Towards Fully Automated Semantic Web Service Composition Based on Estimation of Distribution Algorithm

Chen Wang[1(✉)], Hui Ma[1], Gang Chen[1], and Sven Hartmann[2]

[1] School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand
{chen.wang,hui.ma,aaron.chen}@ecs.vuw.ac.nz

[2] Department of Informatics, Clausthal University of Technology, Clausthal-Zellerfeld, Germany
sven.hartmann@tu-clausthal.de

Abstract. Web service composition has been a challenging research area, where many researchers have been working on a composition problem that optimizes Quality of service and/or Quality of semantic matchmaking of composite solutions. This NP-hard problem has been successfully handled by many Evolutionary Computation techniques with promising results. Estimation of Distribution has shown its initial promise in solving fully automated service composition, and its success strongly relies on distribution models and sampling techniques. Our recently published work proposed a Node Histogram-Based approach to fully automated service composition. However, many services presented in sampled optimized queues does not contribute to decoded solutions of the queue. Therefore, efforts should be made to focus on learning distributions of component services in solutions. Consequently, we aim to learn more suitable distributions considering services satisfying service dependency in the solutions and use the Edge Histogram Matrix to learn restricted sampled outcomes satisfying the dependency. Besides that, we proposed effective sampling techniques with high efficiency in a straightforward implementation. Our experimental evaluation using benchmark datasets shows our proposed EDA-based approach outperforms two recent approaches regarding both efficiency and effectiveness.

Keywords: Web service composition · QoS optimization
Combinatorial optimization

1 Introduction

Web services are reusable components of web applications, and can be published, discovered, and invoked on the Web, providing services to users or other software [1]. *Web service composition* aims to loosely couple web services to provide more complicated functionalities since one atomic web service does not always satisfy users' complex requirement completely. *Fully automated service composition*

T. Mitrovic et al. (Eds.): AI 2018, LNAI 11320, pp. 458–471, 2018.
https://doi.org/10.1007/978-3-030-03991-2_42

constructs a composition of services without strictly obeying any specific service workflow [7]. As the number of web service with similar functionalities has significantly increased, web service composition challenges many researchers to find composition solutions with the best overall *Quality of Service* (QoS) within polynomial-time. Apart from optimizing QoS, *Quality of Semantic Matchmaking* (QoSM) is often optimized simultaneously that creates more challenges for researchers [14].

Many Evolutionary Computation (EC) techniques have been widely used to achieve QoS-aware web service composition in a fully automated way [4,5,8,10,14–17]. Often, conventional EC techniques [4,8,15,17] rely on domain-dependent genetic operators to generate new candidate solutions. Estimation of Distribution Algorithm (EDA) is different from most conventional EC-based techniques because a probabilistic model is learned based on the distribution of superior subpopulation, and further used for sampling new candidate solutions. EDA has been widely used in many problem domains, such as portfolio management and cancer chemotherapy optimization, achieving better results compared to conventional EC-based techniques [2], and it has been used for solving semi-automated service composition, where service composition workflow is given in advance. Learning distribution over a pre-defined structure of a workflow is relatively less challenging. To support learning distributions over uncertain structures of candidate composite solutions in fully automated web service composition, our recently published work [16] proposed a Node Histogram-Based work for fully automated service composition with the aim to find composition solutions with optimized QoS and QoSM. The algorithm has been demonstrated to achieve higher effectiveness and efficiency than one PSO-based approach [14].

Despite the initial success in EDA for solving fully automated service composition problems. A more suitable distribution model over superior subpopulation needs further studies. Therefore, opportunities still exist to further investigate the potential use of other distribution models for supporting fully automated service composition and propose effective sampling algorithms to support sampling composition solutions from these distribution models.

The overall goal of this paper is to *propose an effective EDA-based approach to fully automated semantic web service composition*, where QoS and QoSM are jointly optimized. We achieve three objectives in this work.

1. To learn more suitable distributions that can naturally capture the most essential ingredients for building effective service composition solutions, we consider dependencies of components services in composite solutions and using Edge Histogram Matrix (EHM) to learn a distribution of restricted sampled outcomes satisfying the service dependencies. To achieve that, we will develop an ontology-based querying technique for efficiently querying the dependencies and a way of using EHM to learn those dependencies for service compositions.
2. To easily achieve high efficiency in a straightforward implementation, and to effectively sample candidate composition solutions of high quality and validity

from EHM directly, we will propose a guided edge histogram-based backward graph sampling algorithm.
3. To demonstrate the effectiveness of our overall EDA-based approach, we conduct experiments to compare it against two recent works [14,16] that solve the same problem in semantic web service composition.

2 The Semantic Web Service Composition Problem

We consider a *semantic web service* (*service*, for short) as a tuple $S = (I_S, O_S, QoS_S)$ where I_S is a set of service inputs that are consumed by S, O_S is a set of service outputs that are produced by S, and $QoS_S = \{t_S, c_S, r_S, a_S\}$ is a set of non-functional attributes of S. The inputs in I_S and outputs in O_S are parameters modeled through concepts in a domain-specific ontology $\mathcal{O}$. The attributes t_S, c_S, r_S, a_S refer to the response time, cost, reliability, and availability of service S, respectively, which are four commonly used QoS attributes [18].

A *service repository* $\mathcal{SR}$ is a finite collection of services supported by a common ontology $\mathcal{O}$. A *service request* (or *composition task*) over a given $\mathcal{SR}$ is a tuple $T = (I_T, O_T)$ where I_T is a set of task inputs, and O_T is a set of task outputs. The inputs in I_T and outputs in O_T are parameters that are semantically described by concepts in the ontology $\mathcal{O}$. We use two special services $Start = (\emptyset, I_T, \emptyset)$ and $End = (O_T, \emptyset, \emptyset)$ to account for the input and output requirements of a given composition task T, and add them to $\mathcal{SR}$.

A *composite service* (or *composition solution*) is represented as a directed acyclic graph (DAG). Its nodes correspond to those services in $\mathcal{SR}$ (also called *component services*) that are used in the composition, including *Start* and *End*.

In this paper, we are concerned with the *Semantic Web Service Composition Problem* where we aim to jointly optimize QoS and QoSM. In previous work [14–16] we have proposed and explored a comprehensive quality model for evaluating these quality aspects. The comprehensive quality of a composition solution can be evaluated based on a weighted sum of all quality criteria in QoS and QoSM using the fitness function in Eq. (1):

$$Fitness = w_1 \hat{MT} + w_2 \hat{SIM} + w_3 \hat{A} + w_4 \hat{R} + w_5(1 - \hat{T}) + w_6(1 - \hat{C}) \quad (1)$$

with $\sum_{k=1}^{6} w_k = 1$. This objective function aggregates the quality criteria of semantic matching type $\hat{MT}$, semantic similarity $\hat{SIM}$, availability $\hat{A}$, reliability $\hat{R}$, time $\hat{T}$, and cost $\hat{C}$. $\hat{T}$ and $\hat{C}$ are offset by 1, so that higher scores correspond to better quality. We refer to [14–16] for details on the calculation of each quality criterion. Therefore, the goal of our semantic web service composition is to maximize the objective function in Eq. (1) to find the best solution.

3 Our EDA-Based Approach for Service Composition

In this section, we introduce our EDA-based approach for fully automatic semantic web service composition. We first outline our EDA-based service composition approach in Sect. 3.1. Subsequently, we discuss three ideas behind this approach: the first one is a proposed ontology-based querying technique for querying service dependency in Sect. 3.2; the second one is an application of EHM for learning service dependency in Sect. 3.3; the third one is a proposed sampling technique for building composite solutions in Sect. 3.4.

As the success of EDA strongly relies on its distribution model, especially when the number of outcomes (i.e., component services) sampled from a distribution is huge, we aim to learn a suitable distribution model. Our recent work [16] learns the distribution of each service in $\mathcal{SR}$ at each absolute position of a service queue. However, many services presented in sampled optimized queues does not contribute to decoded solutions of the queue. Therefore, efforts should be made on learning distributions of the component services that contribute to composite solutions. Therefore, we aim to learn distributions restricted by the dependencies among component services in DAG-based solutions, and this distribution can be easily presented in EHM. To achieve that, we proposed an ontology-based querying technique for querying dependencies of services in $\mathcal{SR}$. This technique provides a set of outcomes, whose distributions are to be learned in EHM, and we will demonstrate an application of EHM by mapping DAG-based solutions and dependencies.

Furthermore, to easily achieve high efficiency in a straightforward implementation, and to sample component services satisfying services dependencies that contribute to composition solutions with high quality and validity, we proposed a Guided Edge Histogram-Based Backward Graph-Sampling Algorithm. This algorithm builds a DAG-based composition from *End* to *Start* using guided information of services dependencies, and service layers, see details in Sect. 3.4.

Algorithm 1. Our EDA-based method for service composition.

Input : composition task T, service repository $\mathcal{SR}$ and $g \leftarrow 0$
Output: an optimal composition solution $\mathcal{G}^{opt}$

discovery task-related web services and layers $\mathcal{L}_p$ (where $p = 0, \dots, q$) ;
label $\mathcal{O}$ with task-related web services using Algorithm 2;
initialize $\mathcal{P}^g$ with m valid DAG-based solutions, each solution represented as a $\mathcal{G}_k^g$ (where $k = 1, \dots, m$);
evaluate each solution in $\mathcal{P}^g$ using Eq. 1;
generate $\mathcal{EHM}^g$ from the top $\frac{1}{2}$ of best solutions in $\mathcal{P}^0$;
while $g <$ *maximum number of generations* **do**
    sample m solutions $\mathcal{G}_k^{g+1}$ sampled from $\mathcal{EHM}^g$ using Algorithm 3;
    populate $\mathcal{P}^{g+1}$ with newly sampled solutions ;
    evaluate each solution in $\mathcal{P}^{g+1}$ using Eq. 1;
    generate $\mathcal{EHM}^{g+1}$ from the top $\frac{1}{2}$ of the best solutions in $\mathcal{P}^{g+1}$;
    set $g \leftarrow g + 1$;
let $\mathcal{G}^{opt}$ be the best solution in $\mathcal{P}^g$;

3.1 Outline of Our EDA-Based Method

We outline our proposed algorithm in Algorithm 1. We start with filtering task-relevant services with respect to any specific composition task, utilizing a simple discovery algorithm from [11] to identify all relevant services and their layers $\mathcal{L}_p$ from *Start* (where $p = 0, \ldots, q$ and q is the number of layers). Basically, the first layer contains services that can be immediately executed by using I_T, and the second layer contains the remaining services that can be executed by using I_T and outputs provided by services in the previous layers. Other layers can be discovered in the similar way, see details in [11]. After that, we label $\mathcal{O}$ with task-related web services using Algorithm 2, which enables us to identify non-zero entries in EHM for setting bias, see details in Sect. 3.3. Next, we initialize a population $\mathcal{P}^0$ with m DAG-based candidate solutions by a greedy search algorithm over randomly sorted $\mathcal{SR}$ [14] for building graphs. Those candidate solutions are evaluated using Eq. 1. Then, the top half best-performing solutions are used to generate a $\mathcal{EHM}^g$ (where $g = 0$), see details in Sect. 3.3. The following steps (Step. 5 to Step. 9) will be repeated until the maximum number of generations is reached: we sample m new valid candidate solutions from $\mathcal{EHM}^g$ using our proposed Guided Edge Histogram-Based Graph-sampling Algorithm. These newly sampled candidate solutions form the next population $\mathcal{P}^{g+1}$ and will be evaluated and selected to learn $\mathcal{EHM}^{g+1}$.

In summary, we propose a way of learning EHM from high-quality solutions discovered by EDA so far and a novel sampling technique for building valid solutions from EHM.

3.2 Discovery of Service Dependency

Service dependency represents a relationship between two services (i.e., one service S_j and its predecessor S_i) that are determined by the existence of robust causal links [14] between these two services. In other words, one service can be either partially or fully satisfied by its predecessor, denoted as $S_i \rightarrow S_j$.

To identify service dependencies regarding each service, we proposed an ontology-based querying technique to efficiently find their predecessor services in $\mathcal{SR}$. We first create labels for concept nodes of a taxonomy tree in $\mathcal{O}$ with task-related services using Algorithm 2. In this Algorithm, we mark each tree node with two sets of services, i.e., O_C and I_C, where robust causal links can be ensured from services in O_C and services in I_C. We can query the predecessors of one service S by a union of O_C from concept nodes with respect to input-related concepts of S. We will demonstrate this technique in Example 1.

Example 1. Suppose we have a service repository $\mathcal{SR}$ consisting of a single service $S_0 = (\{c, d\}, \{e\}, QoS_{S_0})$. Let us consider the *service request* $T = (\{a, b\}, \{i\})$. The two special services $Start = (\emptyset, \{a, b\}, \emptyset)$ and $End = (\{i\}, \emptyset, \emptyset)$ are defined by the given composition task T. Concepts related to a, b, c, d, e, and i are Dog, Artificial Data, Data, Canine, Animal Robot and Robot respectively. These concepts are represented and labeled with services in an taxonomy tree in Fig. 1. The predecessor of *End* is S_0, which is a service in O_{Robot} of concept *Robot* related to i. The predecessor of S_0 is *Start*, which is a service in an union of O_{Data} and O_{Canine} related to c and d respectively.

Algorithm 2. Labeling services on taxonomy tree in $\mathcal{O}$

Input : $\mathcal{SR}$ and $\mathcal{O}$
Output: a labeled $\mathcal{O}$

foreach *concept* C *in taxonomy tree in* $\mathcal{O}$ **do**
  label two empty service set I_C and O_C in relation to inputs and output;
foreach S *in* $\mathcal{SR}$ **do**
  foreach I_S *of* S **do**
    find concepts C of I_S on taxonomy tree in $\mathcal{O}$;
    foreach C *in* $C \cup$ *its child concepts* **do**
      put S to I_C of C;
  foreach O_S *of* S **do**
    find concepts C of O_S on taxonomy tree in $\mathcal{O}$;
    foreach C *in* $C \cup$ *its parent concepts* **do**
      put S to O_C of C;
return labeled $\mathcal{O}$;

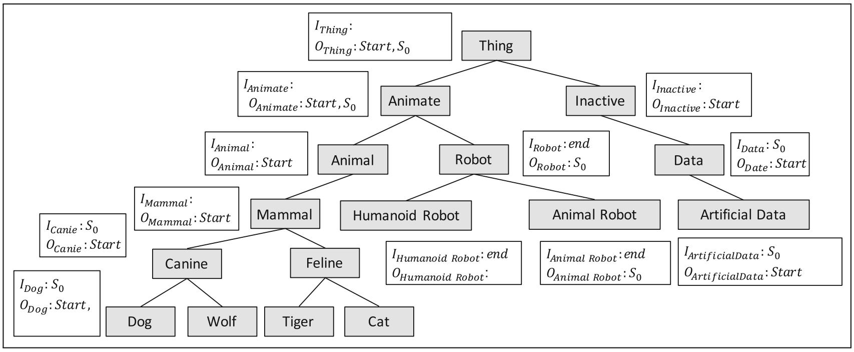


Fig. 1. An example of labeled $\mathcal{O}$

3.3 Application of Edge Histogram Matrix

Let $\mathcal{D} = \{S_i \rightarrow S_j\}$ be the set of all existing service dependencies among all possible pairs of services in $\mathcal{SR}$. Let $\mathcal{G}$ be a DAG-based composition solution consisting of a set of service dependencies, satisfying $\mathcal{G} \subset \mathcal{D}$. Consequently, $\mathcal{G}_k^g$ represents the k^{th} $(0 \leq k < m)$ DAG-based composite solution, and $\mathcal{P}^g = [\mathcal{G}_0^g, \ldots, \mathcal{G}_k^g, \ldots, \mathcal{G}_{m-1}^g]$ is represented as a population of solutions of generation g.

Example 2. Suppose we have a service repository $\mathcal{SR}$ consisting of five services $S_0 = (\{c, d\}, \{e\}, QoS_{S_0})$, $S_1 = (\{a\}, \{f, g\}, QoS_{S_1})$, $S_2 = (\{a, b\}, \{h\}, QoS_{S_2})$, $S_3 = (\{f, h\}, \{i\}, QoS_{S_3})$ and $S_4 = (\{a\}, \{f, g, h\}, QoS_{S_4})$. Let us consider the *service request* $T = (\{a, b\}, \{i\})$ as in Example 1.

The initial population $\mathcal{P}^0$ may consist of m composition solutions for T, given by their DAG-representations, such as follows (note that $m = 6$ in this example):

$$\mathcal{P}^0 = \begin{bmatrix} \mathcal{G}_0^0 \\ \mathcal{G}_1^0 \\ \mathcal{G}_2^0 \\ \mathcal{G}_3^0 \\ \mathcal{G}_4^0 \\ \mathcal{G}_5^0 \end{bmatrix} = \begin{bmatrix} \{Start \rightarrow S_1, Start \rightarrow S_2, S_1 \rightarrow S_3, S_2 \rightarrow S_3, S_3 \rightarrow End\} \\ \{Start \rightarrow S_0, S_0 \rightarrow End\} \\ \{Start \rightarrow S_0, S_0 \rightarrow End\} \\ \{Start \rightarrow S_4, S_4 \rightarrow S_3, S_3 \rightarrow End\} \\ \{Start \rightarrow S_4, S_4 \rightarrow 3, S_3 \rightarrow End\} \\ \{Start \rightarrow S_1, Start \rightarrow 2, S_1 \rightarrow S_3, S_2 \rightarrow S_3, S_3 \rightarrow End\} \end{bmatrix}$$

The *edge histogram matrix at generation g* (denoted by $\mathcal{EHM}^g$) is a matrix with entries $e^g_{i,j}$ (where $i, j = Start, 0, 1, \cdots, m-1, End$) as follows:

$$e^g_{i,j} = \begin{cases} \sum_{k=0}^{m-1} \delta_{i,j}(\mathcal{G}_k^g) + \varepsilon_{i,j} & \text{if } i \neq j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$\delta_{i,j}(\mathcal{G}_k^g) = \begin{cases} 1 & \text{if } S_i \rightarrow S_j \in \mathcal{G}_k^g \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$\varepsilon_{i,j} = \begin{cases} \frac{b_{ratio}}{|\mathcal{D}|} \sum_{k=0}^{m-1} |\mathcal{G}_k^g| & \text{if } S_i \rightarrow S_j \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Herein, b_{ratio} is a predetermined constant (called bias ratio), $|\mathcal{G}_k^g|$ denotes the number the service dependencies in $\mathcal{G}_k^g$, while $|\mathcal{D}|$ denotes the number of all service dependencies in $\mathcal{SR}$. Roughly speaking, entry $e^g_{i,j}$ counts how often service dependency $S_i \rightarrow S_j$ occurs in all composition solutions in population $\mathcal{P}^g$.

3.4 A Guided Edge Histogram-Based Backward Graph-Sampling Algorithm

The sampling algorithm is proposed based on an Edge Histogram-Based Sampling Algorithm [13]. By providing the distribution information of predecessors of each service in EHM, it is then possible to build up a composition graph from the dependencies. Some useful information is used to guide the sampling to produce only restricted outcomes, which makes this algorithm more effective: only row indexes of non-zero entries in $\mathcal{EHM}^g$ are to be sampled, and layer information is used to verify sampled predecessors for preventing cycles in solutions. This algorithm builds a DAG in a backward way. It has been suggested in [11] that backward graph building has its advantage over the forward graph building since it does not create dangling services. This sampling algorithm is summarized in Algorithm 3.

In Algorithm 3, we first initialize a DAG-based solution $\mathcal{G}$ with an empty set of service dependencies, and a set of service *SerSet*, whose inputs satisfactions required to be checked, with *End*. The following steps are repeated if *SerSet* does not only contains *Start* or any service in *SerSet* are not fully satisfied (Step.

Algorithm 3. Guided Edge Histogram-Based Backward Graph-Sampling Algorithm

Input : $\mathcal{EHM}^g$
Output: a composition solution $\mathcal{G}$
1: initial $\mathcal{G} = \{\ \}$ and $SerSet = \{End\}$;
2: **foreach** S_j *in* $SerSet$ **do**
3:   **if** $SerSet$ *does not only contains start and* S_j *is not fully satisfied* **then**
4:     identify $\mathcal{L}_p$ *s.t.* $S_j \in \mathcal{L}_p$;
5:     determine a set SC of row indexes for non-zero entries in $\{e^g_{,j}\}$;
6:     **while** *inputs of* S_j *is not fully satisfied and* SC *is not empty* **do**
7:       sample one predecessor x with probability $\frac{e^g_{x,j}}{\sum_{i \in SC} e^g_{i,j}}$;
8:       identify $\mathcal{L}_{p'}$ *s.t.* $S_x \in \mathcal{L}_{p'}$;
9:       **if** $p' \leq p$ *and any unsatisfied input of* S_j *is fulfilled by* S_x **then**
10:         put $S_x \rightarrow S_j$ into $\mathcal{G}$;
11:         **foreach** $S_{j^\star}$ *in* $SerSet$ **do**
12:           identify $\mathcal{L}_{p^\star}$ *s.t.* $S_{j^\star} \in \mathcal{L}_{p^\star}$;
13:           **if** $p' \leq p^\star$ *and any unsatisfied input of* $S_{j^\star}$ *is fulfilled by* S_x **then**
14:             put $S_x \rightarrow S_{j^\star}$ into $\mathcal{G}$;
15:         add S_x to $SerSet$;
16:       remove x from SC;
17:     remove S_j from $SerSet$;
18: **return** $\mathcal{G}$;

2 to Step. 17): for each service S_j in $SerSet$, we identify its layer $\mathcal{L}_p$. Meanwhile, we initialize a set, SC, consisting of row indexes of non-zero entries in $\{e^g_{,j}\}$. Afterward, another repeated sampling process is used to produce predecessors of S_j until S_j is fully satisfied (Step. 6 to Step. 16). During the sampling, let S_x be the corresponding service of sampled service index x, if the layer that contains S_x is ahead of or the same to that of S_j, and any unsatisfied inputs of S_j can be fulfilled by S_x (Step. 9), we create a dependency $S_x \rightarrow S_j$ and put it into $\mathcal{G}$ (Step. 10). Meanwhile, to create a more compacted DAG, we also check the satisfaction of other services in $SerSet$ in the similar way that we create the dependency with S_j (Step. 11 to Step. 14). Later on, the sampled predecessor S_x is added to $SerSet$ and sampled x is removed from SC. Once S_j is fully satisfied, we remove it from $SerSet$, and repeat creating dependencies for newly added services in $SerSet$ until the stop conditions are met (Step. 2 to Step. 17). Then, a $\mathcal{G}$ is returned.

4 Experimental Evaluation

We experimentally evaluate the performance of our proposed EDA-based approach (named as EHM-EDA). In particular, we compared it to two recent works [14,16] (named NHM-EDA and PSO respectively) that were conducted to solve the same problem. Two Web Service composition Challenge (WSC) benchmarks, i.e., WSC-08 and WSC-09 extended with QoS attributes are utilized for the experiment. These two benchmarks are widely used in recent service composition research, e.g. in [4,8,10,14–17].

The same number of evaluation times are ensured to conduct a fair comparison. In particular, we set the population size as 200, the number of generations as 300, and b_{ratio} as 0.0002. We run 30 independent repetitions for all the competing approaches. We set the weights in the fitness function Eq. (1) to balance the QoSM and QoS, following the existing work [14–16], i.e., w_1 and w_2 are set to 0.25, and w_3, w_4, w_5 and w_6 to 0.125. We set the parameter p for the plugin match 0.75 as recommended in [3]. Additional experiments are also conducted with other weights and parameters, where the same behavior is usually observed.

4.1 Comparison of the Fitness

We utilize an independent-sample T-test to test the significant difference in mean fitness and mean execution time over 30 repetitions of the three methods. In particular, a significant level 5% is established for all pairwise comparisons over the composition tasks in WSC-08 and WSC-09. We highlight the top performance with its related fitness value and standard deviation in Table 1, while the pairwise comparisons of fitness are summarized in Table 2. In pairwise comparisons, *win/draw/loss* shows frequencies one method outperforms, equals or is outperformed by another method.

Table 1. Mean fitness values for our approach in comparison to NHM-EDA [16] and PSO [14] (Note: the higher the fitness the better)

Task	EHM-EDA	NHM-EDA [16]	PSO [14]
WSC-08-1	0.5326 ± 0	0.504916 ± 0.010355	0.522621 ± 0.00283
WSC-08-2	0.614333 ± 0	0.614333 ± 0	0.614333 ± 0
WSC-08-3	0.456083 ± 0.000194	$0.455118 \pm 6.8e{-05}$	0.454343 ± 0.000531
WSC-08-4	0.463066 ± 0.001054	0.464498 ± 0.000117	0.464511 ± 0.000133
WSC-08-5	0.474222 ± 0.000414	0.469205 ± 0.000245	0.468536 ± 0.001148
WSC-08-6	0.472665 ± 0.000382	$0.474322 \pm 9.9e{-05}$	0.472942 ± 0.000736
WSC-08-7	0.488584 ± 0.000527	0.480765 ± 0	0.479235 ± 0.000502
WSC-08-8	0.462254 ± 0.00017	0.46182 ± 0	0.461478 ± 0.000371
WSC-09-1	0.604377 ± 0.00429	0.569929 ± 0.005625	0.568493 ± 0.009659
WSC-09-2	0.471123 ± 0.000234	$0.471164 \pm 1.2e{-05}$	0.4711 ± 0.000283
WSC-09-3	0.551159 ± 0	0.551159 ± 0	0.551159 ± 0
WSC-09-4	0.471059 ± 0.000404	0.472804 ± 0.000227	0.471512 ± 0.000904
WSC-09-5	0.47269 ± 0.000104	0.470408 ± 0	0.470132 ± 0.000304

Tables 1 and 2 show that the two EDA-based methods outperform the PSO-based method [14]. This observation agrees with the findings in our previous work [16] that learning the distributions of the superior subpopulation can help to find

Table 2. Summary of the statistical significance tests for fitness, where each column shows the win/draw/loss score of one method against a competing one for all tasks of WSC-08 and WSC-09.

Dataset	Method	EHM-EDA	NHM-EDA [16]	PSO [14]
WSC-08 (8 tasks)	EHM-EDA	-	2/1/5	2/1/5
	NHM-EDA [16]	**5**/**1**/2	-	1/2/5
	PSO [14]	**5**/**1**/2	5/2/1	-
WSC-09 (5 tasks)	EHM-EDA	-	1/2/2	0/2/3
	NHM-EDA [16]	**2**/**2**/1	-	0/2/3
	PSO [14]	**3**/**2**/0	3/2/0	-

higher-quality composition solutions. For the two EDA-based methods, EHM-EDA appears to be more effective. This corresponds well with our expectations that taking the services dependencies into account can enhance the competency of EDA for improving the quality of composition solutions.

It has been discussed in the examples of composition solutions analyzed in [14,15], a small improvement of fitness that measures QoS and QoSM can make a significant difference in the practical use of the computed composition service.

4.2 Comparison of the Execution Time

Tables 3 and 4 show the mean execution time with standard deviation over 30 repetitions and the frequencies of pairwise comparisons respectively.

Table 3 shows that two EDA-based approaches require less execution time consistently over PSO [14]. For the two EDA-based approaches, our EDA-based approach requires significantly and consistently less execution time than the competing EDA-based approach [16]. These correspond well with our assumptions: on the one hand, although useful services are more likely to be put in front of sampled service queue for the decoding algorithm in NHM-EDA [16], improvements on the efficiency may not be outstanding; on the other hand, our proposed sampling technique achieves outstanding efficiency with a straightforward implementation.

4.3 Comparison of the Convergence Rate

To investigate the effectiveness of our EDA-based approach, we use WSC-08-05 and WSC-08-08 as examples for demonstrating the convergence rate of fitness over 30 independent runs. Note that WSC-08-08 is a more challenging task than WSC08-05 as it involves more service dependencies and results in larger composite services.

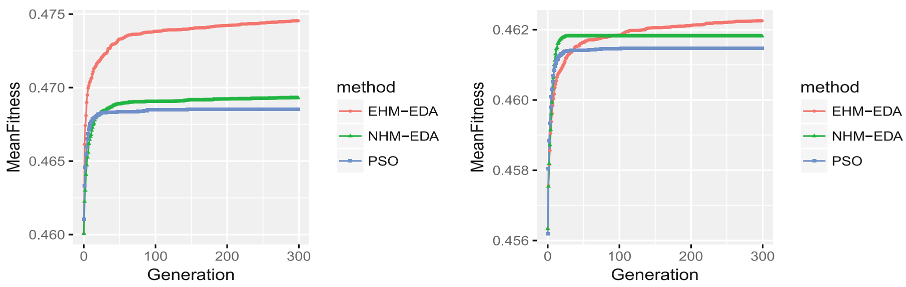
Figures 2a and b show the mean fitness of the best solutions found by EHM-EDA, NHM-EDA [16] and PSO [14] over 300 generations for the two composition tasks. In Fig. 2a, for the less challenging composition task (WSC08-5), we

Table 3. Mean execution time in seconds for our approach in comparison to NHM-EDA [16] and PSO[14] (Note: the lower the execution time the better)

Task	EHM-EDA	NHM-EDA [16]	PSO [14]
WSC-08-1	20 ± 1	152 ± 7	200 ± 130
WSC-08-2	13 ± 1	89 ± 12	130 ± 79
WSC-08-3	104 ± 4	1753 ± 87	4786 ± 1471
WSC-08-4	29 ± 1	86 ± 4	353 ± 109
WSC-08-5	50 ± 2	833 ± 141	4241 ± 1712
WSC-08-6	231 ± 7	18436 ± 1043	48215 ± 13973
WSC-08-7	96 ± 2	1351 ± 205	5482 ± 3277
WSC-08-8	204 ± 5	1267 ± 87	5890 ± 1534
WSC-09-1	18 ± 2	136 ± 11	284 ± 196
WSC-09-2	135 ± 11	2306 ± 283	6419 ± 1786
WSC-09-3	126 ± 4	782 ± 46	2273 ± 1007
WSC-09-4	733 ± 29	71932 ± 4370	105568 ± 31797
WSC-09-5	535 ± 20	6692 ± 565	19266 ± 5840

Table 4. Summary of the statistical significance tests for execution time, where each column shows the win/draw/loss score of one method against a competing one for all tasks of WSC-08 and WSC-09.

Dataset	Method	EHM-EDA	NHM-EDA [16]	PSO [14]
WSC-08 (8 tasks)	EHM-EDA	-	0/0/8	0/0/8
	NHM-EDA [16]	**8**/0/0	-	0/0/8
	PSO [14]	**8**/0/0	8/0/0	-
WSC-09 (5 tasks)	EHM-EDA	-	0/0/5	0/0/5
	NHM-EDA [16]	**5**/0/0	-	0/0/5
	PSO [14]	**5**/0/0	5/0/0	-



(a) WSC-08-5 (b) WSC-08-8

Fig. 2. Mean fitness values of best solutions over generations

observe that our EDA-based converges much faster against the two competing methods while the two competing methods reach a plateau in their early stages. In Fig. 2b, for the more challenging composition task (WSC-08-08), the two competing methods happen to converge fast in the early stage, but our EDA-based method eventually outperforms them. It can be inferred from those observations that EHM-EDA is less prone to premature convergence to local optima, but it may suffer from low convergence rate in more complex datasets, such as WSC08-8.

5 Related Work

AI planning and EC techniques have been acquired in web service composition to compute solutions automatically. AI planning is a commonly used technique to handle dynamic scenarios with agents in constructing composition plans, but combinatorial optimization is not a focus [12]. EC techniques have been widely used for optimizing QoS and/or QoSM in fully automated service composition [4,5,8–10,14–17]. These EC-based works can be categorized into two groups: conventional EC-based and model learning-based approaches.

Conventional EC techniques have been used to breed candidate solutions for an optimization purpose. Genetic Programming (GP) employs genetic operators directly on tree-based solutions, and it allows the evolution of composition structure as well as services for exploration and exploitation. [8] proposed a context-free grammar for initializing tree-based candidate solutions, while [17] randomly initialized tree-based candidate solutions without ensuring structures of composite solutions, but they proposed a general adaptive rule of crossover and mutation for improving quality of computed composite solutions. These two works present a low convergence rate since their population always consists of invalid candidate solutions that are required to be penalized by the fitness functions. To increase the convergence rate, a random greedy search algorithm was utilized in [4,10] to construct DAG-based valid candidate composite solutions for each population, and two different tree conversion algorithms were proposed to allow a straightforward application of GP. However, their tree-based representation allows replicas of subtrees that potentially build up huge trees. To eliminate these replicas, a tree-like representation was proposed in [15]. Other conventional EC techniques, like swarm intelligence, such as Particle Swarm Optimization was utilized to optimize the order of a queue of services, and each service is corresponding to the position of a particle, a decoding algorithm [14] are developed to decode the queue into DAG-based solutions.

Despite some successes in conventional EC techniques, some efforts have been made to investigate model learning-based algorithms, such as EDA. Two works [5,6] proposed EDA-based approaches to semi-automated services composition, but their distributions models can hardly support fully automated service composition. One recent work [16] proposed a novel representation that allows a Node Histogram Matrix to learn the distributions from composite solutions structured in different composition workflows. However, opportunities still exist to propose more effective approaches by proposing more suitable distributions, and sampling techniques from that distribution also remain to be developed.

6 Conclusion

In this paper, we proposed an effective EDA-based approach, which learns suitable distributions by considering service dependencies, and efficiently samples high-quality solutions. The advantages of this approach have been experimentally illustrated by comparing it with NHM-EDA [16] and PSO [14]. In the future, we will study its scalability for more challenging datasets as scalability is a common difficulty faced by most algorithms, and develop local search strategies to enhance its searching ability.

References

1. Curbera, F., Nagy, W., Weerawarana, S.: Web services: why and how. In: Workshop on Object-Oriented Web Services-OOPSLA (2001)
2. Hauschild, M., Pelikan, M.: An introduction and survey of estimation of distribution algorithms. Swarm Evol. Comput. **1**(3), 111–128 (2011)
3. Lécué, F.: Optimizing QoS-aware semantic web service composition. In: Bernstein, A., et al. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 375–391. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04930-9_24
4. Ma, H., Wang, A., Zhang, M.: A hybrid approach using genetic programming and greedy search for QoS-aware web service composition. In: Hameurlain, A., Küng, J., Wagner, R., Decker, H., Lhotska, L., Link, S. (eds.) Transactions on Large-Scale Data- and Knowledge-Centered Systems XVIII. LNCS, vol. 8980, pp. 180–205. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46485-4_7
5. Peng, S., Wang, H., Yu, Q.: Estimation of distribution with restricted Boltzmann machine for adaptive service composition. In: IEEE ICWS, pp. 114–121 (2017)
6. Pichanaharee, K., Senivongse, T.: QoS-based service provision schemes and plan durability in service composition. In: Meier, R., Terzis, S. (eds.) DAIS 2008. LNCS, vol. 5053, pp. 58–71. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68642-2_5
7. Rao, J., Su, X.: A survey of automated web service composition methods. In: Cardoso, J., Sheth, A. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 43–54. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30581-1_5
8. Rodriguez-Mier, P., Mucientes, M., Lama, M., Couto, M.I.: Composition of web services through genetic programming. Evol. Intell. **3**(3–4), 171–186 (2010)
9. Sadeghiram, S., Ma, H., Chen, G.: Cluster-guided genetic algorithm for distributed data-intensive web service composition. 2018 IEEE Congress on Evolutionary Computation (CEC) (2018)
10. da Silva, A.S., Ma, H., Zhang, M.: Genetic programming for QoS-aware web service composition and selection. Soft Comput. **20**, 1–17 (2016)
11. da Silva, A.S., Mei, Y., Ma, H., Zhang, M.: Evolutionary computation for automatic web service composition: an indirect representation approach. J. Heuristics **24**, 1–32 (2017)

12. Tong, H., Cao, J., Zhang, S., Li, M.: A distributed algorithm for web service composition based on service agent model. IEEE Trans. Parallel Distrib. Syst. **22**(12), 2008–2021 (2011)
13. Tsutsui, S., Pelikan, M., Goldberg, D.E.: Node histogram vs. edge histogram: a comparison of PMBGAs in permutation domains. MEDAL Report (2006009) (2006)
14. Wang, C., Ma, H., Chen, A., Hartmann, S.: Comprehensive quality-aware automated semantic web service composition. In: Peng, W., Alahakoon, D., Li, X. (eds.) AI 2017. LNCS (LNAI), vol. 10400, pp. 195–207. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63004-5_16
15. Wang, C., Ma, H., Chen, A., Hartmann, S.: GP-based approach to comprehensive quality-aware automated semantic web service composition. In: Shi, Y., et al. (eds.) SEAL 2017. LNCS, vol. 10593, pp. 170–183. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68759-9_15
16. Wang, C., Ma, H., Chen, G., Hartmann, S.: Knowledge-driven automated web service composition—an EDA-based approach. In: International Conference on Web Information Systems Engineering. Springer (2018)
17. Yu, Y., Ma, H., Zhang, M.: An adaptive genetic programming approach to QoS-aware web services composition. In: IEEE CEC, pp. 1740–1747 (2013)
18. Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: Proceedings of the 12th International Conference on World Wide Web, pp. 411–421. ACM (2003)