ANT-BASED METHODS FOR SEMANTIC WEB SERVICE DISCOVERY AND COMPOSITION

Viorica Rozina Chifu, Cristina Bianca Pop, Ioan Salomie, Mihaela Dinsoreanu,
Tudor David, Vlad Acretoaie
Department of Computer Science
Technical University of Cluj-Napoca
26-28 Baritiu str., Cluj-Napoca, Romania
Email: {Viorica.Chifu, Cristina.Pop, Ioan.Salomie, Mihaela.Dinsoreanu}@cs.utcluj.ro

ABSTRACT

Biological swarms are capable of successfully dealing with complex life problems such as foraging, nest construction or survivability only by obeying to local rules and by communicating indirectly without the existence of a central coordinator. It has been noticed that the behavior of biological swarms offers us many clues regarding the construction of efficient methods applicable in various domains of computer science. Inspired by the behavior of ants, we propose in this paper two methods for semantic Web service discovery and composition. For efficiently discovering the services that will be involved in the composition process we choose to organize the available services in clusters according to the similarity between their semantic descriptions. To semantically compose services we propose a graph model that stores all the composition solutions for a specified user request. We further apply an ant-inspired selection method on the composition graph to identify the optimal composition solution according to user constraints. We have tested and validated the two ant-inspired methods on an extended version of the SAWSDL-TC benchmark collection.

Keywords: ontology, semantic Web service, semantic matching, ant-based service clustering, composition graph model, ant-based selection.

1 INTRODUCTION

The growing popularity of Web services as a method of publishing software functionality over the Internet has determined researchers to focus their attention on finding efficient ways to take advantage of the service-oriented architecture. Web service discovery and composition are topics that have been widely discussed in the area of distributed systems for the past few years. The prospects of discovering services and composing them to perform more complex tasks have considerably improved with the introduction of semantic Web services. However, the existence of a large number of available services makes the discovery process inefficient, unless services are grouped according to specific criteria. Service clustering is a solution to this problem. The objective of clustering is to gather data items that share common features into groups called clusters. Another issue in semantic Web service composition is the selection of the optimal composition solution. 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 proposes two methods for semantic service discovery and composition inspired by the behavior of ants. For efficiently discovering the services that will be involved in the composition process we choose to organize the available services in clusters according to the similarity between their semantic descriptions. To semantically compose services we propose a graph model that stores all the composition solutions for a specified user request. We further apply an ant-inspired selection method on the composition graph to identify the optimal composition solution according to user constraints. This paper is structured as follows. Section 2 reviews some of the existing works related to service clustering and composition. Section 3 presents the ant-based service clustering method, while section 4 illustrates our composition method. In section 5 we propose an ant-based method for selecting the optimal composition solution. We end our paper with experimental results and conclusions.

2 RELATED WORK

There is a considerable amount of research done in the domain of automatic Web service clustering and composition. In this section we review some of
the existing works related to the service clustering and composition.

2.1 Service Clustering

In [1], text mining techniques combined with an ant inspired algorithm are applied for clustering service syntactic descriptions gathered from the Web. The service descriptions are obtained with the help of a search engine. The proposed approach considers the service syntactic features as clustering criteria such as WSDL files, service host or service name. These features are obtained by performing text analysis. In order to cluster services, authors use the Tree-Traversing Ant algorithm and introduce a measure of similarity based on the considered clustering criteria.

A statistical clustering method is discussed in [2] which identifies clusters of services based on a given query. The method combines distance measures applied on a vector space associated to a service repository with the k-means method. As in [1], the clustering method analyses the services’ syntactic descriptions.

In [3], authors propose a multi-stage semantic-based service clustering method. The method (i) pre-processes the WSDL files for consistency checking, (ii) semantically annotates the service descriptions by mapping the WSDL elements to OWL-S and (iii) clusters the services based on their semantic provided in the OWL-S Service Profile and Grounding. In the clustering process the Jaccard coefficient is used to measure the semantic similarity between two services. The similarity scores are stored in a matrix. By using this matrix in a hierarchical agglomerative clustering approach, the proposed method clusters services and stores the clusters in an UDDI registry.

In [4], authors propose a two-step semantic-based clustering algorithm which provides the relevant services in response to a search query. First, the algorithm eliminates the irrelevant services from the query perspective. Then, the Probabilistic Latent Semantic Analysis is used to identify common matching concepts between the remaining services and the issued query. Our clustering method is inspired by the behavior of real ants which cluster corpses to build cemeteries. As opposed to [1, 2] that cluster services based on their syntactic descriptions which is not always feasible, our method clusters the services based on their semantic descriptions. By considering the semantic descriptions, the clustering process can be improved, as it enables reasoning which may lead to refined results.

2.2 Service Composition

Methods based on ant colony optimization [5] have been proposed for selecting the optimal service composition in [6], [7] and [8] using the QoS attributes as selection criteria. All the approaches apply an ant-based selection method on a composition graph. In [6] and [8] the graph nodes represent abstract services having associated sets of concrete services, while the edges denote the interactions between services.

In contrast with [6] and [8], in [7] the graph edges represent services, while the nodes describe how these services interact. The QoS models defined in [6], [8] and [7] are similar and make distinction between the QoS attributes that should be minimized or maximized. The main difference between the three approaches derive from the way the formulas proposed in ant colony optimization are adapted for Web service composition. In addition, [6] and [7] introduce the concept of multi-pheromone for representing the QoS attributes, while [8] considers a chaos operator to avoid stagnation in a local optimum.

Our selection method also applies ant colony optimization to identify the optimal service composition solution. The main differences between our approach and the ones presented in [6], [7] and [8] concern the structure of the composition graph, the selection criteria and the way in which we adapt the ant colony optimization technique for selecting the optimal composition. In our case, a node of the composition graph is a cluster of similar services, while an edge links two clusters if there is a high degree of semantic match between them. As opposed to [6] and [8] which build the composition graph by mapping concrete services to a predefined abstract workflow, we dynamically generate the composition graph based on a service composition request. We consider as selection criteria not only the QoS attributes but also the semantic similarity between the services involved in composition in contrast to the other approaches reported in the research literature. In the case of adapting ant colony optimization to the problem of selecting the optimal composition we have redefined the probability that an ant chooses a service by considering both the QoS attributes and the semantic quality.

3 SERVICE CLUSTERING

A prerequisite for composing Web services is the capability of discovering suitable services which can be involved in the composition process. To improve the efficiency of the services discovery process we have chosen to organize the available services into clusters based on their semantic similarity. For grouping similar services, we have adapted the ant-based algorithm described in [9]. We consider that two services are similar if there is a degree of match (DoM) between their semantic descriptions. To evaluate the DoM between the
semantic descriptions of two services we define a set of metrics which take into consideration the concept and property hierarchies from the ontology. In our approach, the semantic description of a service includes a set of ontology concepts annotating the service input and output parameters.

3.1 Metrics for Evaluating the DoM between Two Services

To evaluate the degree of match DoM_service between two services we adopt a decomposition technique. First we evaluate the DoM_input / DoM_output between the concepts describing the input/output parameters of the considered services. Second, we evaluate the DoM_service between the two considered services based on DoM_input and DoM_output.

The evaluation method is the same for DoM_input and DoM_output and implies the evaluation of DoMs between pairs of concepts annotating the input/output service parameters and then combining the results. The degree of match DoM_f (Eq. (1)) between a pair of concepts has two components: one representing the match between the concepts themselves (DoM_f) and one representing the match between their properties (DoM_p).

\[
\text{DoM}_f(c_1, c_2) = \frac{\text{DoM}_f(c_1, c_2) + \text{DoM}_p(p(c_1), p(c_2))}{2}
\]

(1)

For computing DoM_p we generalized the formula defined in [10]. In our formula (see Eq. (2)) we also consider the general case in which there is a degree of match between two concepts even if they are siblings. In addition, by introducing a multiplication between two sub-unitary fractions, we ensure that the DoM between two concepts that are not in an ancestor-descendant relation is higher than the DoM between two concepts that are not in such a relation. The values of DoM are in the interval [0, 1].

\[
\text{DoM}_p(c_1, c_2) = \frac{\vert \{c \mid c \subseteq c_1\} \vert \ast \vert \{c \mid c \subseteq \text{nca}(c_1, c_2)\} \vert}{\vert \{c \mid c \subseteq \text{nca}(c_1, c_2)\} \vert}
\]

(2)

In Eq. (2) the inclusion operator denotes the subsumption relationship and \(\text{nca}\) is the nearest common ancestor of two concepts in the considered ontology.

For computing DoM_f we take into account all the properties of two concepts (see Eq. (3)).

\[
\text{DoM}_f(p(c_1), p(c_2)) = \frac{\text{DoM}_f(p(c_1), p(c_2)) + \text{DoM}_p(p(c_1), p(c_2))}{2}
\]

(3)

In Eq. (3), DoM_f (p(c_1), p(c_2)) is the degree of match between the sets of properties associated to the concepts \(c_1\) and \(c_2\).

In Eq. (4), \(\text{DoM}_s\) is the degree of match between two properties \(p_1(c_1)\) and \(p_2(c_2)\) of concepts \(c_1\) and \(c_2\) and is computed similarly \(\text{DoM}_c\).

The DoM between the sets of concepts annotating the input/output parameters of two services \(s_1\) and \(s_2\) is computed with Eq. (5):

\[
\text{DoM}_{\text{input/output}}(s_1, s_2) = \text{DoM}_{\text{input}}(s_1, s_2) \ast \text{DoM}_{\text{output}}(s_1, s_2)
\]

(5)

In Eq. (5), \(\text{DoM}_{\text{input/output}}\) is computed as follows:

\[
\text{DoM}_{\text{input/output}}(s_a, s_b) = \sum_{i=1}^{\vert C(s_a) \vert} \max_{j=1}^{\vert C(s_b) \vert} (\text{DoM}_f(c_i, c_j)) / \vert C(s_a) \vert
\]

(6)

where \(c_j\) is a concept annotating an input/output of the service \(s_b\) and \(C\) is the set of concepts annotating the input/output parameters of the two services.

Finally, \(\text{DoM}_{\text{service}}\) is computed as an arithmetical mean between \(\text{DoM}_{\text{input}}\) and \(\text{DoM}_{\text{output}}\) giving equal importance to the inputs and outputs of the two services.

3.2 The Ant-based Service Clustering Algorithm

The service clustering process is similar to the way ants build cemeteries by clustering the corpses present in the vicinity of their colony. Therefore, we chose to adapt an ant-based clustering algorithm [9] to the problem of service clustering. Consequently, the clustering behavior of ants is mapped to the case of service clustering as follows: (i) an ant becomes an artificial ant, (ii) the environment in which the ant moves becomes a toroidal grid [9] (iii) a corpse the ant moves becomes a service and (iv) a cluster of corpses the ant builds becomes a cluster of similar services. In our approach, we consider that two services are similar if the degree of match between their semantic descriptions is higher than a specified threshold. The semantic similarity between two services is measured using the equations presented in sub-section 3.2.1.

The clustering algorithm (Algorithm_1) takes as input a set of services and the number of ants used in the clustering process. The algorithm can be applied in two cases: when it creates clusters of services from scratch and when it distributes a set of services to an existing set of clusters \(\text{CLS}\). The first case appears when the condition \(s_{\text{Nr}} > \Delta \ast |\text{CLS}|\) is satisfied - a number \(s_{\text{Nr}}\) of services have been distributed among the clusters in \(\text{CLS}\) and the re-
clustering of the whole CLS should be performed. The following steps are performed in the first case: (1) the services in CLS are partitioned (PARTITION) according to the number of their inputs; (2) the set CLS becomes empty; (3) each service partition is submitted to the ant clustering algorithm (Algorithm_2) and the resulting clusters are added to CLS; (4) the number sNr of services processed from the last clustering becomes 0. For the second case, the condition sNr > Δ * |CLS| does not hold and it means that the services given as input need to be distributed among the clusters in CLS (ADD-SERVICE).

Algorithm_1: Ant_Service_Clustering

Inputs: S – the set of services (to be clustered); nrAnts – the number of artificial ants; sNr - the number of services processed from the last clustering.
Output: CLS - the updated set of clusters;
Comments: SP – set of service partitions;
begin
if (sNr > Δ * |CLS|) then
    SP = Partition (CLS); CLS = Ø
    foreach sp in SP do
        CLS = CLS U Ant_Clustering (sp, nrAnts)
    end foreach
    sNr = 0
else
    foreach s in S do
        CLS = Add_Service (CLS, s); sNr = sNr + 1
    end foreach
end if
return CLS
end

Algorithm_2 clusters a given set of services. The services are first randomly placed on the GRID (Distribute_Services_Randomly) and the artificial ants are randomly associated to a service (Pick_Random_Service) and to a location (Pick_Random_Location). The artificial ants then start moving services around for a fixed number of iterations, considering a move at each iteration step. In each iteration step, an artificial ant performs a move in the grid (Move) and decides whether to place the service in a grid element (Decide_Drop). The first move is influenced by the best position determined so far, which is kept in the memory of the artificial ant. If the artificial ant does not decide to drop the service in this best position, then the ant performs random moves.

The decision of placing a service in a grid element is taken according to a probability $P_{\text{drop}}$ value defined as follows [9]:

$$P_{\text{drop}}(\text{ant, ant.s}) = \min(1.0, f^a) \quad (7)$$

In Eq. (7) we have adapted the definition of $f$ for the case of semantic Web services:

$$f = \begin{cases} 
\frac{1}{N} \sum_{j \in L_j} (1 - \frac{\Delta(s, s_j)}{\alpha}), & \text{if } (1 - \frac{\Delta(s, s_j)}{\alpha}) > 0 \\
0, & \text{otherwise} 
\end{cases} \quad (8)$$

where

- $s$ is the service the artificial ant carries, while $s_j$ is a service located in the neighborhood of the ant.
- $N$ normalizes $f$ and depends on the neighborhood size and the number of services in that neighborhood.
- $\Delta$ evaluates the degree of match between two services $s_i$ and $s_j$ and is computed as:

$$\Delta(s_i, s_j) = 1 - \text{DoM}(s_i, s_j) \quad (9)$$

- $L_j$ is the set of grid neighborhood services of the ant position:

$$L_j = \{ s_i | \forall s_j, |i - s_i| < \frac{\delta}{2} \text{ and } |j - s_i| < \frac{\delta}{2} \} \quad (10)$$

where $\delta$ is a value equal to the size of the neighborhood. $\delta$ increases linearly during the run of the clustering algorithm.

Algorithm_2: Ant_Clustering

Inputs: Ants – the set of ants; S – the set of services that will be clustered; nrIt – the number of iterations
Output: CLS – the set of clusters
Comments: G – the toroidal grid
begin
Distribute_Services_Randomly (S, G)
foreach ant in Ants do
    Pick_Random_Service (ant, S)
    Pick_Random_Location (ant, G)
end foreach
for $i = 0$ to nrIt do
    ant = Choose_Random_Ant(Ants)
    Move (ant, G)
    if (Decide_Drop(ant)) then
        Drop_Service(ant, G)
        while (Decide_Pick(ant) = false) do
            Move (ant, G)
        end while
        Pick_Service (ant, G)
    end if
end for
CLS = Retrieve_Clusters (S, G)
return CLS
end

If the ant decides to drop the service (Decide_Drop), it places the service in a grid element (Drop_Service) and then moves in the grid (Move) until it decides to pick-up another service (Decide_Pick). Just as in the case of dropping a service, the decision to pick-up a service is taken...
according to the probability $P_{\text{pick}}$ defined as follows [9]:

$$P_{\text{pick}}(ant, s) = \min(1.0, \frac{1}{f^2})$$  \hspace{1cm} (11)

where $f$ is defined as in Eq. (8). At the end of Algorithm 2, once similar services are within a neighborhood, a cluster retrieval algorithm is run, which identifies the neighborhood of services and organizes them into clusters (Retrieve_Clusters). The neighborhood is established using a cluster retrieval distance (CRD) between two services.

4 THE COMPOSITION GRAPH

In this section we propose a composition graph model which represents the search space for a selection method that identifies the optimal composition according to the user constraints.

4.1 The Composition Graph Model

The composition graph is a directed acyclic graph in which a node is represented as a cluster of similar services and an edge denotes a semantic relationship between two clusters.

In our approach, we consider three types of graph nodes: several service nodes, one input node and one output node. A service node contains a service cluster, with the property that there is a high degree of semantic match (DoM) between the services in the cluster. We allow services with similar functionalities, inputs and outputs to be part of the same cluster. The services of a cluster may have the same functionality and also may differ with at most one input/output parameter.

The input node contains a cluster with a single service which only has outputs representing a set of ontology concepts describing the user providing inputs. The output node on the other hand, contains a cluster with a single service having just inputs representing the concepts describing the user requested outputs.

A directed edge links a pair of service clusters if there is a high degree of semantic match between the outputs of one of the clusters and the inputs of the other cluster.

The construction of the composition graph starts with the input node. Then, the graph is expanded with new clusters of services which have a high degree of match with the cluster contained in the input node. This way, the expansion of the graph is performed until all the inputs of the output node are satisfied. In the construction process, a service cluster is added only if the clusters already present in the graph provide all the outputs required for its execution. The service clusters are provided by a discovery method which is detailed in the following sub-section.

A composition solution is a directed acyclic sub-graph which contains the service of the input node, a set of services (one service from each cluster in the graph) and the service of the output node.

4.2 The Composition Algorithm

The composition algorithm (Algorithm 3) takes as inputs (i) the user request, (ii) the set of services available in a repository and (iii) two threshold values, one for intra-cluster matching and another one for concept matching. First the composition graph is initialized (Initialize_Graph) based on the user request so that it contains only the input node. Then, the algorithm iteratively performs the following two operations: (1) a set $CL$ of service clusters are discovered (Discovery) based on the semantic matching between the concepts describing the inputs of these services and the ones describing the outputs of the services that are already in the graph, (2) for each discovered cluster $c$ a corresponding node is created and added to the composition graph and as a result the set of graph edges is updated (Update_Edges). A new edge is added between two nodes if there is a semantic matching between the clusters associated to these nodes.

---------------
Algorithm_3: Build_Composition_Graph
---------------

Input: $\text{req} = (\text{in}, \text{out})$ - the user request; $\text{CLS}_R$ - set of available service clusters; $c\text{Th} -$ intra-cluster matching threshold; $\text{cTh} -$ concept matching threshold.
Output: $G = (V, E)$ - the composition graph containing a set of nodes $V$ and a set of edges $E$.

begin
$v_{\text{in}} = \text{Generate_Input_Node}(\text{req.in})$
$v_{\text{out}} = \text{Generate_Output_Node}(\text{req.out})$
$G = \text{Initialize_Graph}(v_{\text{in}})$
while ($v_{\text{out}} \notin G.V$) or (!Fixed_Point($G$)) do
  $CL = \text{Discovery}(\text{CLS}_R, G.V, c\text{Th})$
  foreach $c \in CL$ do
    $G.V = G.V \cup \{c\}$
    $G.E = \text{Update_Edges}(G.V, c\text{Th})$
  end for
end while
if ($v_{\text{out}} \notin G.V$) then return null
$G = \text{Prune}(G)$
return $G$

The algorithm ends either when the user requested output parameters are provided by the services added to the composition graph or when no new service clusters have been added to the composition graph for a predefined number of iterations (Fixed_Point). If at least one composition solution has been found we employ a pruning process.

The discovery algorithm (Algorithm 4) takes as input the following parameters: (i) the set $\text{CLS}_R$ of available service clusters returned by the ant-based
clustering algorithm, (ii) a set $CLS_G$ of already discovered clusters that are part of the composition graph – initially, the list contains a generic cluster with no inputs, (iii) a concept matching threshold $cTh$ and (iv) an intra-cluster matching threshold $ciTh$.

Algorithm 4: Cluster_Discove

Inputs: $CLS_R$ – set of available service clusters; $CLS_G$ – set of service clusters part of the composition graph; $cTh$ – concept matching threshold; $ciTh$ – intra-cluster matching threshold;
Output: CLS$G$ – the updated set of service clusters part of the composition graph;
Comments:
begin
foreach $cl$ in $CLS_R$ do
if ($cl$ not in $CLS_G$) then
    add = true
    missing = false
end if
foreach $cl$.in in $cl$.in do
    maxDoM = 0
foreach $cl'$ in $CLS_G$ do
    foreach $cl$.out in $cl$.out do
        if (DoM($cl$.in, $cl$.out) > maxDoM) then
            maxDoM = DoM($cl$.in, $cl$.out)
        end if
    end foreach
    if (maxDoM < $cTh$) then
        if (missing == true or DoM_In_Cluster($cl$.in, $cl$) > $cTh$) then
            add = false
            else
                missing = true
            end if
        end if
    end foreach
end foreach
if (add == true) then
    CLS$G$ = CLS$G$ U $cl$
end if
end foreach
return CLS$G$
end

The algorithm iterates the set $CLS_R$ provided by the ant-based clustering algorithm, considering those clusters that are not already present in the composition graph. A service cluster can be added to the set $CLS_G$ even if it has at most one input that has no corresponding output in $CLS_G$. The condition for adding the discovered cluster to $CLS_G$ is that the missing input is not representative for all the services in the tested cluster. Once a candidate cluster from $CLS_R$ is selected, the best matching output provided by the clusters in $CLS_G$ is selected for each input of the candidate cluster. If the matching value is below the concept matching threshold $cTh$ and either the tested input is representative for the cluster or there has already been an unmatched input for this cluster ($DoM_{In\_Cluster}$), the cluster will not be discovered. If the matching value is below the concept matching threshold $cTh$ and none of the other conditions are met, the tested input is marked as missing so as to not allow a second missing input for this cluster.

5 SELECTING THE OPTIMAL SOLUTION IN WEB SERVICE COMPOSITION

For finding the optimal composition we adapted the Ant Colony Optimization (ACO) meta-heuristic [5]. Our ant-inspired selection method identifies the optimal composition solution encoded in the composition graph. To establish whether a solution is optimal according to QoS user preferences and semantic quality we define an appropriate fitness function.

5.1 Mapping ACO to the Problem of Selecting the Optimal Composition Solution

The ACO meta-heuristic relies on a set of artificial ants which communicate with each other to solve combinatorial optimization problems. Just as in the ACO meta-heuristic we consider a set of artificial ants that traverse the composition graph in order to find the optimal composition solution. In its search, an artificial ant has to choose at each step a new edge that links the service where the ant is currently positioned with a new service. The choice is stochastically determined with the probability $p$ [5] defined as follows:

$$p_{s_i,s_j} = \begin{cases} 
\tau_{ij}^\alpha \times \eta_{ij}^\beta & \text{if } c_{pq} \in N(s^p) , \\
\sum_{s_i \in V} \tau_{ij}^\alpha \times \eta_{ij}^\beta & \text{otherwise.} 
\end{cases} \quad (12)$$

where $\eta_{ij}$ represents some heuristic information and $\tau_{ij}$ is the level of pheromone on the edge leading to a candidate service. The heuristic information associated to the edge linking the service $s_i$ where the ant is currently positioned and the candidate service $s_j$ is computed with the following equation:

$$\eta_{ij} = \frac{w_{QoS} \times QoS(s_j) + w_{Match} \times DoM_{services}(s_1, s_2)}{w_{QoS} + w_{Match}} \quad (13)$$

where:
- $w_{QoS}$ and $w_{Match}$ are the weights representing the importance of the QoS score compared to the degree of match;
- $QoS$ represents the score of the candidate service and is computed as a weighted mean of the QoS.
attributes considered in the underlying QoS model:

- \( DoM_{services} \) represents the degree of match between services \( s_1 \) and \( s_2 \) (see sub-section 3.1).

Initially, the level of pheromone corresponding to each edge is assigned a constant value \( \tau_0 \). Within the search process the level of pheromone is updated in the following two situations: (1) at each step made by an artificial ant in the composition graph – local update and (2) at the end of an iteration when an ant identifies a composition solution – offline update. The local update is performed using the equation defined in [5]. For performing the offline update we adapt the formula defined in [5] as follows:

\[
\tau_{i,j} = (1 - \rho) \times \tau_{i,j} + \rho \times \Delta \tau_{i,j}
\]

(14)

where \( \Delta \tau_{i,j} = \frac{1}{S_{best}} \), where \( S_{best} \) is the best score of a composition solution found so far and \( \rho \) is the pheromone evaporation rate.

5.2 The Ant-based Selection Algorithm

The selection algorithm (Algorithm_5) uses a set of artificial ants to iteratively search for the optimal composition solution in the composition graph. Initially, all ants are placed in randomly chosen graph nodes (Initialize_Ants). Then, in their search, each ant tries to identify a composition solution (Find_Solution) by performing two types of searches: backward and forward.

A backward search is performed by the ant to identify the services that provide the inputs of the service where the ant is currently positioned. This process is repeated until there are no more services whose inputs have no correspondents to outputs of other services.

In the forward search, the ant continues to expand the partial solution identified so far by iteratively picking forward edges towards the output node. Once a new node is picked, the ant will again go on back edges searching for services that provide all its inputs.

Both in the backward and the forward searches, a service is chosen stochastically with the probability defined in Eq. (12).

When an ant is searching for a composition solution, several local pheromone updates are performed. In addition, an offline pheromone update (according to Eq. (14)) is performed after all ants have identified a composition solution aiming to promote the best solution found so far.

The algorithm runs until no changes of the optimal solution found so far have been detected for a predefined number of iterations \( noIt \). In the end, the algorithm returns a ranked set of composition solutions, the first being the optimal one.

---

Algorithm_5: Ant_based_Selection

```
Input: noIt - the number of iterations; noAnts - the number of artificial ants; \( G = (V, E) \) - the composition graph;
Output: SOL - the set of the best composition solutions;
Comments: Find_Min_Score determines the lowest score of a solution from SOL; Find_Max_Score determines the highest score of a solution from SOL; Score returns the score of a composition solution; Remove_Worse removes a solution from SOL which has the lowest score; Add adds a solution to SOL.

begin
SOL = \( \emptyset \)
noItModif = 0
Ants = Initialize_Ants(noAnts)
while (noItModif < noIt) do
    noItModif = noItModif + 1
    max = 0
    foreach a \( \in \) Ants do
        result = Find_Solution(G, a)
        minScore = Find_Min_Score(SOL)
        maxScore = Find_Max_Score(SOL)
        if (minScore < Score(result)) then
            SOL = Remove_Worse(SOL)
        end if
        SOL = Add(SOL, result)
    end foreach
    noAnts = Add_Ants(noAnts)
end while
return SOL
```
```

6 FRAMEWORK IMPLEMENTATION AND EXPERIMENTAL RESULTS

To validate our approach to automatically discover and compose semantic Web services, we have developed an experimental framework that integrates the ant-based clustering method, the composition method as well as the ant-inspired selection method presented above.

6.1 Framework Architecture

The architecture of the framework is presented in Figure 1. In what follows, we describe the functionality of each framework component.

The Ontology Driven Graphical User Interface guides the user in the composition process by providing a controlled language that uses the ontology concepts. It enables the user to specify composition requirements in terms of functional and non functional parameters. The Semantic Web Service Repository is a repository of semantically annotated services based on the SAWSDL standard [11]. The semantic descriptions generated for each
service are based on a set of domain ontologies, which serve as a common vocabulary used by the descriptions.

Figure 1: Framework architecture

The Service Matching Module is responsible for evaluating the semantic similarity between two services. The module interacts directly with the ontology repository. The Service Clustering Module searches the Semantic Web Service Repository for services with high semantic similarity between them, and groups them together. The Service Discovery Module searches for clusters of services based on some available inputs (given as ontology concepts). The Service Composition Module interacts with the Service Discovery Module to build the composition graph which stores all the possible solutions for a specified user request. The Selection Module interacts with the Service Composition Module to find the optimal composition solution according to QoS attributes and semantic quality. The Selection Module returns a ranked set of composition solutions, the first one being the optimal solution. The user is allowed to choose the solution he prefers. After a service composition solution is selected, the Service Invocation Module will actually execute the services part of the composition, and present the user the final output.

6.2 Experimental Results

We have tested and evaluated our framework on the SAWSDL-TC service test collection [12]. In our experiments we have used 737 Web services which are semantically annotated using concepts from several domain ontologies according to the SAWSDL specification. The services belong to the following domains: education, medical care, food, travel, communication, economy and weapon. For composition we have extended the SAWSDL-TC standard collection with new services from the medical care domain. In this section we present and analyze the experimental results obtained by applying our ant-inspired methods on the extended SAWSDL-TC collection.

6.2.1 Results Evaluation for the Ant-based Service Clustering Method

Experiments were carried out to test the behavior of the Web service clustering and retrieval algorithm under different settings for the algorithm's parameters. As the total number of Web services in the considered set is 737, the parameters presented in Table 1 were theoretically defined and maintained constant throughout the performed experiments.

Table 1: Clustering parameters kept constant in the presented experiments.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grid width</td>
<td>120</td>
</tr>
<tr>
<td>Grid height</td>
<td>120</td>
</tr>
<tr>
<td>Number of ant agents</td>
<td>10</td>
</tr>
<tr>
<td>Agent memory size</td>
<td>10</td>
</tr>
</tbody>
</table>

The clustering parameters with the highest level of impact on the final results were found to be the number of iterations of the clustering algorithm and the cluster retrieval distance. The first parameter influences the final positions of the services on the grid, while the second one is crucial in determining how these positions are interpreted when performing the cluster retrieval operation. To be more precise, the clustering algorithm will stop after a given number of iterations. The cluster retrieval algorithm will stop once the distance on the grid between any pair of clusters is greater or equal to the cluster retrieval distance. The theoretical recommended number of iterations for the clustering algorithm is 2000 X NumberOfServices. Given that a set of 737 service descriptions has been used, a recommended number of iterations would be around 1.5 million. Also, the recommended value for the cluster retrieval distance is 5. The theoretical minimum number of clustering iterations allowable regardless of the data set on which clustering is performed is 1 million.

Overall, the results obtained for 1 million, 1.5 million and 2 million iterations are within the same ranges, showing that as long as the iterations count is kept above the theoretical minimum, the variations in the results are not very significant. For comparison purposes, the graphical results obtained for 1 million iterations and 1.5 million iterations are presented in Figures 2(a) and 2(b). The points in the images represent Web services. They are colored differently according to the cluster they belong to, and the clusters are also outlined with red rectangles. On the other hand, the cluster retrieval distance has a significant impact on the final results. A value of 3 for this parameter leads to the creation of very restrictive small clusters with an average size of 2, a small intra-cluster variance and a high average intra-cluster similarity.
Figure 2: Comparison of the clusters obtained for a CRD of 5 and a number of iterations of the clustering algorithm of 1 million, and 1.5 million respectively. (a) presents the results for 1 million iterations and (b) for 1.5 million iterations.

These clusters are hardly usable, as there are very similar services placed in different clusters. In contrast, a value of 6 for this parameter leads to large clusters in which the services grouped together are less similar, as shown by the high intra-cluster variance, high average cluster size and low average intra-cluster similarity. The graphical format of the clusters obtained for a 1.5 million iterations of the clustering algorithm and cluster retrieval distances ranging between 3 and 6 is presented in Figures 3(a) through 3(d).

Figure 3: Comparison of the clusters obtained for 1.5 million iterations of the clustering algorithm and various values of CRD. (a) results for CRD = 3, (b) results for CRD = 4, (c) results for CRD = 5 and (d) results for CRD = 6.

The two usable values for the cluster retrieval distance that were derived from these experiments are 4 and 5. Choosing between them depends on the purpose for which the clustering operation is performed. If smaller and more coherent clusters are desired, the value of 4 is recommended. If larger and somewhat more diverse clusters are needed, a value of 5 is best suited for this parameter. However, both of these values perform well in terms of intra-cluster variance and average intra-cluster similarity.

6.2.2 Results Evaluation for the Ant-based Selection Method

To evaluate our approach on a concrete composition, we have chosen a scenario from the medical care domain. This scenario refers to a typical request from a person (identified by name, address, country), which has certain symptoms and wants to go to a particular doctor. As a result, the person will be assigned to a hospital room (indicated by city, hospital name and room number) at a certain date. In this subsection we show the way the proposed composition technique behaves under this scenario, and then we evaluate the obtained results. For the considered scenario the user request is presented in Table 2.

Table 2: User request for the medical care scenario.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>PersonName</td>
<td>City</td>
</tr>
<tr>
<td>Address</td>
<td>Hospital</td>
</tr>
<tr>
<td>Country</td>
<td>Room</td>
</tr>
<tr>
<td>Symptom</td>
<td>TransportNumber</td>
</tr>
<tr>
<td>Physician</td>
<td>DateTime</td>
</tr>
</tbody>
</table>

In our experiment we have set the number of solutions to 5, the number of ants to 4 and the number of iterations in which no change has occurred in the best compositions list until we stop to 3. We also set all the weights used to calculate the score of a solution to 1. This way, we give the same importance to the QoS score and the semantic matching score. Also, within the QoS score computation, all the attributes count the same.

In Table 3 we present the discovered services involved in the composition scenario.

Out of the 1200 possible composition solutions, the selection algorithm processes only about 90 solutions until it reaches the optimal one.

In Figure 4 we present the top 5 composition solutions returned by the selection algorithm. The first solution is the optimal one. We can see that some of these solutions have a similar structure of the composition graph, but the services instantiated in each node differ from one another. Also, it is worth noting the importance of the weights given to various components of the score. The second
solution in these results actually has a better overall matching between the concepts. However, the nodes in the first solution have a higher overall QoS score. If the weights assigned to these components would have been even a little different, the order of the first two solution would have been another. The user should choose the values for the weights carefully, as they will have a big influence of the results he will receive.

Table 3: The discovered services for the medical care scenario.

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Cluster ID</th>
<th>Inputs</th>
<th>Outputs</th>
<th>QoS Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>5</td>
<td>Symptom; Physician</td>
<td>Diagnosis</td>
<td>Running time = 314; Availability = 59; Throughput = 4; Reliability = 67</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>Symptom; Physician</td>
<td>Disease</td>
<td>Running time = 1569; Availability = 69; Throughput = 12; Reliability = 60</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>City; Hospital; Room; DateTime</td>
<td>TransportNumber</td>
<td>Running time = 1367; Availability = 74; Throughput = 6; Reliability = 69</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Diagnosis</td>
<td>TaxedPrice; CostAndHealingPlan</td>
<td>Running time = 1137; Availability = 76; Throughput = 12; Reliability = 70</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>Diagnosis</td>
<td>TaxedPriceInEuro; CostAndHealingPlan</td>
<td>Running time = 781; Availability = 86; Throughput = 6; Reliability = 65</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>Disease</td>
<td>TaxedPrice; CostAndHealingPlan</td>
<td>Running time = 759; Availability = 50; Throughput = 12; Reliability = 70</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>Disease</td>
<td>TaxedPriceInEuro; CostAndHealingPlan</td>
<td>Running time = 307; Availability = 58; Throughput = 10; Reliability = 65</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>Pacient</td>
<td>HealthInsuranceNumber; CommercialOrganization</td>
<td>Running time =1169; Availability = 65; Throughput = 6; Reliability = 68</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>Pacient</td>
<td>HealthInsuranceNumber; InsuranceCompany</td>
<td>Running time = 1423; Availability = 84; Throughput = 2; Reliability = 62</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>Pacient</td>
<td>HealthInsuranceNumber; Organisation</td>
<td>Running time = 309; Availability = 56; Throughput = 12; Reliability = 60</td>
</tr>
<tr>
<td>16</td>
<td>3</td>
<td>Pacient</td>
<td>HealthInsuranceNumber; PublicOrganisation</td>
<td>Running time = 220; Availability = 92; Throughput = 10; Reliability = 75</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>PersonName; Address; Country</td>
<td>Pacient</td>
<td>Running time = 1651; Availability = 67; Throughput = 12; Reliability = 61</td>
</tr>
<tr>
<td>33</td>
<td>7</td>
<td>Pacient; Diagnosis</td>
<td>HealthInsuranceNumber; CommercialOrganization; TaxedPrice; CostAndHealingPlan</td>
<td>Running time = 984; Availability = 71; Throughput = 10; Reliability = 60</td>
</tr>
<tr>
<td>35</td>
<td>7</td>
<td>Pacient; Diagnosis</td>
<td>HealthInsuranceNumber; InsuranceCompany; TaxedPrice; CostAndHealingPlan</td>
<td>Running time = 1490; Availability = 87; Throughput = 4; Reliability = 68</td>
</tr>
<tr>
<td>26</td>
<td>6</td>
<td>HealthInsurance Number; InsuranceCompany; TaxedPrice; CostAndHealingPlan</td>
<td>City; Hospital; Room; DateTime</td>
<td>Running time = 1556; Availability = 58; Throughput = 11; Reliability = 70</td>
</tr>
<tr>
<td>28</td>
<td>6</td>
<td>HealthInsurance Number; InsuranceCompany; TaxedPriceInEuro; CostAndHealingPlan</td>
<td>City; Hospital; Room; DateTime</td>
<td>Running time = 178; Availability = 84; Throughput = 4; Reliability = 73</td>
</tr>
</tbody>
</table>
CONCLUSIONS AND FUTURE WORK

This paper presented two ant-inspired methods applied in two phases of the automatic Web service composition process, namely the discovery and the selection of the optimal composition phases. To improve the efficiency and accuracy of services discovery process we have organized the set of available services into service clusters. Our clustering method was inspired by the way ants cluster dead corpses and build cemeteries. The clustering method is based on a set of metrics for evaluating the similarity/dissimilarity level between two services by considering their semantic descriptions. In the case of composition, our approach combines a service composition graph model with the ant colony optimization metaheuristic to find the optimal composition solution. The experimental results obtained by testing our ant-based methods on the SAWSDL-TC benchmark collection of services validate our approach.

REFERENCES