Local Consistency Reinforcement for Enhancing Web Service Composition

Ahad AlQabasani[1], Ahlem Ben Hassine[1], Arif Bramantoro[2,*] and Asma AlMunchi[3]

[1]*CS-AI Department, College of Computer, Science and Engineering, University of Jeddah, Saudi Arabia*
[2]*School of Computing and Informatics, Universiti Teknologi Brunei, Bandar Seri Begawan, Brunei Darussalam*
[3]*Cyber-Security Department, College of Computer, Science and Engineering, University of Jeddah, Saudi Arabia*
E-mail: aalqabasani.stu@uj.edu.sa; abinhusain@uj.edu.sa; arif.bramantoro@utb.edu.bn; ammunshi@uj.edu.sa
[*]*Corresponding Author*

Received 26 May 2021; Accepted 26 December 2021;
Publication 12 April 2022

Abstract

Composing Web services to fulfill a user query remains one of the most challenging research problems due to its importance to the world economy. Several composition techniques have been proposed, but these techniques are becoming more and more expensive due to the tremendous growth of data over the internet, the intensive use of services, and the continuous changes of available services. The workflow-based Web service composition problem is an NP-complete problem, due mainly to the property of the workflow structuring the composition and the diversity of underlying constraints. Reinforcing local consistency is one of the well-known pre-processing techniques to reduce the complexity of most NP-complete problems. These techniques are mainly dedicated to binary constraint satisfaction problems. Therefore, only few researchers devoted their efforts toward reinforcing some level of

Journal of Web Engineering, Vol. 21_4, 989–1016.
doi: 10.13052/jwe1540-9589.2142

local consistency on Web service composition problems that can be semantically defined in the Ontology Web Language for Web Services (OWL-S) control structure based constraints. The main goal of this paper is to propose a novel approach for reinforcing a reasonable level of local consistency, node and arc-consistency on any Web service composition problems. It is expected to provide a more up-to-date and realistic representation of any Web service composition problems without any user interactions, but with the ability to cope with any types of user's query, such as incomplete, ambiguous, and others. This is due to the existence of several novel rules for reinforcing node and arc-consistency on all types of control structure constraints. Experiments were performed on random problems of different levels of complexity to evaluate the performance of the proposed approach.

Keywords: Arc-consistency, constraint problems, NP-complete, web service composition, ontology web language for web services.

1 Introduction

As a result of the explosion of data over the internet, the popularity of services over the Web, and dynamic changes of available services over any platforms; the use of Web services becomes a central factor in human civilization. A Web service is a Web-accessible entity that authorizes reaching various services through WWW to satisfy the diversity of consumer interests and needs. Web services can be generally classified into two types: simple Web services and composite Web service [1]. Simple Web service is a Web service that has no dependency on other Web services to accomplish a consumer request, whereas composite Web service refers to the clustering and orchestration of the functionalities of different Web services to deliver what the user request.

Recently, a simple Web service is not enough to fulfill every users' queries. Several applications that provide several types of Web Service Composition (WSC) solutions were proposed. Nevertheless, these solutions are resource-consuming especially when dealing with complex and incomplete queries. In this research, we are interested in improving the WSC process and optimizing required resources. Our main goal is to propose a solution as a pre-processing step for the composing process that enhances the complexity of the WSC problem by reducing the consumption of resources in terms of CPU time and memory. Our idea is to enforce several optimal levels of consistency on the original composing problem in order to reduce the number of inconsistencies. Enforcing consistency is a known approach in

the Constraint Satisfaction Problem (CSP) community that is often used for complex real problems. However, direct application of this approach on Web service is not straightforward due to the different semantic and direction of possible relationships between Web services. These relationships can be formalized as one of the main feature of services orchestration [2]. It is how these services are combined together by using control structures of the latest open standards of Ontology Web language for Web Services (OWL-S) [3]. OWL-S control structures are defined as follows [3, 4]:

- Sequence
  The tasks of Sequence are described as a list to be executed in order without any predefined conditions.
- Split
  The components of a Split process are a bag of tasks to be executed concurrently. The split process completes as soon as all of its components are executed.
- Split+Join
  The process consists of concurrent and synchronized execution of a bunch of tasks. A Split+Join control structure completes when all of its components have been executed. Using two control structures of Split and Split+Join, it is possible to design a partial synchronization by splitting and joining few tasks.
- Any-Order
  It allows a bag of tasks to be executed asynchronously and in an unspecified order. This control structure requires the execution and completion of all its components. The execution of processes in Any-Order construct cannot overlap, but their completion is required.
- Choice
  It calls for the execution of a single task from a given bag of tasks. Any of the given tasks may be chosen for execution.
- If-Then-Else
  The If-Then-Else is a control construct that has properties of if condition, then, and else holding different aspects of the If-Then-Else. Its semantics is intended as ”Test If-condition; if True do Then if False do Else.”
- Iterate
  The Iterate construct makes no assumption on how many iterations are being made for a task, and when to initiate, terminate, or resume it. The initiation, termination and maintenance condition can be specified with *whileCondition* or *untilCondition*.

The main contributions of this research are *i*) to enhance the representation of any WSC problems that can cope with any types of real user's query, such as incomplete, ambiguous, and others; without any interactions with the user and with the idea of extracting missing information in incomplete user query based on existing Web resources, to ensure that all underlying tasks added to the initial problem are solvable; *ii*) to propose novel rules for enforcing a reasonable level of consistency on any WSC problems since the constraint graph of this problem are non-binary and not affordable to transform into binary due to the semantic nature of the OWL-S links; *iii*) to propose a new agent-based problem-solving approach based on the rules for solving general real-world WSC problems.

2 Related Works

Many researchers tackled the challenge of formalizing and solving Web services composition problems to fulfill a user query, such as in [5]. An automatic WSC can reduce unexpected failures by minimizing human intervention with automatic generation of composite services process as grounded in [6]. In [7], the authors introduced two algorithms using a service dependency graph, and a services combination through I/O dependencies. The solution starts from the evaluation of whether a certain request made by a consumer can be established or not. As a filtering process, it calculates the minimum cost of the search result of the produced services. These processes produce all paths being traversed, even for the invalid ones or those that lead to a different targeted result.

In [8], a multigraph representation for the recommendation system is introduced. This system follows an automatic service composition by interacting with user preferences and service behavior achieved in the previously processed service's history. In [9], the authors proposed an artificial bee colony algorithm for WSC. It is interesting to note the algorithm is considered as web service selection problem instead of WSC, although several decomposition graphs are presented. The main idea of this algorithm is to employ the nature of bee colony to find the best services amongst other services as neighboring nodes. However, it remains unclear how to perform an overall WSC based on the outcomes of this algorithm. Another type of nature based algorithm was discussed in [10] that proposes to use ant colony system to solve cloud based services composition. Multi agents are collaborated to perform various actions and integrated with ant colony method. In terms of Quality of Service (QoS), the execution time of the

algorithm reaches 30 seconds and the fitness value reaches 39.37. These three papers infer that a graph search method must process all nodes of services by an iteration of exploration process to find a better solution. However, the more solutions the algorithm to search for, the more time and memory it consumes. In addition, the aforementioned solutions lack of integrity in terms of discarding unneeded solutions that remain processed into the final process.

The authors in [11] aim to solve the problem of dynamic changes in user functional requirements that might appear during the execution. The four stages of the automatic WSC process, which includes planning, discovery, selection, and execution; are also discussed in detail. They implement the workflow by using a hierarchical task network planner developed based on the user specification as an input, the desired output, and the constraint as a requirement. The planner can be transformed to an OWL-S with the technique of accuracy check. We incorporate a similar technique to define a complementary concrete web service.

Semantic similarity measurement is one of the techniques used in WSC. To accommodate the semantic similarity, an integrated development environment system for composite services engineering is proposed in [12]. The system requests a user to fully clarify the query together with a main input as well as the preferences. However, if a user is unable to fulfill the query parameters due to an overhead in the process or simply by mistake, it results in an incomplete query, which eventually produces an inadequate outcome. The system requires a clear interpretation of services' semantics and characteristics, because the composition procedures strongly depend on the similarity between web services' semantics.

More and more composition approaches were proposed in the literature due to the importance of this problem. Nevertheless, most of them tried to find the optimal solution. However, finding an optimal solution consumes a considerable amount of time and memory, especially when the size of the problem increases. The aforementioned approaches lack integrity in terms of discarding unneeded sub-solutions that may appear in the final process.

Many researchers performed several investigations to improve the solving process of complex problems as a pre-processing step, known as local consistency reinforcement techniques. These techniques have gain more attention through decades and, consequently, several improvements are introduced to each reinforced consistency level [13]. Arc-consistency is the most common filtering technique. Several extensions were proposed in [14, 15], where the new versions of arc-consistency above AC-3 have been introduced to enhance

worst-case complexity, such as AC-4, AC-6, and AC-7; but ultimately they are harder to implement.

In [16], authors redefined another level of consistency, which is circuit consistency. The proposed technique is d-Dynamic Circuit Consistency, where d is the order of different constraints. This level is different from the arc-consistency. It is based on an oriented graph of constraints. Despite the importance of these techniques, only few efforts were devoted towards involving several levels of consistency as a pre-processing step for enhancing the solving process of any WSC problems. The main reason is that the constraint graph in the WSC problem is on the one hand a non-binary graph and on the other hard oriented according to the control structure constraints, therefore, it is difficult to transform into a binary one due to the semantic characteristics of the used OWL-S links.

In [17], the authors propose a new approach based on two local consistency reinforcement methods, i.e. node and arc-consistency. Their idea is to start by directly removing the values that do not satisfy unary constraint and comprise the least worthy of QoS. It is followed by pruning the values of the inconsistent arc. In this work, the authors did not consider all types of OWL-S control structures that may exist in the constraint graph. Moreover, there is no effect on the reinforced consistency level. Although the objective is to keep only the highest service quality in the skyline set, the authors did not consider different types of (non-binary) constraints that can exist between different tasks. In [18], the authors propose an orchestration of complex tasks with the goal of reaching an optimal representation to solve any WSC problems. In their proposed CSP-based formalization, they rely on the interaction of the user in order to deal with incomplete queries. However, none of the levels of local consistency is considered.

Table 1 compares our proposed approach to the aforementioned existing techniques in terms of the workflow pattern, the use OWL-S to employ

Table 1 Previous works on WSC comparison

Work	Semantic Use	Flow Pattern	OWL-S	Local Consistency
[19]	QoS	Sequence, AND, XOR	N/A	N/A
[11]	QoS	Sequence, Choice	OWL-S Semantics	N/A
[12]	Semantic similarity	Graph transformation	OWL-S Semantics	N/A
[17]	QoS	N/A	N/A	Considered
Ours	Semantic similarity	Sequence, Split, Join	OWL-S control structures	Considered

its control structure on the composition, and the accommodation of local consistency.

3 Consistency Reinforcement for WSC

In this section, we describe our novel formalization of any WSC problems followed by the description of the proposed rules to reduce their complexity before starting the solving process.

3.1 New Formalisation for WSC Problem

The proposed formalisation is a novel one in which *i*) we deal with any type of user query, such as ambiguity, incompleteness, and others; *ii*) we rely on the recommendation given to each existing Web services, instead of the QoS that is not always available and the enforcement of QoS increases the complexity as well as the execution time as shown in [19]; *iii*) we increment the current problem by using the available Web resources rather than interacting with the users or using the existing workflows which may not be available due to the lack of public standards on their execution language. Hence, the extraction of missing information in incomplete user query is performed based on existing Web resources, to be sure that all underlying tasks are solvable. Note that the information related to the hidden tasks can be exploited to make the query feasible. In the WSC problem, whatever the query given by the user, we have to find the ”best” composite Web service that fulfills this query. There is no way to answer it without providing a solution.

The WSC problem can be defined by a set of Distributed Dynamic Constraint Optimisation Problems (DDCOP) (X, D, C, f), as follows:

- $X = \{X_1, X_2, \ldots, X_n\}$ with n is the number of initial ambigous/ incomplete tasks, that we call abstract Web services, in the user query. Each X_i = ($\{X_i.in\}$, $\{X_i.out\}$, $\{X_i.pre\}$, $\{X_i.post\}$). Note that for the initial task X_1 (resp. the final task X_n), the set of inputs and preconditions (resp. outputs and postconditions) are given by the user. This set of tasks can be incremented according to the type of the task.
- $D = \{D_1, D_2, \ldots, D_m\}$ with m is the number of initial available concrete Web services for each X_i, whose service profile descriptions semantically match the task specification [20]. The input of the Web service (resp. the output) is semantically included in the input of the task (resp. output of the Web service). For each concrete Web service $s_{k_j} \in D_k$, we assign a degree of recommendation, $w_{k_j} \in [0, 1]$. This

value can be determined using a machine learning based recommender system.

- $C = C^H \cup C^S$, two types of constraints, *hard* and *soft* constraints. The C^H defines the constraints between tasks, whereas C^S defines all the constraints related to the preferences of the users, for example user preferences on the language of several Web services. For each constraint $C_j \in C^S$, we assign a *penalty* $\psi_j \in [0, 1]$ that reflects the *loss for not satisfying* the constraint with $\sum_{j \in [1..|C^S|]} \psi_j = 1$. If C_j is a *hard* constraint then $\psi_j = 1$. Sometimes the preferences of a user are conflicting, which means that it cannot be all satisfied. Therefore, we try to minimize the less unsatisfied preferences.
- $f(sl)$ is the objective function to optimize as given in (1), where *sl* is a solution of the problem defined by instantiating all variables of the problem with concrete Web services. $f(sl)$ is defined as the summation of the degree of recommendation of all involved concrete Web services in the obtained solution (*sl*) and the importance of all unsatisfied soft constraints by executing *sl*.

$$f(sl) = \alpha \times Rec(sl) - \beta \times Penality(sl) \tag{1}$$

with

$$Rec(sl) = \sum_{s_j \in sl} w_j \tag{2}$$

and

$$Penalty(sl) = \sum_{C_k \in C^S} \psi_k \tag{3}$$

with α and $\beta \in [0, 1]$ as an adjustable weight depending on the service domain. For example, if the user request deals with planning a trip overseas, the weight associated to the user's preferences (i.e. β) should be greater than the one associated to the recommendation (i.e. α), because the user usually prefer a service with less service recommendation.

For our DDCOP, the initial set of abstract Web services and their domains is distributed among a set of Task agents. Each agent initially maintains only one task and tries to find a solution for it. The use of multi-agent system can be indicated by the fact that for an abstract workflow, several tasks can be performed independently and in parallel, in order to reduce the computational complexity. Therefore, assigning an intelligent software compound that is able to act autonomously and independently for taking a decision enhances

the global solving process and consequently provides the result in a faster manner.

Any agents can update its set of task(s) using available resources, which means that each agent maintains a part of the initial CSP and increments it through subtasks or constraints if needed. This is the characteristic of dynamic CSP. The links between Task agents are defined by the OWL-S control structure according to the initial workflow that corresponds to the user query. Each agent tries to find the best solution for its task(s) as defined in (1) as a local goal, while all agents communicate to find the best solution sl^* that satisfies all hard constraints and maximizes the global goal as given in (4).

$$f(sl^*) = \arg\max_{sl} f(sl) \tag{4}$$

Solving this DDCOP problem involves two main steps:

1. The first step is a preprocessing one, in which all Task agents cooperate to reduce the size of the initial problem by reinforcing the most reasonable level of consistency, i.e. node and arc-consistency, on each type of the OWL-S constraints based on the rules proposed in this paper. If the domain maintained by any of the Task agent becomes empty, the agent tries to change its underlying task by another set of tasks, using its available resources, if possible. Otherwise, a deeper investigation to replace this task is performed.
2. The second step is the solving one, in which all agents communicate to find a "good" solution for the problem according to (4).

3.2 Consistency Reinforcement Rules for WSC Problem

The WSC problem differs from other CSP problems, since the constraints in its corresponding constraint graph are *i*) naturally oriented, *ii*) not applicable to map to binary constraints, i.e. most constraints involve only one variable from one side and several variables from the other side, and *iii*) extracted from OWL-S control structures and hence it is subject to the semantics of these control structures.

Applying existing techniques for reinforcing a certain level of consistency is not possible, since most of these techniques are dedicated for binary oriented constraints. Therefore, we need to propose new rules for enforcing consistency according to each OWL-S control structure based constraint. Consistency reinforcement techniques are mainly used with NP-complete problems in order to reduce their search space and consequently their complexity.

The reinforcement of local consistency filters the possible existing Web services and, therefore, reduces the complexity of the underlying problem. Hence, incorporating the most used level of our rules increases the optimality of the result. The higher the consistency level applies, the more constraints the variables (available Web services) will have and the more services will be reduced. Composing Web services is a complex task. Consequently, a direct reinforcement of existing algorithms is not possible, since the composition of the services to the customer must follow the scenario or the workflow of executing several Web services. We refer this process to service orchestration, which is achieved by using control structures of the open standards OWL-S.

We propose the concept of complement between two Web services that is defined as follows.

Definition 1 *A concrete Web service* s_{j_l} *is a* **complement** *of another concrete Web service* s_{i_k}*, i.e.* **complement**$(\mathrm{s}_{i_k}, \mathrm{s}_{j_l})$*, if and only if* s_{j_l}*.in* $\subseteq \mathrm{s}_{i_k}$*.out.*

Definition 2 *An abstract Web service* X_i *is* **node consistent** *(NC), if and only if all concrete Web services* s_{i_k} *in its domain D(*X_i*) can perform the task specified by* X_i*.*

In order to invoke a concrete Web service s_{i_k}, we need to provide all the required inputs for this Web service. For *i*=1, these inputs are given by the user, otherwise they are provided by the previous invoked Web services.

Rule 1. An abstract Web service X_i is NC, if and only if for all $s_{i_k} \in \mathrm{D}(X_i)$, s_{i_k}.in $\subseteq \{X_1.\text{in} \cup X_j.\text{out}$ for all X_j that precedes X_i and $i \neq 1\}$. Note that X_1.in defines all the initial information (inputs) provided by the user in the query. These information can be used in each task of the abstract workflow.

In order to reinforce NC for each abstract Web service X_i, we should remove all s_{i_k} from D(X_i) that does not satisfy $Rule$ 1. It means that we should remove all s_{i_k} from D(X_i), where s_{i_k}.in are not provided by previous X_j.

Arc-consistency is the most used level of consistency due to its complexity with regards to the performed reduction. Arc-consistency can be applied only on binary constraints. Applying arc-consistency on the WSC problem cannot be performed automatically due to the nature of the underlying constraint graph, i.e constraints are not binary and cannot be transformed easily into binary ones. Therefore, we propose several rules for enforcing arc-consistency on any WSC problems according to the type of OWL-S link.

Definition 3 *An abstract Web service* X_i *is* ***arc consistent*** *if and only if, D(*X_i*)* $\neq \emptyset$ *and each concrete Web service* s_{i_k} *in the domain of D(*X_i*) has*

a complement s_{j_l} ***in the domain of all*** X_j *with* i≠j*, related to* X_i *by one of the OWL-S control structure.*

In the following, the proposed arc-consistency rules according to the OWL-S control structures are briefly described. Note that only major OWL-S control structures are considered in this paper due to the efficiency of the discussion.

Sequence structure is a binary oriented constraint between two abstract Web services. Checking arc-consistency for this constraint should be done from both sides using the following proposed rule.

Rule 2. An abstract Web service X_i is arc consistent if and only if X_i is arc consistent with all X_j related to it by Sequence control structure, i.e. *link*(X_i, X_j)=*Sequence*. X_i is arc consistent with X_j if and only if *i*) each Web services s_{i_k} in the domain of X_i have at least one complement in the domain of X_j and *ii*) each Web service s_{j_l} in the domain of X_j has served as a complement to a Web service in X_i.

Formally, an X_i is arc-consistent *iff* X_j with *link*(X_i, X_j)=*Sequence*, *i*), $\forall$ $s_{i_k} \in D(X_i)$, $\exists$ $s_{j_l} \in D(X_j)$ such that s_{j_l}.*input* $\subseteq$ s_{i_k}.*output*, and *ii*) $\forall$ $s_{j_l} \in D(X_j)$, $\exists$ $s_{i_k} \in D(X_i)$, s_{j_l} served as a complement for s_{i_k}.

To reinforce arc-consistency on the WSC problem, we should detect all the sequence links (X_i, X_j) and remove all concrete Web services that do not satisfy *Rule 2*. Figure 1 illustrates an arc-consistency reinforcement between two tasks. Note that if a domain of any abstract Web service becomes empty, the problem is inconsistent, i.e there is no composite Web service that can satisfy the user query.

Split structure is represented in Figure 2. Since it is an oriented constraint that involves more than two abstract Web services on two sides of the link: one at the beginning of the oriented link, and two or more at the end of the link. Hence, checking consistency for these two sides should be established

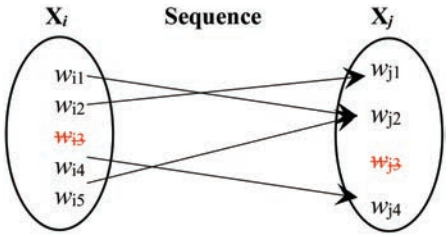

Figure 1 Reinforcing arc-consistency on sequence structure.

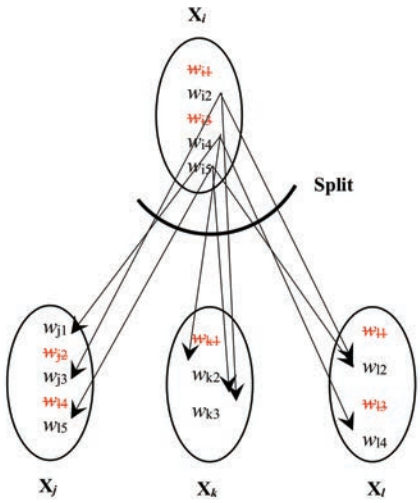

Figure 2 Reinforcing arc-consistency on split structure.

differently. We use the pair (A, B), where A is a unique abstract Web service and the set B is the set of all abstract Web services as a result of the split link as follows.

Rule 3. For each pair of abstract Web services $(X_i,\{X_j\})$ where $\{X_j\}$ is the set of all abstract Web services involved in the split control structure, i.e. *link*$(X_i, \{X_j\})$=*Split*; X_i is arc-consistent if and only if *i*) each $s_{i_k} \in \mathrm{D}(X_i)$, s_{i_k} has at least one complement in the domain of each task in $\{X_j\}$, and *ii*) for each task $X_j \in \{X_j\}$, for each $s_{j_l} \in \mathrm{D}(X_j)$, s_{j_l} has served as a complement for at least s_{i_k} in the $\mathrm{D}(X_i)$.

Formally, an X_i is arc-consistent with $\{X_j\}$ where *link*$(X_i, \{X_j\})$=*Split*, *iff i*) $\forall\, s_{i_k} \in \mathrm{D}(X_i), \forall\, X_j \in \{X_j\}, \exists\, s_{j_l} \in \mathrm{D}(X_j)$ such that s_{j_l} is a complement of s_{i_k}, and *ii*) $\forall\, X_j \in \{X_j\}, \forall\, s_{j_l} \in \mathrm{D}(X_j)$, s_{j_l} has served as a complement for at least one $s_{i_k} \in \mathrm{D}(X_i)$.

In order to reinforce arc-consistency on all split structures, we need to remove all concrete Web services from the domain of all abstract Web services in each pair $(X_i, \{X_j\})$ that do not satisfy *Rule 3* as illustrated in Figure 2. If one of the domains of abstract Web services becomes empty, then the problem is inconsistent.

Join structure is represented in the Figure 3. Since this constraint is two-sided oriented constraint that involves a set of tasks in one side and one task on the other side, checking consistency should be accomplished differently from both sides.

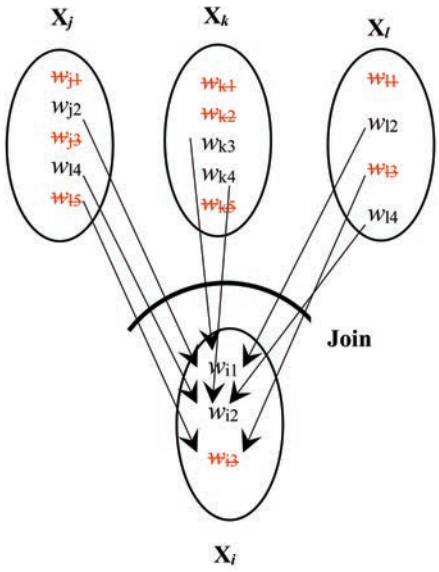

Figure 3 Reinforcing arc-consistency on join structure.

Rule 4. A pair of abstract Web services ($\{X_i\}$, X_j) where *link*($\{X_i\}$, X_j)=*Join*, the set $\{X_i\}$ is arc-consistent with X_j if and only if *i*) for each X_i in $\{X_i\}$, for each s_{i_k} in D(X_i), there exists at least one s_{m_n} in the domain of X_m in $\{X_i\}$, with $m \neq i$, such that the tuple (s_{i_k},..., s_{m_n} ,...) has at least one complement s_{j_l} in the domain of X_j and *ii*) for each s_{j_l} in the domain of X_j, s_{j_l} has served as a complement for a tuple in $\{X_i\}$.

Formally, $\forall$ ($\{X_i\}$, X_j) where *link*($\{X_i\}$, X_j)=*Join*, $\{X_i\}$ is arc-consistent with X_j *iff i*)$\forall X_i \in \{X_i\}$, $\forall s_{i_k}$ in D(X_i), $\exists s_{m_n} \in$ D(X_m), $X_m \in \{X_i\}$ and $m \neq i$, such that (s_{i_k},..., s_{m_n},...) has at least one complement $s_{j_l} \in$ D(X_j) and *ii*) $\forall$ $s_{j_l} \in$ D(X_j) $\exists$ (s_{i_k},..., s_{m_n},...) $\in$ D(X_i), such that s_{j_l} served as a complement for this tuple.

In order to reinforce arc-consistent on all existing ($\{X_i\}$, X_j) related by a Join structure, we should remove all concrete Web services from the domains of $\{X_i\}$, and the domain of X_j that do not satisfy *Rule 4* as presented in Figure 3. Any obtained empty domain yields to an inconsistent problem.

Choice structure can be considered a generalisation of if-then-else structure in the way of arc-consistency reinforcement. This structure involves one abstract Web service from one side and more than one Web services from the other side (as given by Split structure). However, arc-consistency should be considered differently form both sides (and differently from Split structure).

Rule 5. For each pair of abstract Web services (X_i, $\{X_j\}$) where $\{X_j\}$ is the set of all abstract Web services involved in Choice control structure, i.e. *link*(X_i, $\{X_j\}$)=*Choice*; X_i is arc-consistent if and only if *i*) each $s_{i_k} \in \mathrm{D}(X_i)$, s_{i_k} has at least one complement in the domain of *at least one of the tasks* in $\{X_j\}$, and *ii*) for each task X_j in $\{X_j\}$, for each $s_{j_l} \in \mathrm{D}(X_j)$, s_{j_l} has served as a complement for at least one s_{i_k} in the $\mathrm{D}(X_i)$.

Formally, an X_i is arc-consistent with $\{X_j\}$ where *link*(X_i, $\{X_j\}$)=*Choice*, *iff i*) $\forall\, s_{i_k} \in \mathrm{D}(X_i)$, $\forall\, X_j \in \{X_j\}$, $\exists\, s_{j_l} \in \mathrm{D}(X_j)$ such that s_{j_l} is a complement of s_{i_k}, and *ii*) $\forall\, X_j \in \{X_i\}$, $\forall\, s_{j_l} \in \mathrm{D}(X_j)$, s_{j_l} has served as a complement for at least one $s_{i_k} \in \mathrm{D}(X_i)$.

Reinforcing arc-consistency on all Choice structures involves removing from the domain of all abstract Web services in each pair (X_i, $\{X_j\}$) all concrete Web services that do not satisfy *Rule 5*. If one of the domains of this abstract Web services becomes empty, then the problem is inconsistent.

Our proposed approach involves two steps that are repeated until some stop conditions are satisfied:

- For each type of OWL-S constraint, use the corresponding rules to discard inconsistent concrete Web service from the corresponding domain. The domain reduction should be performed on both sides of the link. Note that, this process should be performed alternately until no more reduction is possible, e.g. removing values from one side domain may lead to more reductions on the other side.
- Propagate the performed reduction on one constraint to the next constraints. The algorithm stops if *i*) the domain of any abstract Web services X_i becomes empty $\mathrm{D}(X_i)=\emptyset$, which means that no available concrete Web service can fulfill the task, or *ii*) no more inconsistent concrete Web services can be removed from the domain of any tasks.

Figure 4 represents the architecture of our approach that includes several stages of prepossessing required to reduce the problem. Note that an agent is managing all algorithms to act as one system.

4 Illustrative Example

E-commerce service has gained huge attention in recent years since it provides an easy and fast service to the consumer. E-commerce can be defined as the process of selling physical items or services through a digital medium. Hence, the use of Web services is one of the best techniques to provide e-commerce service.

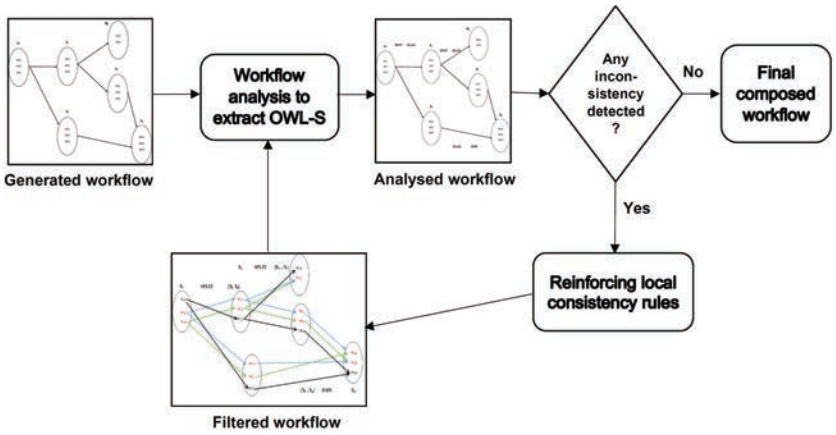

Figure 4 Proposed WSC architecture.

To illustrate the execution of the proposed approach, we introduce an example of composing e-commerce services. It clarifies how our solution (reinforcing consistency on OWL-S structures) affects in Web service composition problem in general. Assume that we have a query, i.e. attending a conference, the workflow in Figure 5 illustrates the use of OWL-S control structures for performing the following tasks: Reserve a flight, Book a hotel, Book a taxi from/to the airport, Money changing.

To respond to users' query, several e-commerce services with different areas (vehicles, housing, and banking) are filtered to produce the best services to the customer. Each task represents the composition process of Web service to its field, and each task has input and output. Having user query with {user.id, conference.date, city}, we can formalize this problem using the proposed DDCOP approach in (X, D, C, f) as follows:

- $X = \{X_1, X_2, X_3, X_4\}$ where:
  - X_1 represents the task of reserving the flight based on the conference schedule. $X_1.in$ = {user.id, conference.date, city}, $X_1.out$ = {flight.id, flight.price, flight.time, date, city, user.id}.
  - X_2 represents booking hotel at the same period of conference event. $X_2.in$ = {user.id, date, city}, $X_2.out$ = {hotel.price, booking.id, check.out.time, user.id, city}.
  - X_3 represents taxi booking, but the customer wants the taxi services two times during the stay. $X_3.in$ = {date, time, location}, $X_3.out$ = {taxi.price, booking.id}.

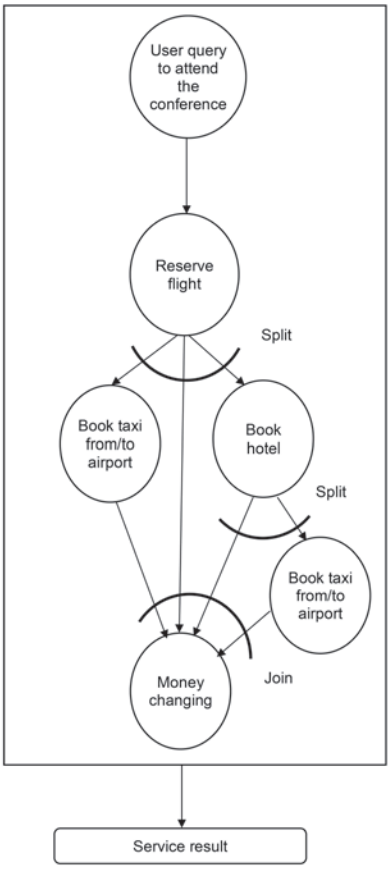

Figure 5 Attending conference service workflow in OWL-S.

– X_4 represents the final task of money changing that confirms all previous tasks. $X_4.in = \{$user.id, task.id, task.price$\}$, $X_4.out = \{$confirmation$\}$.

The enforcement of consistency is required to reach the goal of optimal e-commerce services composition, due to the priority between tasks, as well as the strong relationship between the inputs and outputs.

- $D = \{D(X_1), D(X_2), D(X_3), D(X_4)\}$ where:
  - $D(X_1)=\{$available flight services$\}$.
  - $D(X_2)=\{$available hotel services$\}$.
  - $D(X_3)=\{$available taxi services$\}$.
  - $D(X_4)=\{$available banking services$\}$.

Based on the number of available services in WWW, the execution over CPU time and memory in the composition process is exhausted [1]. The involvement of constraints in DDCOP approach provides more reasonable results to the user query. Moreover, node and arc-consistency reinforcement can reduce resource consumption.

- $C^H \cup C^S = \{C_1, C_2, C_3, C_4\}$ where:
  - C_1: $X_1.user.id \neq null$.
  - C_2: $X_1.price + X_2.price + X_3.price <=$ \$ 2000.
  - C_3: $X_2.date <$ conference.date where ($X_2.date$ = conference.date - 1), i.e. one day before conference date.
  - C_4: All services are in Arabic language.

Assume that $C^H = \{C_1, C_3\}$, and $C^S = \{C_2, C_4\}$. For each constraint $C_j \in C^S$, we assign a *penalty* for C^S is $\psi_S = \{W(C_2) = 0.7, W(C_4) = 0.3\}$ that reflects the *loss for not satisfying* the constraint with $\sum_{j \in [1..|C^S|]} \psi_j = 1$. If C_j is C^H, then ψ_j=1.

- $f(sl)$ is the objective function to optimize as given in (1). Table 2 provides the recommendation weight of each S_j (Web services available in X_i).

By assigning the values $\alpha = 0.2$ and $\beta = 0.8$, we associate a higher value to β to deliver the service that satisfies user's preferences. In the following, we present the possible combinations of solutions (SL) based on the weights' values on Table 2. Note that the column Rec in Table 3 is calculated using Equation (2), and the penalty value associated to each solution reflects the dissatisfaction of the particular composition of *sl* (SL#).

After formalizing the problem and giving the data above, we can perform our contribution on enforcing consistency algorithms on OWL-S control structures, by adding another layer of constraints on each control structure. The composition process is solved as follows:

1. Pre-processing step: filter Web services by applying the rules mentioned in section 3.2 to save more time and memory caused by expansive calculations to be performed in the next step.

Table 2 Recommendation weight to the available S_j

X_1	X_2	X_3	X_4
S_{11} (0.2)	S_{21} (0.4)	S_{31} (0.8)	S_{41} (0.9)
S_{12} (0.5)		S_{32} (1)	S_{42} (0.4)
S_{13} (0.1)			

Table 3 Values to solve $f(sl)$

SL#	Rec	ψ_j
$SL_{1a} = \{S_{11}, S_{21}, S_{31}, S_{41}\}$	2.3	0.12
$SL_{1b} = \{S_{11}, S_{21}, S_{32}, S_{41}\}$	2.5	0.03
$SL_{1c} = \{S_{11}, S_{21}, S_{31}, S_{42}\}$	1.8	0.10
$SL_{1d} = \{S_{11}, S_{21}, S_{32}, S_{42}\}$	2	0.05
$SL_{2a} = \{S_{12}, S_{21}, S_{31}, S_{41}\}$	2.6	0.08
$SL_{2b} = \{S_{12}, S_{21}, S_{32}, S_{41}\}$	2.8	0.2
$SL_{2c} = \{S_{12}, S_{21}, S_{31}, S_{42}\}$	2.1	0.18
$SL_{2d} = \{S_{12}, S_{21}, S_{32}, S_{42}\}$	2.3	0.06
$SL_{3a} = \{S_{13}, S_{21}, S_{31}, S_{41}\}$	2.2	0.01
$SL_{3b} = \{S_{13}, S_{21}, S_{32}, S_{41}\}$	2.4	0.5
$SL_{3c} = \{S_{13}, S_{21}, S_{31}, S_{42}\}$	1.7	0.02
$SL_{3d} = \{S_{13}, S_{21}, S_{32}, S_{42}\}$	1.9	0.05

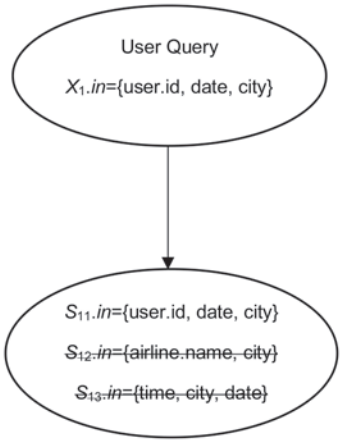

Figure 6 An example of removing Web services after arc-consistency reinforcement.

An example of this composing process is given in Figure 6. Note that the structure of the oriented link between the two tasks is sequence; so Rule 2 in 3.2 is applied to Web service. $S12$ and $S13$ are removed, due to not including all of their inputs in the user's output.

2. Solving step: choose the best option to satisfy user preferences using the optimization function defined in (1). In this step, we compute the remaining solutions in Table 3 that are only the first four rows as listed in Table 4. Therefore, SL_{1b} is the highest adequate service according to user preference. (Note that the full problem is not computed in this example to clarify its process).

Table 4 Results of $f(sl)$

SL#	$f(sl)$
SL_{1a}	0.364
SL_{1b}	0.476
SL_{1c}	0.28
SL_{1d}	0.36

Eventually, using constraints provides a less complex solution by reducing the domain of the region values before assigning the value. As a result, the complexity of searching for the values to assign the variables is reduced and, therefore, the time and memory consumption is enhanced.

5 Experimental Evaluation

The aim of the experiment is to evaluate the performance of the proposed pre-processing treatment by reinforcing node and arc-consistency on all abstract Web services, mainly when the number of available concrete Web services increases. We implemented a sequential version (using only one agent as suggested in [21]) of the proposed approach and we performed the reinforcement of node and arc-consistency (as the most reasonable and strong enough) for the most difficult and used types of OWL-S control structures, such as Sequence, Split, Join, and Choice.

We apply the proposed approach to a more complex and random cases that any processes of Web services composition occasionally experience. Note that we constructed a random generator to test our approach with a diverse data set. As pointed out by authors in [12], it is hard to generate a diverse data set because it limits the experiments on simpler cases. Hence, we developed the prototype from the scratch. It starts from the initiation of workflow parameters, such as inputs, outputs, and links); that represent the interactions between tasks in WSC problem as also developed in [22]. The environment in the implementation is Eclipse IDE 2020 with Java programming language. It runs on Windows 10 with Intel Core i5, 3.40GHz, and 4GB RAM.

For the dataset, we considered the following parameters, n is the number of abstract Web services, m is the number of concrete Web services per abstract Web services, and p is the percentage of constraints between abstract Web services in the workflow. We generated 30 abstract workflows for each combination of $<n, m, p>$ with n=$\{5, 7, 10\}$ as in the real world WSC problem, the number of tasks cannot exceed 10, m=$\{10, 30, 50, 100\}$ for

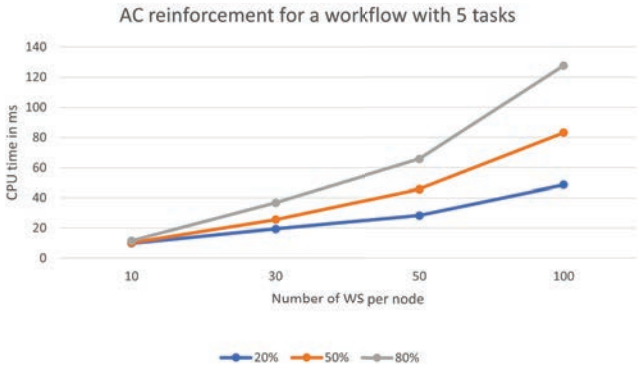

Figure 7 Results for {5, m, p} in terms of CPU time.

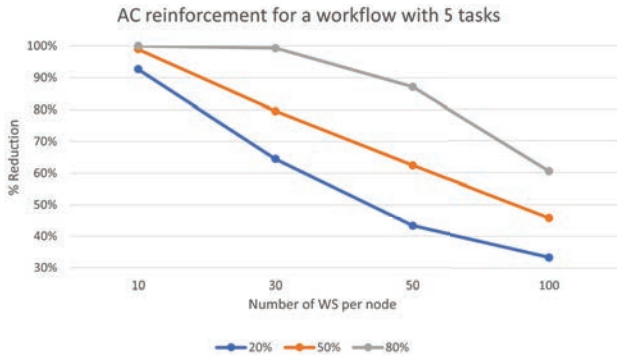

Figure 8 Results for {5, m, p} in terms of percentage of reductions.

each task with at most 100 Web services can be provided, and p={20%, 50%, 80%} to deal with easy (not highly connected workflow) and hard (highly connected workflow) problems with a total of 1080 samples. We tried these parameter options to simulate the real composition problems. The average of the obtained results were expressed in terms of *i*) percentages of reductions, and *ii*) CPU time.

The reinforcement of arc-consistency is able to detect the inconsistency of the problems without solving them for easy problems with $m<30$. When we increased the number of the concrete Web services, the CPU time is greater, less than one second as shown in Figure 7. Even for hard problems with $p>80\%$, arc-consistency is able to prune 60% of the size of the initial problem and consequently decrease its complexity as illustrated in Figure 8.

In the second part of experiments, as shown in Figure 9, we increased the number of abstract Web services, and we noticed that arc-consistency is

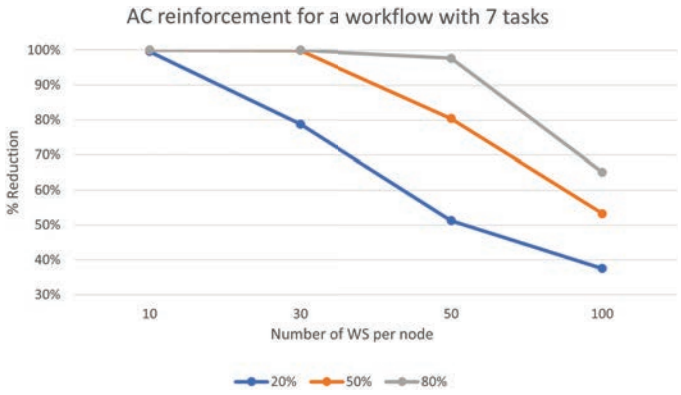

Figure 9 Results for {7, m, p} in terms of percentage of reductions.

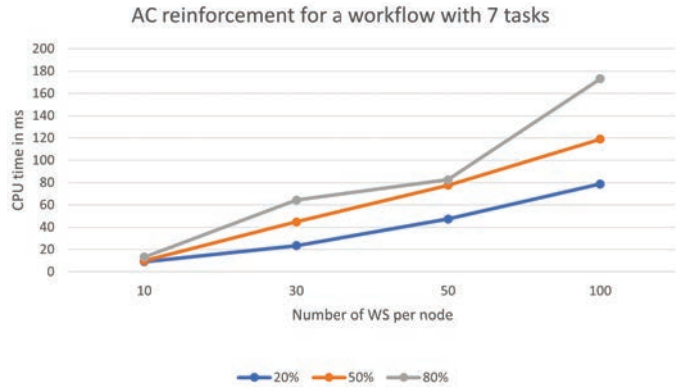

Figure 10 Results for {7, m, p} in terms of CPU time.

able to reduce with more than 60% complex problem in greater, less than one second as illustrated in Figure 10. The reduction for $p = 80\%$ is more important than for $p = 20\%$, this is due to the fact that when the number of constraints is less $(p = 20\%)$, few confusion between concrete Web services accrued and mostly no need to deduct them.

In the last part of the experiments, as shown in Figure 11, we increased the number of abstract Web services to n=10. We noticed that the arc-consistency performs better. Most problems with the number of concrete Web services m<50 are detected as inconsistent, therefore there is no solution for them for a reasonable CPU time as illustrated in Figure 12.

Based on these experimental results, it can be inferred that reinforcing arc-consistency on WSC problems is non-trivial due to its capability to

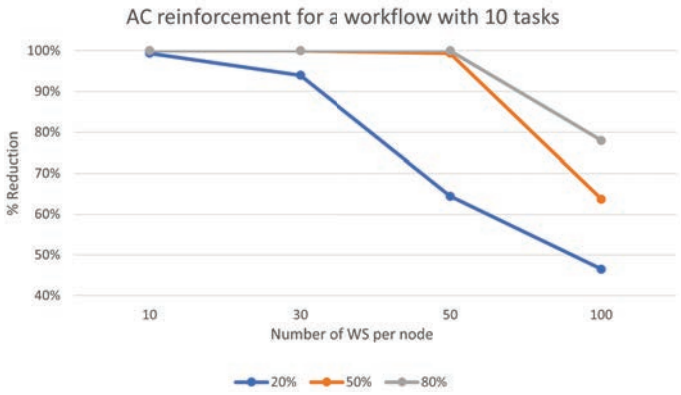

Figure 11 Results for {10, m, p} in terms of percentage of reductions.

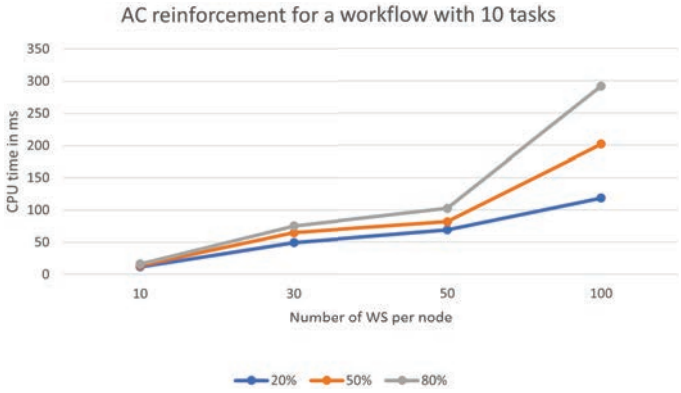

Figure 12 Results for {10, m, p} in terms of CPU time.

reduce a significant amount of time for the solving process. This level of consistency is of high utility mainly when the complexity of the underlying problem increases. Hence, the experiment supports our assumption that each consistency level reinforcement is worth for WSC and able to deduct more unreliable Web services. This is based on our findings that we obtained 70% highest percentage of reduction over inconsistent web services (beside 100% after arc-consistency reinforcement) for $p = 80\%$ after enforcing node consistency.

Table 5 summarizes the different performance impacts for both node and arc consistency levels. There is a higher reduction noted on the reinforcement of both levels of consistencies. Hence, it can be inferred that the constraints

Table 5 Reinforcement comparison result

Reinforcement Case	Lowest Reduction	Highest Reduction	Max CPU Time (ms)	Highest Inconsistency	Lowest Inconsistency
NC	15%	80%	50	m=10 n=10	m=100 n=5
AC	35%	100%	350		
NC and AC	86%	100%	250		

over the composition process increase due to the effectiveness of node consistency and subsequently the arc consistency reinforcement on the data.

6 Conclusion

In this paper, we addressed the problem of applying node and arc-consistency on any Web services composition problems. A new approach based on arc-consistency reinforcement for different OWL-S constraints were proposed. The idea is to perform arc-consistency of non-binary constraints according to their semantics. This approach allows pruning all values that cannot be part of any solutions for the problem. Several rules for arc-consistency were presented to support the semantic of OWL-S constraints. Hence, the proposed approach based on these rules could reduce the complexity of the underlying problem and made the search for the optimal solution more affordable. The experimental obtained results showed that the proposed approach is able not only to prove the inconsistency of most complex problems in advance, but also to reduce the failure possibility of workflow execution. As for future work, we intend to reinforce more levels of consistency.

References

[1] B. Medjahed and A. Bouguettaya, *Describing and Organizing Semantic Web Services*, pp. 73–99. New York, NY: Springer New York, 2011.

[2] Y. Hammal, K. S. Mansour, A. Abdelli, and L. Mokdad, "Formal techniques for consistency checking of orchestrations of semantic web services," *Journal of Computational Science*, vol. 44, pp. 1–17, 2020.

[3] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, *et al.*, "Owl-s: Semantic markup for web services," *W3C member submission*, vol. 22, no. 4, 2004.

[4] C.-H. Liu, S.-L. Chen, J. Y. Kuo, and T.-Y. Huang, "A flow graph-based test model for owl-s web services," in *2011 International Conference on Machine Learning and Cybernetics*, vol. 2, pp. 897–902, IEEE, 2011.

[5] A. B. Hassine, S. Matsubara, and T. Ishida, "A constraint-based approach to horizontal web service composition," in *International semantic Web conference*, pp. 130–143, Springer, 2006.

[6] L. Thomas and A. Immanuel, "Web service composition: A survey on the various methods used for web service composition.," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 3, pp. 665 – 670, 2017.

[7] P. Guluru and R. Niyogi, "New approaches for service composition based on graph models," in *2014 Seventh International Conference on Contemporary Computing (IC3)*, pp. 507–512, IEEE, 2014.

[8] F. Slaimi, S. Sellami, O. Boucelma, and A. B. Hassine, "A multigraph approach for web services recommendation," in *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*, (Cham), pp. 282–299, Springer International Publishing, 2016.

[9] F. Dahan, H. Mathkour, and M. Arafah, "Two-step artificial bee colony algorithm enhancement for qos-aware web service selection problem," *IEEE Access*, vol. 7, pp. 21787–21794, 2019.

[10] F. Dahan, "An effective multi-agent ant colony optimization algorithm for qos-aware cloud service composition," *IEEE Access*, vol. 9, pp. 17196–17207, 2021.

[11] U. Arul and S. Prakash, "Toward automatic web service composition based on multilevel workflow orchestration and semantic web service discovery," *International Journal of Business Information Systems*, vol. 34, no. 1, pp. 128–156, 2020.

[12] A. Abid, M. Rouached, and N. Messai, "Semantic web service composition using semantic similarity measures and formal concept analysis," *Multimedia Tools and Applications*, vol. 79, no. 9, pp. 6569–6597, 2020.

[13] K. Ghedira, *Constraint satisfaction problems: csp formalisms and techniques*. John Wiley & Sons, 2013.

[14] M. Mouhoub, "Dynamic arc consistency for csps," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 13, no. 2, pp. 45–58, 2009.

[15] S. Kong, S. Li, and M. Sioutis, "Exploring directional path-consistency for solving constraint networks," *The Computer Journal*, vol. 61, no. 9, pp. 1338–1350, 2018.

[16] A. Idrissi and A. B. Hassine, "Circuit consistencies," in *PRICAI 2004: Trends in Artificial Intelligence* (C. Zhang, H. W. Guesgen, and W.-K. Yeap, eds.), (Berlin, Heidelberg), pp. 124–133, Springer Berlin Heidelberg, 2004.

[17] H. Fekih, S. Mtibaa, and S. Bouamama, "Local-consistency web services composition approach based on harmony search," *Procedia computer science*, vol. 112, pp. 1102–1111, 2017.

[18] A. Bramantoro, A. B. Hassine, S. Matsubara, and T. Ishida, "Multilevel analysis for agent-based service composition.," *J. Web Eng.*, vol. 14, no. 1&2, pp. 63–79, 2015.

[19] V. Gabrel, M. Manouvrier, and C. Murat, "Web services composition: complexity and models," *Discrete Applied Mathematics*, vol. 196, pp. 100–114, 2015.

[20] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *International semantic web conference*, pp. 333–347, Springer, 2002.

[21] P. Philipp, M. Maleshkova, A. Rettinger, and D. Katic, "A semantic framework for sequential decision making for journal of web engineering," *Journal of Web Engineering*, vol. 16, no. 5&6, pp. 471–504, 2017.

[22] M. Abdel-Salam, W. Bahgat, E. Mohamed Eldaydamony, and A. Atwan, "A novel framework for web service composition," *International Journal of Simulation: Systems, Science and Technology*, vol. 20, p. 1, 07 2019.

Biographies

Ahad AlQabasani received her bachelor degree in computer science from the Faculty of computer and information technology, Tabuk University, Saudi Arabia, in 2018. She completed her master degree in computer science, at the College of Computer Science and Engineering, Computer Science and Artificial Intelligence Department, University of Jeddah, Saudi Arabia. She had an article accepted at International Conference on Signal, Control and Communication (SCC), published at IEEE Xplore in 2019. She also had an accepted article to be published at the International Journal of Modelling, Identification and Control (IJMIC). Her research interests include constraints problems, Web service composition, and speech recognition.

Ms. AlQabasani participated at the 3rd scientific forum, in 2018, and won the second place in the research track at University of Jeddah, Saudi Arabia.

Ahlem Ben Hassine is Ph.D. graduated in Computer Science, specialty in Artificial Intelligence from the Japan Advanced Institute of Science and Technology (JAIST) Japan, in 2005. Then, she had two years of research fellow at the Computational Linguistics Group, Language Grid project at the National Institute of Information and Communication Technology (NICT) Kyoto-Japan.

From 2007 to 2016, she was an assistant professor at the National School of Computer Science (ENSI-Tunis). Currently she is an assistant professor at the College of Computer Science and Engineering, Computer Science and Artificial Intelligence Department, University of Jeddah, Saudi Arabia. Her Research interests involve constrained problems, multi-agent systems, meta-heuristics, machine learning, renewable energy and Web Composition. She is the author of more than 30 papers in books, international journals and high ranked conferences. She was also a recipient of many grants and awards, from JAIST, C&C, NICT during her PhD and Post-doc program.

Arif Bramantoro is currently a senior assistant professor in School of Computing and Informatics, Universiti Teknologi Brunei, Brunei Darussalam. Previously, he was an associate professor in Information Systems Department, Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Saudi Arabia. From 2012 to 2016, he was an assistant professor in Information Systems Department, College of Computer Sciences and Information, Al-Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, Saudi Arabia.

He was an expert researcher at Information Services Platform Laboratory, National Institute of Information and Communication Technology (NICT), Japan, from 2011 to 2012. He received Ph.D. from Department of Social Informatics, Kyoto University, Japan, in 2011. He holds master degree from Faculty of Information Technology, Monash University, Melbourne, Australia in 2006. His bachelor degree was obtained from Department of Informatics, Institute Technology of Bandung, Indonesia in 2001.

His research interests include service system, business process workflow, and business intelligence. He is the author of more than 40 articles. He was a recipient of Research Excellence Award in 2016 from Deanship of Scientific Research, Al-Imam University, Saudi Arabia; and Best Paper Award from IEEE International Conference on Cloud Computing (ICCC) in 2015.

Asma AlMunchi is an associate professor in Cybersecurity Department in College of Computer Science and Engineering at University of Jeddah, Kingdom of Saudi Arabia. She has obtained her Ph.D. degree in Information Security from Curtin University, Australia in 2014, Master degree in Internet Security and Forensic (with distinction) in 2009 from Curtin University, and B.Sc. in Computer Science from King Abdulaziz University, Kingdom of Saudi Arabia in 2004. She is currently serving as a supervisor of Cybersecurity department–Female section in College of Computer Science and Engineering at University of Jeddah, and Vice Dean of Faculty of Computing and Information Technology-Female section at University of Jeddah-Khulais branch, Kingdom of Saudi Arabia. Her research interest includes educational technology, and information security.