. An example of a cluster of literals would be {*E*, *EE*, *EEE*} in L_1 which is linked to the second cluster of A_1 services from the first layer. The link indicates that the s_2, s_3 and s_4 services in this cluster, have *E*, *EE* and, *EEE* as outputs. A composition solution for the *EPG* of figure 1 consists of the following subsets of services: {[s_1 s_2], [s_6, s_7]}. In each solution subset, only one service of its cluster can be considered.

B. The Matrix of Semantic Links

The *Matrix of Semantic Links* (*MSL*) is built in parallel with the construction of the *EPG*. The matrix stores the semantic links established between services on different layers in the *EPG*. One Web service could be linked to several other Web services. We say that there is a *semantic link* between two services s_1 and s_2 if there is a *Degree of Match* (DoM) [8] between the set of the output parameters of service s_1 and the set of the input parameters of service s_2. When computing the DoM between an output concept of a service and an input concept of another service we consider three types of semantic matching: EXACT, PLUG IN, and SUBSUME [8].

In *MSL*, both the columns and the rows are labeled with services from the *EPG*. A column and a row having the same index will be labeled with the same service. *MSL* is formally defined as:

$$MSL = [msl_{ij}]_{i=1\ldots n, j=1\ldots m} \tag{1}$$

where an element msl_{ij} of *MSL* is represented as follows:

$$msl_{ij} = \begin{cases} \phi, & if\ simS(s_i.out_k, s_j.in_l) = 0 \\ sl_{ij}, & otherwise \end{cases} \tag{2}$$

In (2), sl_{ij} represents the semantic similarity link between the service on row *i* and the service on column *j*. A semantic similarity link is formally defined as a tuple

$$sl_{ij} = (V, simS) \tag{3}$$

where

- *V* is a set of pairs ($s_i.out_k$, $s_j.in_l$) where $s_i.out_k$ is an output parameter of service s_i, $s_j.in_l$ is an input parameter of service s_j, and the degree of match between $s_i.out_k$ and $s_j.in_l$ is greater than 0.
- *simS* is the semantic similarity score computed between the subset of output parameters of service s_i and the subset of input parameters of service s_j.

We compute simS between the subset outk of output parameters of service si and the subset ink of input parameters of service sj with the following formula:

$$simS(s_i.out_{k_t}, s_j.in_{k_t}) = \frac{\sum_{t=1}^{m} F\_Measure(s_i.out_{k_t}, s_j.in_{k_t})}{m} \tag{4}$$

F_Measure is an adapted version of the similarity measure from information retrieval [12] and is computed as:

$$F\_Measure(s_i.out_{k_t}, s_j.in_{k_t}) = \frac{2 * prc_g(s_i.out_{k_t}, s_j.in_{k_t}) * rec_g(s_i.out_{k_t}, s_j.in_{k_t})}{prc_g(s_i.out_{k_t}, s_j.in_{k_t}) + rec_g(s_i.out_{k_t}, s_j.in_{k_t})} \tag{5}$$

where

- prc_g is the precision between an output parameter of service s_i and an input parameter of service s_j and is computed with the formula:

$$prc_g(s_i.out_{k_t}, s_j.in_{k_t}) = \sqrt{prc_I(s_i.out_{k_t}, s_j.in_{k_t}) * prc_{II}(s_i.out_{k_t}, s_j.in_{k_t})} \tag{6}$$

- rec_g is the recall between an output parameter of service s_i and an input parameter of service s_j and is computed with the formula:

$$rec_g(s_i.out_{k_t}, s_j.in_{k_t}) = \sqrt{rec_I(s_i.out_{k_t}, s_j.in_{k_t}) * rec_{II}(s_i.out_{k_t}, s_j.in_{k_t})} \tag{7}$$

In formulas (6) and (7), prc_I and rec_I represent the precision and recall between an output of a service and an input of another service, while prc_{II} and rec_{II} represent the global precision and recall between all the properties of a service output and of another service input. The precision and recall have been evaluated according to the formulas presented in [12].

In the case of the *EPG* in figure 1, a semantic similarity link between services s_2 and s_7 is established because there is a degree of match greater than 0 between the output of service s_2 and an input of service s_7.

III. The Bee-Inspired Selection Method

For finding the optimal composition we adapted and enhanced the Bee Colony Optimization Metaheuristic defined in [9]. As any metaheuristic, the one proposed in [9] defines a general-purpose algorithmic framework that can be adapted to solve different optimization problems [10]. Our bee-inspired selection method uses the *EPG* and *MSL* generated by the composition method (see section II) as well as a multi-criteria function (fitness function) to find the optimal composition solution. The fitness function considers the QoS user preferences and semantic quality in the evaluation of a composition solution.

A. The Foraging Behavior of Bees

Bees represent complex collective systems in which individuals follow simple rules to build hives and search for food. In the case of nectar gathering, scout bees are sent to find the most satisfying food sources. They randomly fly on varying distances in search of food. Upon returning to the hive, they have three possibilities: to abandon their food source because it was not good enough, to continue to collect nectar from the current discovered source or to communicate to the other bees what they have found so far and thus recruiting followers [9]. This type of communication is achieved through a waggle dance. Each returning bee performs this dance several times in front of the other bees in the hive. The dance encodes information about the distance, the direction and the quality of the food source. The bees in the hive can decide to follow one of the waggle dancing bees. Through this mechanism, the flower patches closer to the nest and with more quality nectar are visited by more bees enabling an efficient pollen harvesting.

B. Mapping the Foraging Behavior of Bees to the Web Service Composition Problem

We mapped the concepts from the foraging behavior of bees to the concepts of the Web service composition problem as illustrated in table II.

TABLE II. CONCEPTS FROM MAPPING THE FORAGING BEHAVIOR OF BEES TO THE WEB SERVICE COMPOSITION PROBLEM

Concepts from the foraging behavior of bees	**Web service composition concepts**
Bee	Artifical bee
Hive	The layer (A_0, L_0) from the *EPG*
Food source	A Web service composition solution
Quality of the food source	The QF function (see formula 9)
Waggle dance	Information about the services and quality of a composition solution
Scout bee	Artificial bee that continues expanding its Web service composition solution
Follower bee	Artificial bee which abandons its service composition solution and adopts the solution of a scout bee

In the Web service composition problem, we consider that the biological bee becomes an artificial bee [9] which explores the *EPG* in search of the optimal composition solution (the best quality food source). We define an artificial bee as follows:

$$bee = (bSol, loyalty) \tag{8}$$

where

- *bSol* is a set of *n* elements $bSolElem_i$ where a $bSolElem_i$ is defined as a set of services $\{s^i_{kl} \mid s^i_{kl}$ is the *l*-th service from cluster *k* on layer *i* of the *EPG*$\}$ and *n* is the total number of layers of the *EPG*;
- *loyalty* is the quality of the *bSol* (similar to the quality of a discovered food source) in terms of QoS attributes and semantic quality.

To evaluate the loyalty we define and use the QF function below:

$$QF(bSol) = \frac{w_{QoS} * QoS(bSol) + w_{Sem} * Sem(bSol)}{w_{QoS} + w_{Sem}} \tag{9}$$

where

- *QoS(bSol)* is the QoS score of the Web service composition solution, *bSol*.
- *Sem(bSol)* is the semantic quality score of the Web service composition solution *bSol*.
- *wQoS* and *wSem* are the weights which illustrate the user preference related to the relevance of QoS and semantic quality during the evaluation of a composition solution.

The QoS score of a composition solution *bSol* is computed using formula 10:

$$QoS(bSol) = \frac{\sum_{i=1}^{n} w_i * qos_i(bSol)}{\sum_{i=1}^{n} w_i} \tag{10}$$

where

- $qos_i(bSol)$ represents the value of a QoS attribute computed for the composition solution *bSol.*
- w_i is the weight associated to the relevance of the qos_i attribute.
- *n* is the total number of QoS attributes considered.

The semantic quality score of a composition solution *bSol* is computed as follows:

$$Sem(bSol) = \frac{\sum_{i=1}^{n} simS(s^j_{kl}.out, s^p_{qr}.in)}{n-1} \tag{11}$$

where

- s^j_{kl} is the service *l* in cluster *k* from layer *j*;
- s^p_{qr} is the service *r* in cluster *q* from layer *p*;
- s^j_{kl}, s^p_{qr} are part of the solution *bSol*, $j < p$;
- *n* is the total number of services involved in the solution *bSol*;
- *simS* computes the degree of match between s^j_{kl} and s^p_{qr} (see formula 4).

In Web service composition, scout bees are mapped to artificial bees that have identified a good solution and that will continue expanding it. On the other hand, follower bees are mapped to artificial non-loyal bees which have identified a poor solution and will adopt the solutions of waggle bees which they will expand.

C. The BEE-INSPIRED SELECTION ALGORITHM

The bee-inspired selection algorithm (**ALGORITHM _1**) determines a set *SOL* of high quality composition solutions evaluated according to the function *QF* (see formula 9), the first solution being the optimal one, by considering the following: (*i*) the *EPG* and *MSL* structures resulted from Web service composition, (*ii*) the weights w_{QoS} and w_{Sem} which state the relevance of a solution's QoS quality compared to its semantic quality, and (*iii*) a number *nBees* of artificial bees used in the search for the optimal solution.

```
----------------------------------------------
ALGORITHM _1: BEE-INSPIRED-WEB-SERVICE-SELECTION
----------------------------------------------
Input: EPG - the enhanced planning graph;
       MSL - the matrix of semantic links;
       wQoS - the weights for the QoS attributes;
       wSem - the weight for the semantic quality;
       nBees - the number of bees;
Output: SOL
Comments: SOL = {sol_1,...sol_n | sol_i = {s^j_kl | s^j_kl is
the l-th service from cluster k on layer j of the
EPG} and n is the number of layers of the EPG};
BEES - the set of bees defined as {bee_1, ...,
bee_nBees},for the definition of bee_i see formula 8;
sol_rec = the recommended solution in each iteration
of the algorithm which is defined similar to sol_i;
begin
  sol_rec = GET-INITIAL-SOLUTION(EPG)
  SOL = {sol_rec}
  while ( !SOLUTION-GOOD-ENOUGH(sol_rec) ) do
    begin
      RESET (BEES, nBees)
      for i = 0 to GET-NR-OF-LAYERS(EPG) do
        begin
          BEES = FORWARD-PASS(EPG, i, BEES, sol_rec)
          BEES = BACWARD-PASS (BEES)
        end
      sol_rec = GET-BEST-SOLUTION(BEES, sol_rec)

      SOL = SOL ∪ sol_rec

    end
  return SOL
end
----------------------------------------------
```

Before performing the actual search, an initial solution is identified using a greedy approach (GET-INITIAL-SOLUTION). The selection algorithm consists of a number of iterations which are performed until a stopping condition is fulfilled (SOLUTION-GOOD-ENOUGH). We consider as stopping condition the case when the quality of the best solution obtained so far is above a pre-established threshold.

In each iteration, the artificial bees are repositioned in the origin of the *EPG* (RESET) from where they start building solutions in an incremental way. From an artificial bee's point of view, an iteration is composed of a number of stages equal to the number of layers (GET-NR-OF-LAYERS) in the *EPG*. Within each stage, the artificial bee performs a forward (FORWARD-PASS) and a backward pass (BACKWARD-PASS) aiming at identifying new candidate composition solutions. At the end of an iteration each artificial bee obtains a solution. The set of solutions obtained within an iteration is evaluated according to the *QF* function (see formula 9). Then, the best solution identified (GET-BEST-SOLUTION) becomes the recommended one which will guide artificial bees in taking decisions when exploring the *EPG* in the next iteration. During the search process, the artificial bees communicate directly by exchanging information about the quality of the partial solutions they have built so far.

Within the forward pass (**ALGORITHM _2**), each bee selects a new service (SELECT-NEXT-SERVICE) from each cluster of the currently explored *EPG* layer and adds the selected services to the candidate solution.

```
----------------------------------------------
ALGORITHM _2: FORWARD-PASS
----------------------------------------------
Input: EPG - the enhanced planning graph;
       lIndex - an EPG layer index;
       BEES - the set of bees;
       sol_rec - the recommended solution;
Output: BEES - updated set of bees
Comments: BEES - the set of bees defined as {bee_1,
..., bee_nBees} (see formula 8);
GET-NR-CLUSTERS - returns the numbers of clusters
from a specified layer in the EPG;
begin
  for i = 0 to GET-NR-CLUSTERS(EPG, lIndex) do
    begin
      for j = 0 to BEES.size do
        begin
          BEES[j] = SELECT-NEXT-SERVICE(BEES[j], sol_rec,
                                        EPG,lIndex,i)
        end
    end
  return BEES
end
```

The selection of a new service (see **ALGORITHM _3**) is influenced by the recommended service defined by the best solution (sol_{rec}) encountered until that moment, and the location of the currently processed cluster in the *EPG*.

```
----------------------------------------------
ALGORITHM _3: SELECT-NEXT-SERVICE
----------------------------------------------
Input: bee - the bee for which we select the next
             service, represented as in formula 8;
       sol_rec - the recommended solution;
       EPG - the enhanced planning graph;
       lIndex - layer index;
       cIndex - cluster index;
Output: bee - the bee with the updated solution
Comments: nServ - the set of candidate next

services defined as a set {(s, quality) | s ∈

EPG};
prob - set of probabilities associated to each
element of nServ;
begin
  nServ = GET-SERVICES(EPG, lIndex, cIndex)
  sum = 0
  for i = 0 to nServ.size do
    begin
      prob[i]=nServ[i].quality=PQ(nServ[i],bee.bSol)
    end
  COMPUTE-PROB(prob)
  SORT-BY-PROBABILITY(nServ, prob)
  destiny = RANDOM()
  for i = 0 to nServ.size do
    begin
      sum = sum + nServ[i].quality
      if (destiny < sum) then
        begin
          ADD-SERVICE(bee, nServ[i])
        end
    end
  return bee
end
```

As a result, an artificial bee chooses to add a service to its partial solution with a probability *P* (calculated by

COMPUTE-PROB) formally defined as:

$$P(cServ_i, sol_{part}) = \alpha * \frac{PQ(cServ_i, sol_{part})}{\sum_{k=1}^{n} PQ(cServ_k, sol_{part})} \quad (12)$$

where

- $cServ_i$ is a candidate Web service which can be added to the partial solution sol_{part}.
- $$\alpha = \begin{cases} 1, & if\ cServ_i \notin sol_{rec} \\ 2, & otherwise \end{cases} \quad (13)$$
- n is the number of elements in $cServ$.
- PQ is a function that evaluates the quality of a service in terms of semantic quality and QoS attributes related to the services in sol_{part} according to the formula below:

$$PQ(cServ_i, sol_{part}) = QF(sol_{part}') \quad (14)$$

where $sol_{part}' = sol_{part} \cup cServ_i$.

The candidate services are further sorted according to their probability to be chosen (SORT-BY-PROBABILITY). The service that will be added to the partial solution of the considered bee is selected according to the service probability and a *destiny* value. The destiny value is introduced to avoid the algorithm stagnancy in a local optimum and to add diversity (ADD-SERVICE).

Within the backward pass (**ALGORITHM _4**), all artificial bees return in the origin of the *EPG* where they have to make two decisions: abandon the current partial solution or expand the partial solution without recruiting new artificial bees.

```
---------------------------------------------
ALGORITHM _4: BACWARD-PASS
---------------------------------------------
Input: EPG; BEES;
Output: BEES - the updated set of bees;
Comments: wBees, uBees - the set of waggle/unloyal
bees defined as in formula 8;
COMPUTE-MAX-LOYALTY - computes the maximum loyalty
(maxLoyalty) among bees
lSum - the sum of loyalties of all bees in BEES;
begin
  wBees = uBees = Ø
  maxLoyalty = COMPUTE-MAX-LOYALTY(BEES)
  for j = 0 to BEES.size do
   begin
    BEES[j].loyalty = BEES[j].loyalty / maxLoyalty
    if (BEES[j].loyalty > Δ1) then
     begin
      wBees = wBees ∪ BEES[j]
     end
    if (BEES[j].loyalty < Δ2) then
     begin
      uBees = uBees ∪ BEES[j]
     end
   end
  lSum = 0
  for j = 0 to wBees.size do
   begin
    lSum = lSum + wBees[j].loyalty
   end
  for j = 0 to wBees.size do
   begin
    wBees[j].loyalty = wBees[j].loyalty / lSum
    while (wBees[j].loyalty * uBees.size > 0) do
     begin
      uBee = GET-FIRST(uBees; uBees = uBees - uBee
      uBee.bSol = wBees[j].bSol
     end
   end
  return BEES
end
```

The first decision is influenced by the loyalty of an artificial bee towards the solution (the loyalty representing the quality of the partial solution evaluated with the *QF* function). By interpreting the loyalty according to some pre-established thresholds (Δ_1 and Δ_2), the bee will either expand its solution (the bee becomes waggle) or abandon it (the bee becomes a follower). The second decision which determines whether the partial solution will be further expanded without recruiting new hive mates is also based on the loyalty interpretation. We modeled the recruiting activity as a simple even distribution of non-loyal bees between the waggle ones.

IV. EXPERIMENTAL RESULTS

We have experimented and evaluated our bee-inspired selection method on a scenario from the trip planning domain. We developed and used in our experiments a set of 110 semantic Web services. The set of services was annotated according to the SAWSDL [11] specification. To semantically describe the user requests and the service capabilities we developed a domain ontology which stores 200 concepts organized on 8 hierarchic levels. As a result, the user request is specified by a set of ontological concepts annotating the provided inputs and requested outputs and by weights indicating the relevance of quality of service compared to semantic quality. Services are semantically described with concepts annotating their functionality, input and output parameters. In this section we partially trace the composition and selection algorithms for the user request given in table III and evaluate the experimental results. The user request specifies the need for a composed service able to make travel arrangements (search and book accommodation, flight and car). In our experiments we have considered four QoS attributes: availability (*Av*), reliability (*Rel*), cost (*Ct*) and response time (*Rt*).

TABLE III. USER REQUEST

$in_1, in_2, \ldots$	$out_1, out_2, \ldots$	*QoS weights*	*SemQ weight*
SourceCity; DestinationCity; StartDate; EndDate; Hotel; MediterraneanFood; NumberOfPersons; NumberOfRooms	Vacation Price	Total QoS: 0.25; Av: 0.15; Rel: 0.25; Ct: 0.45; Rt: 0.15	0.75

Based on the user request and on the available set of services, the composition algorithm iteratively builds the

EPG along with the *MSL*. The resulting *EPG* contains four layers of service clusters. Tables IV and V illustrate two service clusters, one from the second layer which groups services that search accommodation and the other one from the third layer which groups services that book accommodation.

TABLE IV. THE SET OF WEB SERVICES FROM CLUSTER 1 ON LAYER 1 OF THE *EPG*

WS code	***WS Operation***	***in_1, in_2, …***	***out_1, out_2, …***	***QoS***
s_{111}	Search Hotel	DestinationCity; Hotel	HotelName	Av: 3.0; Rel: 2.5; Ct: 0.45; Rt: 1.25
s_{112}	Search ExoticHotel	EuropeanExotic DestinationCity; Accommodation	Mediterranean HotelName	Av: 3.5; Rel: 1; Ct: 2; Rt: 2.75
s_{113}	Search European Exotic Hotel	European DestinationCity; Hotel	Luxury Mediterranean Hotel Name	Av: 4.0; Rel:1.75; Ct: 4.5; Rt: 2

TABLE V. THE SET OF WEB SERVICES FROM CLUSTER 1 ON LAYER 3 OF THE *EPG*

WS code	***WS Operation***	***in_1, in_2, …***	***out_1, out_2, …***	***QoS***
s_{211}	Book Hotel	HotelName; StartDate; EndDate; NumberOfHotelRooms; NumberOf Persons; Breakfast	HotelPrice	Av:0.5; Rel: 1; Ct: 1.25; Rt: 3.5
s_{212}	Book European ExoticHotel	Luxury Mediterranean HotelName; StartDate; EndDate; NumberOf Persons	HotelPrice	Av: 2.05; Rel: 3; Ct: 2; Rt: 2.95
s_{213}	Book Exotic Hotel	Mediterranean HotelName; StartDate; EndDate; NumberOfHotelRooms; NumberOf Persons; Mediterranean Food	HotelPrice	Av: 3.05; Rel: 4; Ct: 4.95; Rt: 1

Table VI presents a subset of the semantic similarity links stored in the *MSL* established between the services from tables IV and V.

TABLE VI. EXAMPLES OF SEMANTIC SIMILARITY LINKS

s_i	s_j	***Semantic similarity link between the outputs of s_i and the inputs of s_j***
s_{111}	s_{211}	({(HotelName, HotelName)}, 1)
s_{112}	s_{211}	({(MediterraneanHotelName, HotelName)}, 0.79)
s_{113}	s_{213}	({(LuxuryMediterraneanHotelName, MediterraneanHotelName)}, 0.902)

For the trip planning scenario we performed two experiments. In the first experiment we generated the entire set of solutions to identify the optimal composition solution which has a score of 9.556. Then, we performed several experiments on our bee-inspired selection technique to check whether the optimal solution identified is indeed the best or is very close to the best. In our experiments we varied the number of bees, from 10 to 150. We noticed that for a stopping condition threshold of 9.5, and for the following parameter values: $\Delta1 = 0.5$, $\Delta2 = 0.2$, wQoS = 0.25 wSem = 0.75, the selection algorithm identifies the optimal solution (for the most of the cases) or identifies a solution very close to the optimal one. Table VII presents the number of iterations in which the optimal solution or a solution close to the optimal one was identified by different numbers of bees.

TABLE VII. NUMBER OF BEES AND THE NUMBER OF ITERATIONS TO REACH THE OPTIMAL SOLUTION OR A SOLUTION CLOSE TO THE OPTIMAL ONE

Number of bees	***Number of iterations***	***Optimal solution score***
5	52	9.556
10	75	9.556
50	11	9.556
70	6	9.549
80	18	9.556
90	10	9.556
100	11	9.556
110	12	9.556
140	7	9.556
150	5	9.549

By analyzing the experimental results we noticed that if the number of bees used in the search process is high, the optimal solution is obtained in less iterations but with a higher execution time.

V. RELATED WORK

In the past few years, biologic behaviors, such as foraging in the case of insects or birds, have inspired several optimization techniques. This section reviews biologically-inspired techniques applied to the problem of selecting the optimal service composition solution.

In [1] and [2], genetic algorithms are used to find the optimal composition solution. The composition method is based on a given service abstract workflow, where each abstract service has a set of candidate concrete Web services with different associated QoS values. Genetic algorithms are used to bind concrete services to the abstract ones aiming at identifying the optimal workflow instance in terms of QoS attributes. The genome is encoded as an integer array in [1] and as a binary string in [2], where each position is associated to an abstract service in the workflow and indicates the concrete service which is selected to be used. Both approaches make use of genetic operators and fitness functions applied on the genome to find the optimal composition solutions. In the case of genetic operators, both approaches perform random mutations to generate new workflow instances. In the case of fitness functions, [1] proposes the use of a dynamic fitness function which penalizes the individuals that do not meet the constraints in each generation, thus favoring a quicker convergence toward an optimal solution. Compared to [1], the approach in [2] proposes three fitness functions, one associated to each considered QoS attribute, in order to increase the probability of finding the optimal solution. Still, genetic algorithms are not scalable due to the fixed way in which the genomes and chromosomes need to be encoded. Moreover, there is the risk that the genetic algorithm may get stuck in a local optimum.

Particle Swarm Optimization (PSO) [3], an optimization technique inspired by the behavior of foraging birds is a viable alternative to genetic algorithms since it converges more rapidly and is more scalable. In [4], PSO is used to obtain the optimal service composition solution. As in [1] and [2], the composition approach is based on a predefined workflow associated to concrete candidate services. The position of the particle is represented by the concrete services mapped on workflow tasks, while velocity indicates how the concrete services should be changed. By performing the operations of addition, subtraction and multiplication (adapted to the case of Web service composition) on positions and velocities, the proposed selection algorithm identifies optimal composition solutions based on QoS attributes. However, similar with genetic algorithms, by applying PSO when searching for the optimal composition solution, the problem of early stagnancy in a local optimum cannot be avoided. To address this issue, the authors have introduced a variation operator to enlarge the searching space.

In [5], genetic algorithms and PSO are combined for service composition aiming at balancing the local and the global search and thus avoiding early stagnation. The mapping of service composition to PSO is similar to the one in [4], but in this case the crossover operation is used to add diversity by introducing new services and favoring the exploration of new search spaces.

Our approach to selecting the optimal composition solution is inspired by the foraging behavior of honey-bees. The differences between our approach and the ones presented above are the following:

- Our composition method does not start from a predefined workflow as in [1], [2], [4] and [5] but from the user request. As a result, a multi-layered planning graph of services is obtained, each layer containing sets of services providing different functionalities.
- In our approach, the problem of local stagnancy in selecting the optimal composition solution encountered in [1], [2] and [4] is overcome by using a bee-inspired technique which favors the exploration of several candidates for the optimal solution.
- Our criteria for evaluating the quality of a candidate solution include not only QoS attributes as in [1], [2], [4] and [5] but also the property of semantic quality.

In our case, the replacement of a concrete candidate service is not done randomly as in [1], [2] and [4], but according to QoS attributes and semantic quality.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a new method for selecting the optimal composition solution which uses a technique inspired by the behavior of bees in nature. The bee-inspired technique optimizes the selection process without considering the entire search space and avoids the local optimum stagnancy problem. The selection algorithm uses two structures, namely an enhanced planning graph and a matrix of semantic links, resulted from Web service composition, to incrementally build better solutions until an acceptable one is found taking into consideration the QoS and semantic quality. The enhanced planning graph and the matrix of semantic links store the set of services that can be composed as well as information about their degree of match. We obtained promising results in testing our bee-inspired selection method on a case study for making travel arrangements involving 110 semantic Web services.

As future work we intend to test the selection method on larger and more complex sets of Web services and to speed-up the process by parallelizing the bees search operations.

REFERENCES

[1] G. Canfora, M. Di Renta, R. Esposito, M. L. Villani, "An Approach for QoS aware Service Composition based on Genetic Algorithms", Proceedings of GECCO'05, pp. 1069-1075, Washington, DC, USA, 2005.

[2] J. Wang, Y. Hou, "Optimal Web Service Selection based on Multi-Objective Genetic Algorithm", Proceedings of the ISCID 2008, pp. 553-556, Wuhan, 2008.

[3] J. Kennedy, R.C. Eberhart, "Particle swarm optimization", Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ. pp. 1942-1948, 1995.

[4] C. Ming, W. Zhen-wu, "An Approach for Web Services Composition Based on QoS and Discrete Particle Swarm Optimization", Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, pp. 37-41, Washington DC, USA, 2007.

[5] J.Liu, J. Li, K. Liu,W. Wei, "A Hybrid Genetic and Particle Swarm Algorithm for Service Composition", Proceedings of the Sixth International Conference on Advanced Language Processing and Web Information Technology, pp.564-567, 2007.

[6] C. B. Pop, V. R. Chifu, I. Salomie, M. Dinsoreanu, I. Vartic, M. Vlad, "Immune-inspired Web Service Composition Framework", Proceedings of the 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC2009), September 26-29, Timisoara Romania, ISBN 978-0-7695-3964-5, pp. 376-383, 2009.

[7] S. Russell, S, P. Norvig, "Artificial Intelligence: A Modern Approach." Upper Saddle River, NJ: Prentice Hall/Pearson Education, ISBN: 0137903952, 2003.

[8] M. Paolucci,, et al., "Semantic Matching of Web Services Capabilities", LNCS, vol.2342, Springer Berlin / Heidelberg, pp. 333-347, 2002.

[9] D.Teodorovic, M. Dell'Orco, "Bee Colony Optimization – A Cooperative Learning Approach to Complex Transportation Problems", Advanced OR and AI Methods in Transportation, pp. 51—60, 2005.

[10] M. Dorigo, M. Birattari, T. Stützle, "Ant Colony Optimization--Artificial Ants as a Computational Intelligence Technique", IEEE Computational Intelligence Magazine, 2006.

[11] SAWSDL, http://www.w3.org/2002/ws/sawsdl/spec/

[12] D. Skoutas, A. Simitsis, T. Sellis, "A Ranking Mechanism for Semantic Web Service Discovery", Proceedings of the IEEE Congress on Services, Salt Lake City, UT, pp.41-48, 2007.