Memetic EDA-Based Approaches to QoS-Aware Fully Automated Semantic Web Service Composition

Chen Wang, Hui Ma, *Member, IEEE*, Gang Chen, *Member, IEEE*, and Sven Hartmann, *Member, IEEE*

***Abstract*—Quality-of-service (QoS)-aware automated semantic Web service composition aims to find a composite service with optimized or near-optimized QoS and quality of semantic matchmaking within polynomial time. To cope with this NP-hard problem with high complexity, a variety of evolutionary computation (EC) techniques has been developed. To improve the effectiveness and efficiency of these techniques, in this article, we proposed a novel memetic estimation of the distribution algorithm-based approach, namely, MEEDA, to tackle this problem. In particular, MEEDA explores four different domain-dependent local search methods that search for effective composite services by utilizing several neighborhood structures. Apart from that, to significantly reduce the computational time of MEEDA, an efficient local search strategy is introduced by combining a uniform fitness distribution scheme for selecting suitable solutions and stochastic local search operators for effectively and efficiently exploiting neighbors. To better demonstrate MEEDA's effectiveness and scalability, we create a more challenging, augmented version of the service composition benchmark dataset. Experimental results on this benchmark show that MEEDA with newly developed domain-dependent local search operator, i.e., layer-based constrained one-point swaps, significantly outperforms existing state-of-the-art algorithms in finding high-quality composite services.**

***Index Terms*—Combinatorial optimization, estimation of distribution algorithm (EDA), quality of service (QoS) optimization, Web service composition.**

I. Introduction

SERVICE-ORIENTED architecture (SOA) has been contributing to the reuse of software components [1]. *Web services* are one of the most successful implementations of SOA to provide services as "modular, self-describing, self-contained applications that are available on the Internet" [2]. Since users' requirements cannot often be satisfied by one existing Web service, Web service composition aims to loosely couple a set of Web services to provide a value-added composite service (i.e., a solution of service composition) that accommodates users' complex requirements [3]. These requirements are often related to functional (i.e., quality of semantic matchmaking as QoSM) and nonfunctional (i.e., quality of service as QoS) requirements, which give birth to *QoS-aware semantic Web service composition* that aims to optimize QoSM and QoS of composite services simultaneously.

The service composition problem is known to be NP-hard [4]. Existing works [5]–[13] to tackle this difficult problem can be classified as *semiautomated* and *fully automated* Web service composition [14] approaches. Semiautomated approaches assume that users will provide an abstract service composition workflow, and all the composite services produced by the composition system must strictly obey the given workflow. However, this assumption is not always valid since the workflow may not be provided or not even known by users. Fully automated approaches do not rely on workflows to be provided. Instead, a composite service will be constructed from scratch by selecting and connecting multiple atomic services obtained from a service repository [14], possibly discovering different workflows. Apparently, compared to semiautomated Web service composition, fully automated Web service composition opens new opportunities to improve QoS and QoSM. Nevertheless, the difficulty of the composition task is also increased [15].

AI planning and evolutionary computation (EC) are two of the most widely used techniques for *fully automated* Web service composition [5], [7], [10], [13], [16]–[18]. AI planning techniques focus on creating functionally correct composite services. However, they often ignore the importance of optimizing the QoS or QoSM of composite services [19]. EC techniques have been widely used to solve the fully automated semantic Web service composition problems that aim to optimize QoSM and QoS. They are considered highly useful in practice as they can efficiently find "good enough" composite services. Promising approaches [5]–[13] based on genetic algorithms (GAs) [20], genetic programming (GP) [21], particle swarm optimization (PSO) [22], and estimation of distribution algorithm (EDA) [23], have been widely investigated in the literature.

Manuscript received December 28, 2020; revised May 16, 2021 and August 30, 2021; accepted October 27, 2021. Date of publication November 12, 2021; date of current version May 30, 2022. This work was supported by the New Zealand Marsden Fund, administrated by the Royal Society of New Zealand under Grant VUW1510. *(Corresponding author: Chen Wang.)*

Chen Wang is with the National Institute of Water and Atmospheric Research and the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6041, New Zealand (e-mail: chen.wang@ecs.vuw.ac.nz).

Hui Ma and Gang Chen are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6041, New Zealand (e-mail: hui.ma@ecs.vuw.ac.nz; aaron.chen@ecs.vuw.ac.nz).

Sven Hartmann is with the Department of Informatics, Clausthal University of Technology, 38678 Clausthal-Zellerfeld, Germany (e-mail: sven.hartmann@tu-clausthal.de).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2021.3127633.

Digital Object Identifier 10.1109/TEVC.2021.3127633

To effectively search for good solutions, EC techniques often employ useful information distilled from promising solutions to produce new offspring. This information can be used either implicitly or explicitly. Conventional EC techniques, such as GA and GP, fall in the implicit camp by producing new solutions through recombining solutions evolved previously [5], [7], [13]. On the other hand, EDA explicitly utilizes solution knowledge in the form a *distribution model* based on a set of parent individuals to achieve good performance in solving classical problems (e.g., traveling salesman problem), compared to GA [24].

In the past, EDA has been applied to semiautomated service composition [6], [25]. For example, Pichanaharee and Senivongse [25] employed a binary representation to encode composite services, and it learns a set of independent probabilities where 0 or 1 appears at each bit of the binary representation. Different from [25], a different distribution model based on the restricted Boltzmann machine is learned based on a similar binary representation [6]. These two works can only cope with semiautomated service composition because their binary representations are designed to encode service selection decision for a given workflow of service composition. Learning distributions over a predefined fixed structure is relatively less challenging. However, for fully automated service composition, there is no predefined structure. Therefore, we need to design a new representation with a suitable distribution model that can catch knowledge of good solutions to achieve fully automated service composition.

Our preliminary research indicates that EDA is more effective at global exploration rather than local exploitation [26]. Since the distribution model encourages the algorithm to explore more promising regions in the entire solution space, without attempting to improve the quality of any specific solutions evolved previously. Meanwhile, the optimization performance can be noticeably improved through local modifications to promising solutions. By restricting the target region for local search and reducing the randomness involved in sampling directly from the distribution model, the application of local search in EDA can potentially expedite the search for optimal solutions. Motivated by this understanding, we aim to develop new memetic EDA approaches for fully automated service composition that seamlessly integrate EDA-based global search with newly developed local search techniques.

Although memetic EDA algorithms have been successfully applied to many challenging problems, such as arc routing [26] and assembly flow-shop scheduling problems [27], their effectiveness for QoS-aware semantic Web service composition is largely unknown. To develop a memetic EDA approach to QoS-aware semantic Web service composition, several challenges remain to be addressed as follows.

First, a composite service is commonly represented as a directed acyclic graph (DAG) [28], [29]. Exploring the neighborhood of a DAG, especially large DAGs, is computationally infeasible [30]. Therefore, researchers [9], [31] often indirectly define the neighborhood of a composite service represented in the form of a permutation, which can be converted to a DAG through a separate decoding process [9]. For example, the "swap" operators proposed in [32] can produce neighboring solutions by swapping two random elements in a permutation. Consequently, a neighborhood is defined by a collection of permutations obtainable through a number of swaps performed on any given permutation. However, such a neighborhood often contains a large proportion of neighboring permutations with inferior quality. For effective local search, the neighborhood must be refined to exclude most of the clearly unwise swapping choices by exploiting domain-specific knowledge.

Second, a traditional strategy that exclusively explores the whole neighboring space of composite services can incur high computational cost. For example, if a swap operator is utilized to explore the neighborhood of a permutation of length n, the size of neighborhood over this permutation is $[(n(n-1))/(2)]$, upon considering all possible combinations of pair swaps [31]. In [32], this neighborhood size is reduced to $n-1$ by restricting all the pair swaps to always include a selected position. In the context of service composition, the length of such a permutation is usually equivalent to the size of the service repository. When the size of the service repository is very large, e.g., a maximum of 15 211 Web services are contained in the WSC09 benchmark [33], exploiting the neighborhood of size $n-1$ becomes very expensive.

Third, it is very challenging to determine which candidate solutions are to be selected for local search in memetic algorithms, as the selection method has a significant impact on the effectiveness and efficiency of memetic EDA. In [9], an equal chance is given to all the candidate solutions to be selected. However, selection methods for local search often vary from many factors, such as EC algorithms, domain problems, etc. Therefore, the existing selection methods cannot be utilized directly in our memetic EDA-based approach.

Fourth, EDA frequently samples repeated solutions, hurting computational efficiency. However, these solutions are essential to ensure good convergence property of the learned distribution model. Therefore, it is vital to develop new techniques that can reduce computational time caused by the repetitive sampling while helping the distribution model to converge.

To address these challenges above, we propose a novel Memetic EDA-based approach, namely, MEEDA, achieving substantially high performance in effectiveness and efficiency. This performance is evidenced by empirical comparison with several recently proposed Web service composition approaches, such as an EDA-based approach [12], a PSO-based approach [10], GA- and Memetic GA-based approaches [9], and a non-EC graph search-based approach [15]. In the process of developing and evaluating MEEDA, this article achieves five main contributions as highlighted below.

1) To perform an effective local search on composite services, we propose four neighborhood structures for candidate solutions. These neighborhoods are structured by developing four novel domain-dependent local search operators, based on constructing and swapping effective building blocks of composite services.
2) To efficiently exploit the neighborhood of composite services, stochastic local search is performed to produce

a fixed, relatively a small number of neighbors, using randomness to avoid getting trapped in local optima.

3) To determine candidate solutions for local search, a uniform fitness distribution scheme is employed to select a small set of suitable candidate solutions. This schema is combined with the stochastic local search to form an effective and efficient MEEDA algorithm.
4) To avoid repetitively sampling while ensuring good convergence, we utilize an archive technique to reserve half the population size of elite individuals to construct a distribution model in the next generation. Meanwhile, reserved individuals can also significantly reduce the overall computational time for their evaluation in next generation.
5) To demonstrate the performance of the MEEDA algorithm, we augment two benchmark datasets, i.e., WSC-08 [34] and WSC-09 [33]. The new benchmark inherits the functionalities provided by services in benchmark datasets WSC-08 and WSC-09 and QoS attributes of Web services in the benchmark dataset QWS [35]–[37]. The number of Web services in the service repository is doubled (with much bigger searching space) in order to examine whether our MEEDA algorithm can solve problems with significantly larger sizes. Both the augmented benchmark datasets and our implementation of the MEEDA algorithm have been made freely available online.[1] We experimentally compare our MEEDA algorithm with five state-of-the-art methods that have been recently proposed to solve the same or a similar service composition problem using the new benchmark. Our experimental results illustrate that our method can outperform state-of-the-art algorithms.

II. Related Work

In this section, we review some state-of-the-art EC-based and non-EC-based service composition approaches for solving fully automated service composition. Afterward, we discuss some memetic EC-based approaches, including their local search techniques. Finally, we discuss EDA and its applications.

A. Fully Automated Web Service Composition

Automated Web service composition aims to simultaneously find a proper service composition workflow and select suitable services for the components of the workflow to achieve optimal quality. Existing works in fully automated Web service composition can be divided into two categories: 1) direct and 2) indirect approaches [9]. The direct approaches represent composite services explicitly in the representation that intuitively displays actual execution flows of composite services. In contrast, the indirect approaches often represent composite services implicitly as permutations, which require a decoding process to build up actual execution workflows.

[1]Two augmented benchmarks for automated Web service composition are available from https://github.com/chenwangnida/Dataset, and the source code of our memetic EDA-based approach is available from https://github.com/chenwangnida/MENHBSA4SWSC.

In the first category, tree- and graph-based representations have been widely used to represent composite services directly. A graph-based evolutionary process is introduced in [28] to directly evolve DAG-based composite services, applying domain-dependent crossover and mutation operators with repairing methods. GP is utilized for searching optimal solutions represented as trees. Rodriguez-Mier *et al.* [7] proposed a context-free grammar for randomly initializing tree-based composite services to ensure the correct structures of composite services. In contrast, Yu *et al.* [13] randomly initializes tree-based composite services and develops adaptive crossover and mutation rates according to the diversity of the population for accelerating the speed of convergence. Both approaches [7], [13] design a fitness function that consider the correctness of solutions and utilize a penalization method for filtering incorrect solutions during the evolution process. To achieve better QoS, Ma *et al.* [5] and Da Silva *et al.* [8] utilized a greedy search algorithm for creating correct DAG-based composition workflows, which are mapped to tree-based ones. During the evolution process, the correctness of the solutions is ensured by domain-dependent crossover and mutation. However, the mapped tree-based representations might grow very large with many replicas of subtrees because of their mapping methods and lead to poor searching performance in finding high-quality composite services. To overcome this issue, Wang *et al.* [11] proposed a tree-like representation, on which replicas of a subtree are represented by pointers from the replicas to the root of the subtree.

In the second category, composite services are represented as permutations, which are then decoded into solutions represented as DAGs [9], [10], [29]. PSO is utilized to find an optimized queue of services (i.e., a permutation), which can be decoded into a corresponding composite service represented as a DAG [29]. Wang *et al.* [10] extended [29] to jointly optimize QoSM and QoS, where a weighted DAG is decoded, where edge weights correspond to QoSM between services. These two PSO-based approaches rely on PSO to determine the weights of the particle's position (that corresponding with a service) to form an ordered service queue. Optimizing QoSM and QoS simultaneously is more challenging than optimizing QoS because the searching space is significantly increased, demanding more effective and efficient searching techniques. It has been suggested that utilizing the indirect representation often contributes to higher performance, compared to direct representation [9]. This is due to the flexibility provided by the indirect representation in both population initialization and search process using evolution operators.

Graph search [15], [38]–[43] is an alternative method for fully automated service composition. Graph search methods work on searching composite services constructed by subgraphs or paths from a service dependency graph. However, constructing such a service dependency graph could suffer a scalability issue when dealing with a large service repository with complex service dependencies. For example, $A^\star$ search [44] is utilized to search composite services presented as paths, which are constructed from a subgraph extracted from a service dependency graph [45]. Another work [40] transforms each search step on a service dependency graph as a dynamic knapsack problem and proposes

a knapsack-variant algorithm to effectively and efficiently generate composite services with a minimal number of component services. However, these two works [40], [44] only focus on minimizing the number of component services without considering QoS or QoSM. Besides that, the scalability of these methods can suffer when the service repository grows. To address this critical issue, Chattopadhyay *et al.* [38] proposed a scalable way of pruning dependency graphs and a novel path-based search method for QoS-aware service composition. This work can efficiently construct near-optimal composite services. However, it only considers a single quality criterion in QoS. To consider multiple quality criteria in QoS, Chattopadhyay *et al.* [15] recently proposed an improved path-based search method based on [38]. Particularly, a node (i.e., an atomic service) associated with a higher rank is preferred during the path construction process, and the nodes are ranked based on the concept of dominance over multiple QoS quality criteria.

In summary, EC techniques have been showing their promises in fully automated Web service composition. Moreover, indirect representations have shown to be more effective than EC approaches with direct representations. Therefore, EC techniques with indirect representations are exciting techniques to be focused on and will be compared to the state-of-the-art graph search methods in this article.

B. Memetic EC-Based Approaches and EDA

Memetic algorithms have drawn growing attention from researchers in recent years and achieved significant successes in many applications [46]. For example, Tabu search is combined with GP to solve QoS-aware data-intensive Web service composition [47]. da Silva *et al.* [9] proposed an indirect memetic approach for QoS-aware Web service composition, where a domain-dependent crossover operator is proposed to produce candidate solutions. Besides that, an exhaustive local search is applied to composite services represented as permutations. However, the produced neighbors are likely to be decoded into the same composite service, potentially affecting algorithm performance.

Recently, EDA has been used to tackle many combinatorial optimization problems [24]. In particular, domain-dependent local search operators are often introduced to enhance the performances of EDA. For example, a probability matrix related to the job priority permutation of a solution is learned in the EDA-based flow-shop job scheduling problem [26]. Meanwhile, different problem-specific local search operators are proposed to enhance the exploitation ability of EDA. An edge histogram matrix is learned from solutions represented by a set of routes for the application of uncertain capacitated arc routing problems [27]. Meanwhile, different move operators, such as single insertion and swap, are proposed to make local improvements.

The use of EDA has only been investigated for semiautomated Web service composition [6], [25], [48]. In a recent conference paper [12], we proposed an EDA-based approach for fully automated Web service composition, where novel permutations are proposed as candidate solutions based on a service repository. The proposed method relies on the learned distribution model in the form of a node histogram matrix (NHM), and the node histogram-based sampling algorithm (NHBSA) [23] is employed to produce candidate solutions according to NHM. The difference between our previous conference paper [12] and this article is that in this article we propose memetic EDA algorithms for which we propose novel local search, to effectively search for good solutions with balanced exploration and exploitation ability.

In summary, on the one hand, memetic EDA-based approaches have been investigated in many problems, other than fully automated service composition, achieving promising results. On the other hand, notwithstanding success achieved in our initial investigation in EDA-based fully automated service composition, EDA's performance can be significantly improved by introducing new ideas to better perform local search in EDA.

III. Semantic Web Service Composition

A *semantic Web service* (*service*, for short) is considered as a tuple $S = (I_S, O_S, \text{QoS}_S)$ 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 $\text{QoS}_S = \{t_S, ct_S, r_S, a_S\}$ is a set of nonfunctional 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, ct_S, r_S, and a_S refer to the response time, cost, reliability, and availability of service S, respectively, which are four commonly used QoS attributes [49].

A *service repository* $\mathcal{SR}$ is a finite collection of services supported by a common ontology $\mathcal{O}$.

A *composition task* (also called *service request*) 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 $\mathcal{O}$.

Two special atomic services $Start = (\emptyset, I_T, \emptyset)$ and $End = (O_T, \emptyset, \emptyset)$ are always included in $\mathcal{SR}$ to account for the input and output of a given composition task T.

We use matchmaking types (MTs) to describe the level of a match between outputs and inputs [50]. For concepts a and b in $\mathcal{O}$, the *matchmaking* returns *exact* if a and b are equivalent ($a \equiv b$), *plugin* if a is a subconcept of b ($a \sqsubseteq b$), *subsume* if a is a superconcept of b ($a \sqsupseteq b$), and *fail* if none of the previous MTs is returned. In this article, we are only interested in *exact* and *plugin* matches for robust compositions, see [51]. As argued in [51], *plugin* matches are less preferable than *exact* matches due to the overheads associated with data processing. For *plugin* matches, the semantic similarity (SIM) of concepts is suggested to be considered when comparing different *plugin* matches.

A *robust causal link* [52] is a link between two matched services S and S', denoted as $S \rightarrow S'$, if an output a ($a \in O_S$) of S serves as the input b ($b \in I_{S'}$) of S' satisfying either $a \equiv b$ or $a \sqsubseteq b$. For concepts a and b in $\mathcal{O}$, the *semantic similarity* $\text{sim}(a, b)$ is calculated based on the edge counting method in a taxonomy like WorldNet [53]. Advantages of this method are simple calculation and good semantic measurement [53]. As discussed in [52], we use *MT* ($\text{type}_{\text{link}}$) and *semantic similarity* (sim_{link}) to denote robust causal link, which is defined

TABLE I
QoS CALCULATION FOR A COMPOSITE SERVICE EXPRESSION C

$C=$	$r_C=$	$a_C=$	$ct_C=$	$t_C=$
$\bullet(C_1,\ldots,C_d)$	$\prod_{k=1}^{d} r_{C_k}$	$\prod_{k=1}^{d} a_{C_k}$	$\sum_{k=1}^{d} ct_{C_k}$	$\sum_{k=1}^{d} t_{C_k}$
$\parallel(C_1,\ldots,C_d)$	$\prod_{k=1}^{d} r_{C_k}$	$\prod_{k=1}^{d} a_{C_k}$	$\sum_{k=1}^{d} ct_{C_k}$	$MAX\{t_{C_k}\|k\in\{1,...,d\}\}$
$+(C_1,\ldots,C_d)$	$\prod_{k=1}^{d} p_k\cdot r_{C_k}$	$\prod_{k=1}^{d} p_k\cdot a_{C_k}$	$\sum_{k=1}^{d} p_k\cdot ct_{C_k}$	$\sum_{k=1}^{d} p_k\cdot t_{C_k}$
$*C_0$	${r_{C_0}}^{\ell}$	${a_{C_0}}^{\ell}$	$\ell\cdot ct_{C_0}$	$\ell\cdot t_{C_0}$

as follows:

$$\text{type}_{\text{link}}=\begin{cases}1, & \text{if } a\equiv b \text{ (\textit{exact} match)}\\ p, & \text{if } a\sqsubseteq b \text{ (\textit{plugin} match)}\end{cases} \tag{1}$$

$$\text{sim}_{\text{link}}=\text{sim}(a,b)=\frac{2N_c}{N_a+N_b} \tag{2}$$

with a suitable parameter p, $0<p<1$, and with N_a, N_b, and N_c, which measure the distances from concept a, concept b, and the closest common ancestor c of a and b to the top concept of the ontology $\mathcal{O}$, respectively. If more than one pair of matched output and input exist from service S to service S', $\text{type}_{\text{link}}$ and sim_{link} will take on their average values.

The *QoSM* of a composite service measured by MT and SIM is obtained by aggregating all robust causal links as follows:

$$\text{MT}=\prod_{j=1}^{m}\text{type}_{link_j} \tag{3}$$

$$\text{SIM}=\frac{1}{m}\sum_{j=1}^{m}\text{sim}_{link_j}. \tag{4}$$

Formal expressions as in [54] are used to represent service compositions. The constructors $\bullet$, $\parallel$, $+$, and $*$ are used to denote sequential composition, parallel composition, choice, and iteration, respectively. The set of *composite service expressions* is the smallest collection $\mathcal{SC}$ that contains all atomic services and that is closed under these constructors. That is, whenever $C_0, C_1, \ldots, C_d$ are in $\mathcal{SC}$, then $\bullet(C_1,\ldots,C_d)$, $\parallel(C_1,\ldots,C_d)$, $+(C_1,\ldots,C_d)$, and $*C_0$ are in $\mathcal{SC}$, too. Let C be a composite service expression. If C denotes an atomic service S, then its QoS is given by QoS_S. Otherwise, the QoS of C can be obtained inductively as summarized in Table I. Herein, $p_1,\ldots,p_d$ with $\sum_{k=1}^{d} p_k=1$ denote the probabilities of the different options of the choice $+$, while ℓ denotes the average number of iterations. Therefore, QoS of a service composition solution, i.e., availability (A), reliability (R), execution time (T), and cost (CT), can be obtained by aggregating a_C, r_C, t_C, and ct_C as in Table I.

In the presentation of this article, we mainly focus on two constructors: 1) sequence $\bullet$ and 2) parallel $\parallel$, similar to most automated service composition works [5], [8], [12], [15], [28], [29], where a *composite service* is often represented in the form of a directed acyclic graph (DAG, denoted as $\mathcal{G}$). In a DAG, nodes represent Web services (also called *component services*) and edges represent robust causal links. A *composite service* can also be indirectly represented as a permutation $\Pi=(\pi_0,\pi_1,\ldots,\pi_{n-1})$, whose elements are $\{0,1,\ldots,n-1\}$ such that $\pi_i\neq\pi_j$ for all $i\neq j$. Each element in Π represents a unique index of a relevant service in the service repository for a composition task. Let the number of relevant services equal m, and each element in Π is associated with one unique indices from 0 to $m-1$. According to [12], a permutation Π needs to be interpreted, and can be further decoded into a $\mathcal{G}$ (denoted as $\Pi\Rightarrow\mathcal{G}$). Such a decoding process can always ensure the correctness of composite services if $\Pi\Rightarrow\mathcal{G}$ holds.

When multiple quality criteria are involved in decision making, objective function of QoS-aware semantic service composition is defined as maximizing the fitness of a solution that is defined as a weighted sum of all quality criteria in (5), assuming the preference of each quality criterion is provided by the user [55]

$$\begin{aligned}\text{Fitness}(C)=w_1\hat{\text{MT}}+w_2\hat{\text{SIM}}+w_3\hat{A}+w_4\hat{R}+w_5\left(1-\hat{T}\right)\\ +w_6\left(1-\hat{CT}\right)\end{aligned} \tag{5}$$

with $\sum_{k=1}^{6} w_k=1$. This objective function is referred to as a *comprehensive quality model* for service composition in this article. We can adjust the weights according to the user's preferences. $\hat{\text{MT}}$, $\hat{\text{SIM}}$, $\hat{A}$, $\hat{R}$, $\hat{T}$, and $\hat{CT}$ are normalized values calculated within the range from 0 to 1 using (6). To simplify the presentation, we also use the notation $(Q_1,Q_2,Q_3,Q_4,Q_5,Q_6)=(\text{MT},\text{SIM},A,R,T,CT)$. Q_1 and Q_2 have minimum value 0 and maximum value 1. The minimum and maximum values of Q_3, Q_4, Q_5, and Q_6 are calculated across all the relevant services (that are determined in Section IV-B) in the service repository $\mathcal{SR}$ using greedy search as in [5] and [8]

$$\hat{Q}_k=\begin{cases}\frac{Q_k-Q_{k,\min}}{Q_{k,\max}-Q_{k,\min}}, & \text{if } k=1,\ldots,4 \text{ and } Q_{k,\max}-Q_{k,\min}\neq 0\\ \frac{Q_{k,\max}-Q_k}{Q_{k,\max}-Q_{k,\min}}, & \text{if } k=5,6 \text{ and } Q_{k,\max}-Q_{k,\min}\neq 0\\ 1, & \text{otherwise.}\end{cases} \tag{6}$$

The goal of QoS-aware semantic service composition is to find a composite service expression $C^{\star}$ that maximizes the objective function in (5). $C^{\star}$ is hence considered as the best possible solution for a given composition task T.

IV. MEEDA APPROACH FOR SEMANTIC WEB SERVICE COMPOSITION

In this section, we present our MEEDA approach to fully automated semantic Web service composition. Particularly, we start by giving an overview of our MEEDA approach. Subsequently, we discuss some essential components in the approach. The goal of the first component is to discover relevant services and service layers (see details in Section IV-B). The goal of the second component is to

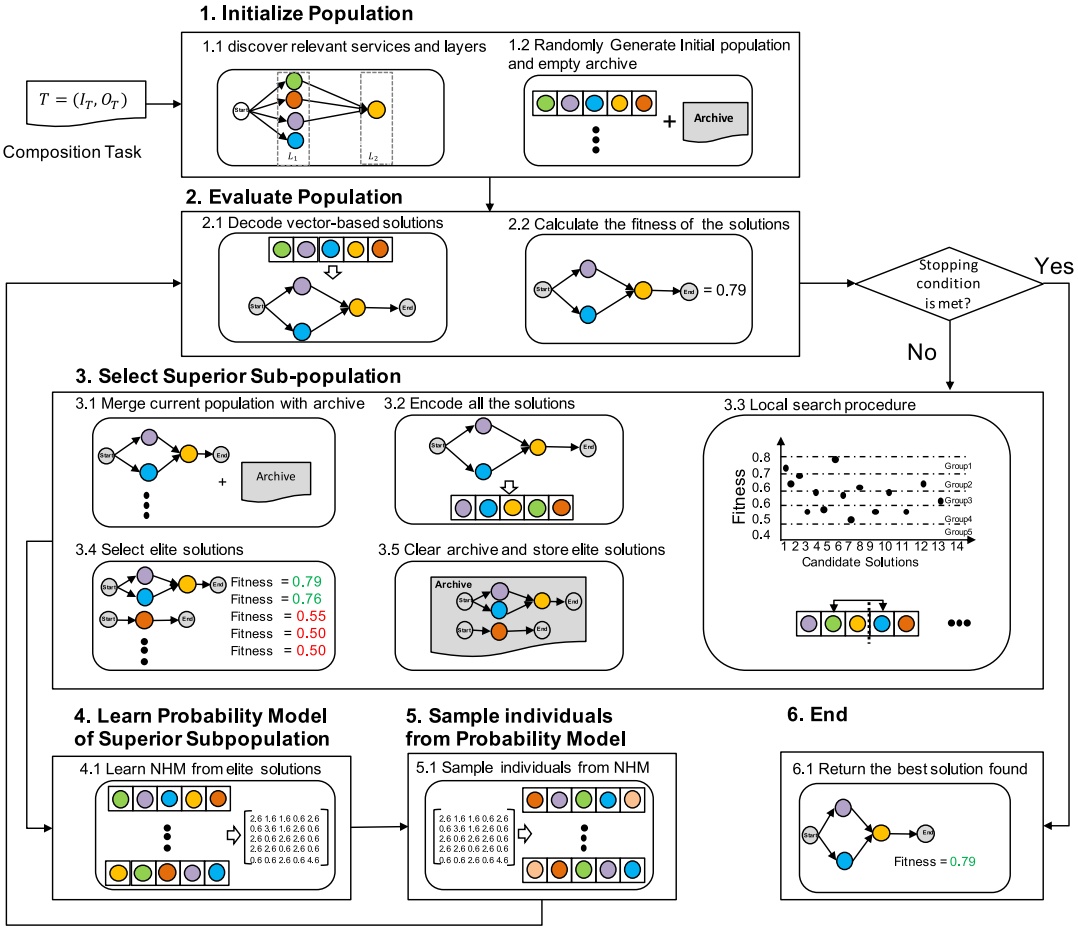

Fig. 1. Overview of memetic EDA-based approach for automated Web service composition.

introduce a permutation-based representation (see details in Sections IV-C and IV-D). The goal of the third component is to introduce an effective local search procedure (see details in Section IV-E).

We propose several key ideas that are jointly employed to build our MEEDA approach.

1) As we discussed in Section I, EDA's success strongly relies on the proper distribution model for learning the distribution of promising solutions. Our initial study [12] represents a composite service as a unique queue of services, i.e., a permutation of atomic services, mapped from a DAG-based solution. Composite services in this permutation form contribute to a reliable distribution model to be learned and new promising permutations to be sampled. Therefore, a bidirectional map is ensured between permutations and DAGs for learning and evaluation purposes.
2) Exploring the neighborhood of a large DAG-based composite service is unusually computationally infeasible [30]. However, it is straightforward to define the neighborhood on a permutation-based representation by the so-called swap operators. To develop effective swap operators, we utilize domain knowledge of service composition to create effective building blocks for these swap operators on permutation-based candidate solutions. These swap operators aim to find fitter neighbors effectively. That is they are likely to make local improvements in the produced neighbors.
3) To significantly decrease the computational time of the local search, it is crucial to select a restricted number of suitable candidate solutions. As we know, fitness measures the importance of different candidate solutions. Moreover, the local search should be performed on solutions with distinctive importance. Therefore, we strategically group candidate solutions based on their fitness values according to a uniform distribution scheme and randomly select a candidate solution from each group for the local search.
4) It is not efficient to exhaustively explore the whole neighbors in the conventional local search [9]. Instead, stochastically searching the neighboring solutions not only can significantly reduce the computational cost but also escape the local optimal easily [27]. Therefore, we introduce a stochastic local search strategy to effectively and efficiently exploit the neighborhood of the selected candidate composite services.

A. Overview of Our MEEDA Approach

An overview of the MEEDA approach is represented in Fig. 1, consisting of the following steps: initialize population, evaluate population, select superior subpopulation, learn the probability

model, sample individuals, and return the best solution. We start with discovering all the relevant services related to a given composition request T in step 1.1, where several service layers are identified (see details in Section IV-B). These relevant services are used to randomly generate m permutations as Π'^{g}_{k}, where a generation counter g starts from 0, and k means the kth of m permeation in a population. In step 2.1, these permutations are decoded into DAG-based solutions using a forward graph building technique [10], based on which the fitness in (5) of each individual can be easily calculated in step 2.2. In step 3.1, we merge the current population $\mathcal{P}^g$ with an archive. The archive is an empty set initially and will store the best composite services ever discovered by the algorithm. By adopting a breath-first search (BFS) strategy on each corresponding DAG-based solution in the merged population, we produce another encoded permutation-based solution Π^g_k in step 3.2. This newly produced solution allows a reliable NHM to be learned for sampling new promising solutions (see details in Section IV-C). In step 3.3, a local search method is applied to a very small set of these permutations. This small permutation set is selected based on a fitness uniform selection scheme over the current population (see details in IV-E1). For each permutation in the set, a stochastic local search is employed to create new permutations as its neighbors, where the best neighbor is identified based on the fitness value. This permutation in the small set is replaced with its best neighbor (see details in Section IV-E2). In step 3.4, the top half of the best-performing solutions are reserved in $\mathcal{P}^g$ according to their fitness values and put them into the archive. In step 4.1, we use the solutions in the archive to learn a NHM^g of generation g. In step 5.1, NHM^g is used to produce offsprings for generation $g+1$ using NHBSA (see details in Section IV-D). Consequently, we go back to step 2.1 to evaluate the fitness of new offsprings. Steps 2–4 will be repeated until the maximum number of generations is reached. Eventually, the best solutions found throughout the evolutionary process is returned.

In a nutshell, we introduce a permutation-based representation derived from the common DAG-based representation. We always switch between these two representations back and forth for better searching or evaluation purposes in our MEEDA approach. In addition, an archive technique is introduced to reserve half the population size of elite individuals to the next generation for better trace promising searching areas and saving computational cost. Furthermore, an effective local search procedure is developed through the use of the selection scheme and the stochastic local search.

B. Relevant Services and Service Layers

Discovering relevant services and service layers is an initial but crucial step for our MEEDA approach. Given a service repository and a composition task, a service is called relevant for the task, if it occurs in any composition for this task, i.e., lies on a path from the start node. We achieve two goals at this initial stage: 1) reduce the size of the service repository $\mathcal{SR}$ by keeping only those that are relevant to the service composition task T and 2) identify service layers of these relevant services. In particular, a group of layers is identified, and each layer contains a set of services that have the same longest distance to *Start*. We adopt a method in [56] to find relevant services and service layers as illustrated in the following example.

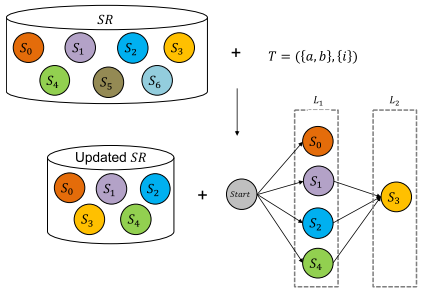

Fig. 2. Example of discovering relevant services and service layers for a service request T.

Example 1: We consider a *composition task* $T = (\{a, b\}, \{i, h\})$ and an $\mathcal{SR}$ consisting of seven atomic services. $S_0 = (\{b\}, \{i, h\}, \text{QoS}_{S_0})$, $S_1 = (\{a\}, \{f, g\}, \text{QoS}_{S_1})$, $S_2 = (\{a, b\}, \{h\}, \text{QoS}_{S_2})$, $S_3 = (\{f, h\}, \{i\}, \text{QoS}_{S_3})$, $S_4 = (\{a\}, \{f, g, h\}, \text{QoS}_{S_4})$, $S_5 = (\{a, c\}, \{f, g, h\}, \text{QoS}_{S_4})$, and $S_6 = (\{c, d, e\}, \{f, g, h\}, \text{QoS}_{S_4})$. The two special services $Start = (\emptyset, \{a, b, e\}, \emptyset)$ and $End = (\{i\}, \emptyset, \emptyset)$ are defined by the given composition task T. Fig. 2 shows an example of discovering relevant services and service layers over a service request T, where five related services (i.e., S_0, S_1, S_2, S_3, and S_4) and two layers (i.e., $\mathcal{L}_1$ and $\mathcal{L}_2$) are found. In particular, a composition task is utilized to discover relevant Web services while the undiscovered Web services are irrelevant. In $\mathcal{L}_1$, S_0, S_1, S_2, and S_4 can be immediately satisfied by the task inputs I_T, i.e., $\{a, b\}$, of task T, and they have the same distance to *Start* (Note that the distance is measured by the number of predecessors). Different from the services in $\mathcal{L}_1$, S_3 in $\mathcal{L}_2$ requires additional inputs provided by services in $\mathcal{L}_1$, with a longer distance to *Start*.

C. Permutation-Based Representation

Composite services are commonly represented as DAGs [5], [8], [10], [11], [28], [29]. Let $\mathcal{G} = (V, E)$ be a DAG-based composite solution from *Start* to *End*, where nodes correspond to the services and edges correspond to the robust causal links. Often, V does not contain all services in $\mathcal{SR}$.

Many combinatorial optimization problems naturally represent solutions as permutations, which can be different in different problems [24]. Here, we present composite services as permutations, and we ensure a bidirectional map between permutations and DAGs. The bidirectional map is crucial for learning the distribution of promising composite services. The reason is that it is less reliable to learn a distribution based on permutations if different permutations are mapped to the same DAG-based composition service. Let $\Pi = (\Pi_0, \ldots, \Pi_t, \Pi_{t+1}, \ldots, \Pi_{n-1})$ be a permutation, whose elements are $\{0, \ldots, t, t+1, \ldots, n-1\}$ such that $\Pi_i \neq \Pi_j$ for all $i \neq j$. Particularly, $\{0, \ldots, t\}$ are service indices (i.e., id number) of the component services in the corresponding $\mathcal{G}$, and are sorted based on the longest distance from *Start* to every component services of $\mathcal{G}$, whereas $\{t+1, \ldots, n-1\}$ be indices

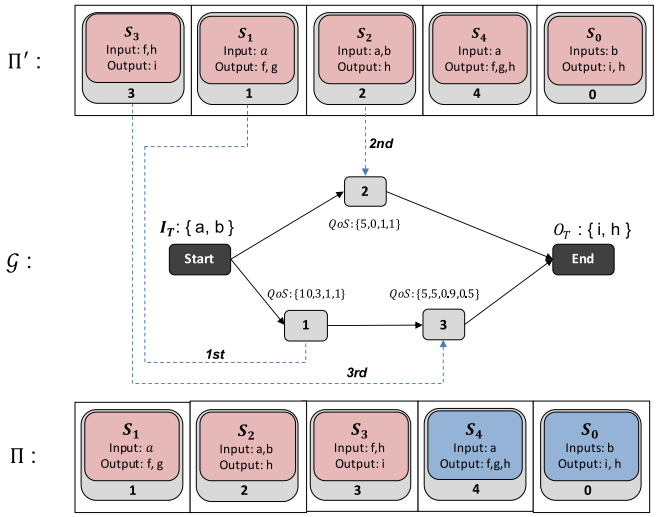

Fig. 3. Example of $\mathcal{G}$ and Π over service request T on Π'.

of remaining services in $\mathcal{SR}$ not utilized by $\mathcal{G}$. We use Π_k^g to present the kth (out of m, m is the population size) service composition solution, and $\mathcal{P}^g = [\Pi_0^g, \ldots, \Pi_k^g, \ldots, \Pi_{m-1}^g]$ to represent a population of solutions of generation g. An example of producing a permutation-based composite solution is shown as follows.

Example 2: Let us consider the *composition task* [i.e., $T = (\{a, b\}, \{i, h\})$] and the four task-relevant Web services (i.e., S_0, S_1, S_2, S_3, and S_4) in Example 1: in addition, $\text{QoS}_{S_0} = \{8, 2, 0.9, 0.7\}$, $\text{QoS}_{S_1} = \{10, 3, 1, 1\}$, $\text{QoS}_{S_2} = \{5, 0, 1, 1\}$, $\text{QoS}_{S_3} = \{5, 5, 0.9, 0.5\}$, and $\text{QoS}_{S_4} = \{8, 6, 0.84, 0.5\}$. Fig. 3 illustrates an example of producing permutation $[1, 2, 3, 0, 4]$ from a DAG, which is decoded from a given permutation $[3, 1, 2, 4, 0]$.

As an example in Fig. 3, take a permutation Π' as $[3, 1, 2, 4, 0]$. This service index queue is decoded into $\mathcal{G}$, representing a composite service that satisfies the composition task T. By determining service positions represented in a permutation, we can build a composite service in the form of a DAG using the Graphplan technique in [57]. The DAG is built in a forward way—from the start node toward the end node. Note that such a graph building process can ensure the functional correctness of a solution. Afterward, $\mathcal{G}$ is mapped to a permutation $\Pi = [1, 2, 3 \mid 0, 4]$. Herein, each position on the left side of | corresponds to a service discovered by a BFS on $\mathcal{G}$ from *Start*, whereas the right side corresponds to the remaining atomic services in $\mathcal{SR}$, but not in $\mathcal{G}$. Note that | is just displayed for the courtesy of the reader, rather than being part of the permutation-based representation. Furthermore, we also permit the encoding $[1, 2, 3 \mid 0, 4]$, as no information can be extracted from $\mathcal{G}$ to determine the order of 0 and 4.

In Fig. 3, each of the component services in $\mathcal{G}$ is marked with QoS attributes (that includes execution time, availability, cost, and reliability). The overall execution time is determined by the maximum time-consumption path, i.e., a path starting from the *start* node to node 1, then followed by node 3, ending with the *End* node. The overall availability and reliability are calculated by a product of each availability and reliability, respectively. The overall cost is calculated by a sum of each cost of component services.

A permutation-based population $\mathcal{P}^g$ can be created with m permutation-based solutions. Considering $m = 6$, $\mathcal{P}^g$ could be represented as follows:

$$\mathcal{P}^g = \begin{bmatrix} \Pi_0^g \\ \Pi_1^g \\ \Pi_2^g \\ \Pi_3^g \\ \Pi_4^g \\ \Pi_5^g \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & | & 0 & 4 \\ 0 & | & 1 & 2 & 3 & 4 \\ 0 & | & 1 & 2 & 3 & 4 \\ 4 & 3 & | & 0 & 1 & 2 \\ 4 & 3 & | & 0 & 1 & 2 \\ 2 & 1 & 3 & | & 0 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 0 & 4 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 3 & 4 \\ 4 & 3 & 0 & 1 & 2 \\ 4 & 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 0 & 4 \end{bmatrix}.$$

D. Application of Node Histogram-Based Sampling

Tsutsui [23] proposed NHBSA as a tool for sampling new candidate solutions, which is commonly represented in the form of permutations. By employing the discussed representation of composite services in Section IV-C, we are now capable of applying NHBSA to sample new permutations as candidate composite services.

The NHM at generation g, denoted by $\mathcal{NHM}^g$, is an $n \times n$-matrix with entries $e_{i,j}^g$ as follows:

$$e_{i,j}^g = \sum_{k=0}^{m-1} \delta_{i,j}\left(\Pi_k^g\right) + \varepsilon \tag{7}$$

$$\delta_{i,j}\left(\Pi_k^g\right) = \begin{cases} 1, & \text{if } \pi_i = j \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

where $i, j = 0, 1, \ldots, n-1$, and $\varepsilon = [(m)/(n-1)]b_{\text{ratio}}$ is a predetermined bias. Roughly speaking, entry $e_{i,j}^g$ counts the number of times that service index π_i appears in position j of the permutation over all permutations in population $\mathcal{P}^g$.

Example 3: Considering $\mathcal{P}^g$ in Example 2, the size of population m equals 6, the dimension size of each individual (i.e., permutation) n equals 5, and $b_{\text{ratio}} = 0.2$, we calculate $\mathcal{NHM}^g$ as follows:

$$\mathcal{NHM}^g = \begin{bmatrix} 2.3 & 1.3 & 1.3 & 0.3 & 2.3 \\ 0.3 & 3.3 & 1.3 & 2.3 & 0.3 \\ 2.3 & 0.3 & 2.3 & 2.3 & 0.3 \\ 2.3 & 2.3 & 0.3 & 2.3 & 0.3 \\ 0.3 & 0.3 & 2.3 & 0.3 & 4.3 \end{bmatrix}.$$

Consider element $e_{0,0}^g$ in the $\mathcal{NHM}^g$ as an example to demonstrate the meaning of each element in the NHM. $e_{0,0}^g$ (that equals 2.3) is made of integer and decimal parts: 2 and 0.3. The integer number 2 means that service index π_0 appears at the first position two times, while the decimal number 0.3 is a ε bias, calculated by $[(6)/(5-1)]0.2$.

Once we have computed $\mathcal{NHM}^g$, we can sample a new candidate solution Π_k^{g+1} (with $k = 0, \ldots, m-1$) from *NHM*^{g} for generation $g+1$ using NHBSA [23]. Particularly, NHBSA

Algorithm 1: Local Search Procedure

Input : $\mathcal{P}^g$, n_{nb} and n_{set}
Output: an updated $\mathcal{P}^g$

Select a small number n_{set} of individulals to form a subset *SelectedIndiSet* of $\mathcal{P}^g$ using ALGORITHM 2;
for each Π *in SelectedIndiSet* **do**
  Generate a size n_{nb} of neighbors from Π by local search;
  Identify the neighbor Π_{best} with the highest fitness;
  replace Π with Π_{best};
return an updated $\mathcal{P}^g$;

Algorithm 2: Fitness Uniform Selection Scheme

Input : $\mathcal{P}^g$ and n_{set}
Output: selected solutions *SelectedIndiSet*

SelectedIndiSet $\leftarrow \{\}$;
Sort $\mathcal{P}^g$ in descending order based on the fitness;
Put the first individual in $\mathcal{P}^g$ into *SelectedIndiSet*;
Calculate fitness range for $n_{set}-1$ groups based on a uniform *interval* between *maxfitness* and *minfitness*;
Assign each permutation in $\mathcal{P}^g$ to $n_{set}-1$ groups based on the fitness value;
Random select one permutation from each group and put it into *SelectedIndiSet*;
return *SelectedIndiSet*;

starts with sampling an element for a random position of a permutation with a probability calculated based on NHM^g. Then, it recursively continues sampling other elements for other positions in the permutation. Once a new permutation Π_k^{g+1} is returned, the same decoding part discussed in Section IV-C will be employed to ensure its functional correctness of the decoded DAG.

E. Effective Local Search Procedure

In this section, we introduce the local search procedure in MEEDA. In particular, we apply a local search to a restricted number of suitable candidate solutions, which are selected via a fitness uniform selection scheme over the current population (see details in Section IV-E1). Furthermore, for each selected solution, a stochastic local search operator is employed to create new permutations as its neighbors, where the best neighbor is identified based on the fitness value (see details in Section IV-E2).

This local search procedure, illustrated in Algorithm 1, takes three inputs: 1) the gth population $\mathcal{P}^g$; 2) the number of selected individuals for local search n_{set}; and 3) the number of neighbors n_{nb}. In this algorithm, we start by selecting a small fixed number n_{set} of candidate solutions to form a subset *SelectedIndiSet* of the current population $\mathcal{P}^g$ using Algorithm 2. The local search is performed on the solutions in *SelectedIndiSet*. For each solution Π in *SelectedIndiSet*, we produce n_{nb} neighbors from Π by local search, and then we identify the best neighbor Π_{best} from the produced neighbors. Consequently, we replace the solution Π with the best neighbor Π_{best}. Eventually, we return an updated $\mathcal{P}^g$.

1) Application of Uniform Distribution Schema: As discussed in [46], two types of selection schemes (i.e., random selection scheme and statistical scheme) have been studied for selecting suitable individuals for local search. The random selection scheme is a primary selection method that selects any individual with a predefined probability. However, it is time consuming and ineffective when the population size is huge [46]. The second statistical scheme is more capable of choosing suitable individuals based on the statistical information of the current population.

Our selection scheme is presented in Algorithm 2. This algorithm applies a local search to a set of selected individuals *SelectedIndiSet*. The size of *SelectedIndiSet*, n_{set}, is

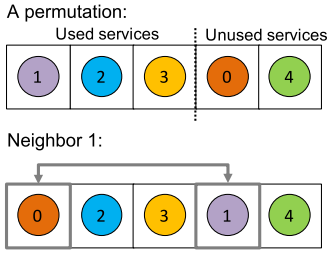

Fig. 4. Example of a constrained one-point swap on $[1, 2, 3|0, 4]$.

a predefined parameter. *SelectedIndiSet* consists of one elite individual and $n_{\text{set}}-1$ individuals from $n_{\text{set}}-1$ groups of individuals in each generation. Particularly, we calculate a fitness interval based on the maximal fitness value, *maxfitness*, and minimal fitness value, *minfitness*, of the current population $\mathcal{P}^g$. Therefore, the population is divided into $n_{\text{set}}-1$ groups based on the calculated fitness interval. Consequently, each group represent distinct importance, and individuals in a group represent similar importance. Note that for every generation, the actual number of selected individuals for local search could be less than n_{set}, whenever no individuals could fall into one group.

2) Stochastic Local Search Operators: To investigate an appropriate structure of neighborhood, suitable local search operators must be proposed by utilizing domain knowledge to guide swap operators. We can then repeatedly apply these local search operators to *SelectedIndiSet*. Apart from that, to reduce the computational time and escape local optima easily, a random subset of the whole neighborhood is explored by performing stochastic local search. Based on the proposed permutation-based representation (also called a *tidy-up permutation*) in Section IV-C, we develop four different stochastic swap operators below.

1) *Constrained One-Point Swap:* For a tidy-up permutation, $\Pi = (\pi_0, \ldots, \pi_t, \pi_{t+1}, \ldots, \pi_{n-1})$, two service indices π_a, where $0 \le a \le t$, and π_b, where $t+1 \le b \le n-1$, are selected and exchanged. To define a local search, we first define the size of the neighborhood. We define the constrained one-point swap with a predetermine fixed, small number of neighbors n_{nb} in consideration of the allowed computational time for local search. Meanwhile,

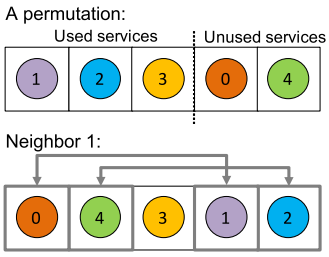

Fig. 5. Example of two-point swap on $[1, 2, 3|0, 4]$.

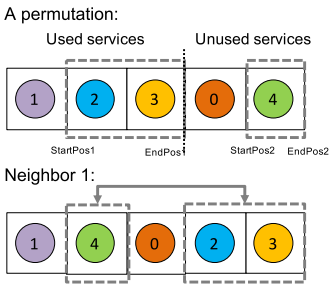

Fig. 6. Example of one constrained block-swap on $[1, 2, 3|0, 4]$.

we randomly produce n_{nb} neighbors by swapping two randomly selected indices, rather than by swapping $n-1$ indices with one fixed index. We expect that swapping two randomly selected indices is more effective. This is evidenced in Fig.10, in the supplementary material (see our discussions in the supplementary material). In addition, we constrain the pair of randomly selected indices that they must be before | and after |, respectively, in every swap to avoid considering those that have lower opportunities for local improvements. For example, one neighbor is created by swapping one pair of used service indices. This swap operation has a high chance to produce the same DAG-based solution. Fig. 4 shows an example of constrained one-point swap for a selected permutation $[1, 2, 3|0, 4]$.

2) *Constrained Two-Point Swap:* For a tidy-up permutation $\Pi = (\pi_0, \ldots, \pi_t, \pi_{t+1}, \ldots, \pi_{n-1})$, four service indices π_{a_1}, π_{a_2}, π_{b_1}, and π_{b_2} are selected, where $0 \leq a_1 \leq t$, $0 \leq a_2 \leq t$, $t+1 \leq b_1 \leq n-1$, and $t+1 \leq b_2 \leq n-1$, $a_1 \neq a_2$, and $b_1 \neq b_2$. π_{a_1} and π_{b_1} are exchanged. Likewise, π_{a_2} and π_{b_2} are exchanged. Based on the constrained one-point swap proposed above, we create a constrained two-point swap operator by combing two constrained one-point swaps into a single operator. Particularly, this operator produces only one neighbor through two consecutive constrained one-point swaps. Compared to the constrained one-point swap, constrained two-point swap is more likely to make more local changes to a candidate solution. Fig. 5 shows an example of a constrained two-point swap for a selected permutation $[1, 2, 3|0, 4]$.

3) *Constrained One-Block Swap:* For a tidy-up permutation $\Pi = (\pi_0, \ldots, \pi_t, \pi_{t+1}, \ldots, \pi_{n-1})$, two subblocks $\{\pi_a, \ldots, \pi_t\}$, where $0 \leq a < t$ and $\{\pi_b, \ldots, \pi_{n-1}\}$, where $t+1 \leq b < n-1$, are selected and exchanged. The constrained one-block swap operates on blocks which are consecutive service indices in a permutation. In this swap, two blocks are built up, starting with two randomly selected points, π_a (i.e., a point must be selected before |) and π_b (i.e., a point must be selected after |), on a permutation, respectively. Fig. 6 shows an example of a constrained one-block swap for a permutation $[1, 2, 3|0, 4]$, where one block is built up from the start position *StartPos*1 to the last position of used services, and another block is built up from the start position *StartPos*2 to the last position of the permutation.

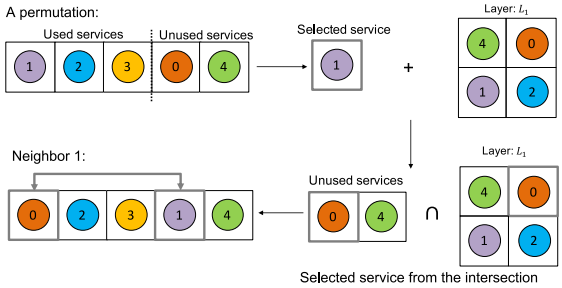

Fig. 7. Example of layer-based one-point swap operation on $[1, 2, 3|0, 4]$.

4) *Layer-Based Constrained One-Point Swap:* For a tidy-up permutation $\Pi = (\pi_0, \ldots, \pi_t, \pi_{t+1}, \ldots, \pi_{n-1})$, one service index π_a, where $0 \leq a \leq t$, is selected, and one layer $\mathcal{L}'$ *s.t.* $\pi_a \in \mathcal{L}'$ is identified. Afterward, another service index π_b is randomly selected from the index set $\mathcal{L}' \cap \{\pi_{t+1}, \ldots, \pi_{n-1}\}$. Consequently, π_a and π_b are exchanged. A layer-based one-point-constrained swap is proposed by extending the constrained one-point swap while considering the layer information, introduced in Section IV-B. The benefit of considering layers allows us to identify a suitable pair of service indices (i.e., services have similar input requirements) for a swap, compared to a constrained one-point swap. This is because such a pair of service indices not only comes from two different parts of a permutation (i.e., before and after |) but also comes from the same layer. By doing these, a layer-based one-point-constrained swap operator is more likely to produce different neighboring solutions in the DAG form. Fig. 7 shows an example of layer-based constrained one-point swap for creating one neighbor from a selected permutation $[1, 2, 3|0, 4]$.

An example in Fig. 8 illustrates the difference between layer-based constrained one-point swap and constraint one-point swap. In the example, one identical solution can be decoded from both a permutation and two of its neighbors. This indicates that the constrained one-point swap does not properly exploit the neighbors of the permutations. In contrast, these two swaps are not permitted in the layer-based constraint one-point swap, where any produced neighbor must strictly follow the layer order on the permutation before |.

In Fig. 8, the permutation $[1, 2, 3|4, 0]$ is highlighted with two layers (i.e., $\mathcal{L}_1$ and $\mathcal{L}_2$) in ascending order. Particularly, $S_1, S_2 \in \mathcal{L}_1$ and $S_3 \in \mathcal{L}_2$. When the constrained one-point swap is performed, S_3 in the permutation are replaced with S_4

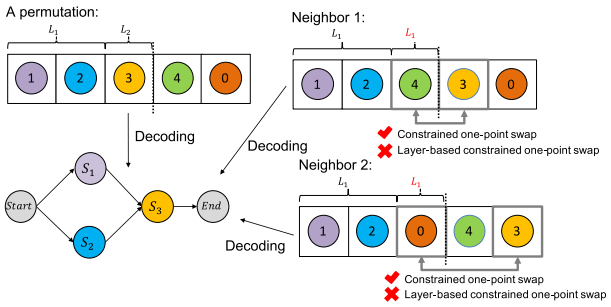

Fig. 8. Example of layer order breached by constrained one swap operation.

or S_0 in the produced neighbor 1 and neighbor 2, respectively. However, $\mathcal{L}_2$ is destroyed in the produced neighbors because of $S_4 \in \mathcal{L}_1$ and $S_0 \in \mathcal{L}_1$. Apparently, the layer-based constrained one-point swap can prevent these two neighbors from being exploited.

V. Experiments

We conduct experiments to evaluate the performances of our MEEDA algorithm with different local search operators, i.e., MEEDA with constrained one-point swap (henceforth referred to as MEEDA-OP), MEEDA with constrained two-point swap (henceforth referred to as MEEDA-TP), MEEDA with constrained layer-based one-point swap (henceforth referred to as MEEDA-LOP), and MEEDA with constrained one-block swap (henceforth referred to as MEEDA-OB). These MEEDA algorithms are compared to some state-of-the-art methods that were recently proposed to solve the same or similar problems: a PSO-based approach [10] (henceforth referred to as PSO), a fixed-length GA-based approach [9] (henceforth referred to as GA), a memetic GA-based approach [9] (henceforth referred to as MEGA), an EDA-based approach [12] (henceforth referred to as EDA), and a non-EC search-based approach [15] (henceforth referred to as PathSearch).

Note that due to the page limit, please refer to our supplemental materia for some important experimental analysis, which includes comparison of the convergence rate, futher analysis of local search operators, and further analysis of two solution examples.

A. Experimental Design

For all the EDA-based approaches, the population size is set to 200, the number of generations equals 100, and b_{ratio} is 0.0002. The size of *SelectedIndiSet* is 6. Meanwhile, the local search operators n_{nb} are allowed to explore up to 20 neighboring solutions with respect to each individual in *SelectedIndiSet* chosen for local search. Using one test case associated with dataset WSC-09, we have fine tuned a pair of parameters, i.e., *SelectedIndiSet* and n_{nb}, with a range of candidate values. For example, pairs of parameter values {4, 30}, {6, 20}, and {8, 15}. We subsequently adopted the parameter settings that produced the best performance on the chosen test case for all subsequent experiments. For other competing methods, including PSO, GA, and MEGA, we use identical population size and generation number as EDA-based approaches. Other parameter settings follow strictly the best reported settings in [9], [10], and [15]. For example, in GA and MEGA, the crossover rate is set to 0.95, and the mutation rate is set to 0.05. In PathSearch, the parameter K (i.e., number of services considered in the path construction at each step) associated with this algorithm is set to 7, which maximizes the performance of PathSearch in their paper. Following the existing works [10]–[12], the weights of the fitness function (5) are simply configured to balance the QoSM and QoS. In particular, we set both w_1 and w_2 to 0.25, and set w_3, w_4, w_5, and w_6 all to 0.125. More experiments have been conducted and show that all our methods work consistently well under different weight settings. The p of $\text{type}_{\text{link}}$ in (1) is determined by the preference of users, and is recommended as 0.75 for the plugin match according to [51].

Two benchmarks WSC-08 [34] and WSC-09 [33] extended with QoS attributes, which are generated from the QoS distribution from QWS [35]–[37], are created. These two benchmarks have already been broadly employed in service composition [5], [10], [13] for experimental evaluations. In WSC08 and WSC09, the semantics of service inputs and outputs are described by OWL-S language. This language allows a high degree of automation in discovering, invoking, composing, and monitoring Web resources. The number of Web services in the service repository is doubled (with much bigger searching space) in order to examine whether our MEEDA algorithm can solve problems with significantly larger sizes. To double the size of WSC08 and WSC09, we clone each service in WSC08 and WSC09 with the same inputs and outputs, but with different QoS values. This augmented datasets provide more services that can be selected for each vertex of the DAG-based solutions. We also make these datasets available to the public. Particularly, WSC08 contains eight composition tasks with increasing size of service repository, i.e., 316, 1116, 1216, 2082, 2180, 4396, 8226, and 16238, and WSC09 contains five composition tasks with the increasing size of the service repository, i.e., 1144, 8258, 16276, 16602, and 30422 $\mathcal{SR}$s, respectively. We run the experiment with 30 independent repetitions for EC-based approaches, including EDA, MEEDA-LOP, MEEDA-TP, MEEDA-OP, MEEDA-OB, EDA, GA, MEGA, and PSO. All the methods are run on a PC with an Intel Core i7-4770 CPU (3.4 GHz) and 8-GB RAM. This hardware configuration will also be used for all the methods presented in this article.

B. Comparison of the Fitness

We employ the independent-sample T-test and Wilcoxon rank-sum test for normal and nonnormal distribution performance observations (i.e., execution time and fitness value), respectively. In particular, we use a significance level of 5% in all the tests to verify the observed differences in Sections V-B and V-C. In particular, we use a pairwise comparison to compare all competing approaches. Then, the top performances are identified, and its related value is highlighted in green color in Table II. Those methods that consistently find the best-known solutions over 30 runs with 0 standard deviations are also marked as top performances. The pairwise comparison results for fitness are summarized in Table III, where *win/draw/loss* shows the scores of one method compared to all the others and displays the frequency that this

TABLE III
SUMMARY OF STATISTICAL SIGNIFICANCE TESTS FOR FITNESS, WHERE EACH COLUMN SHOWS WIN/DRAW/LOSS SCORE OF AN APPROACH AGAINST OTHERS

Dataset	Method	MEEDA-OP	MEEDA-TP	MEEDA-OB	MEEDA-LOP	EDA	PSO	MEGA	GA	PathSearch
WSC-08 (8 tasks)	MEEDA-OP	-	0/8/0	0/6/2	1/7/0	0/6/2	0/1/7	1/4/3	0/1/7/	0/0/8
	MEEDA-TP	0/8/0	-	1/5/2	1/6/1	0/6/2	0/1/7	1/4/3	0/1/7	0/0/8
	MEEDA-OB	2/6/0	2/5/1	-	1/7/0	0/8/0	0/1/7	2/5/1	0/1/7	0/0/8
	MEEDA-LOP	0/7/1	1/6/1	0/7/1	-	0/5/3	0/1/7	1/4/3	0/1/7	0/0/8
	EDA	2/6/0	2/6/0	0/8/0	3/5/0	-	1/7/0	1/6/1	0/2/6	0/0/8
	PSO	7/1/0	7/1/0	7/1/0	7/1/0	7/1/0	-	8/0/0	8/0/0	0/0/8
	MEGA	3/4/1	3/4/1	1/5/2	3/4/1	1/6/1	0/0/8	-	0/0/8	0/0/8
	GA	7/1/0	7/1/0	7/1/0	7/1/0	6/2/0	0/0/8	8/0/0	-	0/0/8
	PathSearch	8/0/0	8/0/0	8/0/0	8/0/0	8/0/0	8/0/0	8/0/0	8/0/0	-
WSC-09 (5 tasks)	MEEDA-OP	-	0/5/0	0/4/1	1/4/0	0/4/1	0/2/3	0/3/1	0/3/2	0/0/5
	MEEDA-TP	0/5/0	-	0/4/1	2/3/0	0/4/1	0/1/4	1/2/1	0/3/2	0/0/5
	MEEDA-OB	1/4/0	1/4/0	-	2/3/0	0/5/0	0/3/2	1/2/1	0/3/2	0/0/5
	MEEDA-LOP	0/4/1	0/3/2	0/3/2	-	0/2/3	0/2/3	0/3/1	0/2/3	0/0/5
	EDA	1/4/0	1/4/0	0/5/0	3/2/0	-	0/1/4	2/1/2	0/3/2	0/0/5
	PSO [10]	2/2/0	4/1/0	2/3/0	3/2/0	4/1/0	-	3/1/0	2/2/1	0/0/5
	MEGA [9]	2/3/0	2/2/1	2/2/1	2/3/0	2/1/2	1/1/3	-	1/2/2	0/0/5
	GA	2/3/0	2/3/0	2/3/0	3/2/0	2/3/0	1/2/2	2/2/0	-	0/0/5
	PathSearch	5/0/0	5/0/0	5/0/0	5/0/0	5/0/0	5/0/0	5/0/0	5/0/0	-

TABLE IV
MEAN EXECUTION TIME (IN S) FOR OUR APPROACH IN COMPARISON TO EDA, PSO, MEGA, GA, AND PATHSEARCH. (NOTE: THE SHORTER THE TIME THE BETTER)

Dataset	MEEDA-OP	MEEDA-TP	MEEDA-OB	MEEDA-LOP	EDA	PSO	MEGA	GA	PathSearch
WSC08-1	156 ± 12	211 ± 19	422 ± 70	112 ± 8	72 ± 6	111 ± 76	622 ± 74	102 ± 11	3
WSC08-2	72 ± 9	98 ± 13	133 ± 14	72 ± 6	32 ± 4	68 ± 48	118 ± 17	21 ± 3	6
WSC08-3	8470 ± 462	8807 ± 621	12849 ± 740	8329 ± 346	1296 ± 137	15789 ± 2602	68382 ± 13142	12514 ± 1575	18
WSC08-4	87 ± 6	112 ± 7	177 ± 7	84 ± 6	38 ± 4	211 ± 104	766 ± 253	147 ± 36	6
WSC08-5	1705 ± 148	2136 ± 142	4363 ± 154	1742 ± 122	979 ± 96	2549 ± 1517	47603 ± 47104	3801 ± 1512	19
WSC08-6	17524 ± 843	19954 ± 1352	43621 ± 2062	17303 ± 1569	13964 ± 1735	33119 ± 12194	947368 ± 157828	51287 ± 11561	101
WSC08-7	2025 ± 138	2869 ± 1258	8096 ± 448	1918 ± 119	1181 ± 115	4456 ± 2825	81847 ± 20610	5499 ± 1526	90
WSC08-8	4375 ± 371	5066 ± 350	11341 ± 666	4283 ± 368	2677 ± 308	6153 ± 1951	148133 ± 29304	10931 ± 1667	273
WSC09-1	159 ± 23	239 ± 38	314 ± 27	159 ± 18	62 ± 7	126 ± 139	506 ± 104	65 ± 12	7
WSC09-2	3314 ± 551	4311 ± 686	7573 ± 553	3362 ± 505	2204 ± 378	3652 ± 1516	49455 ± 17831	4081 ± 1433	138
WSC09-3	1643 ± 146	2303 ± 191	4638 ± 343	1614 ± 124	819 ± 65	2198 ± 2038	18998 ± 3300	1713 ± 394	233
WSC09-4	92342 ± 7584	103433 ± 6847	214067 ± 12358	86543 ± 6046	70105 ± 6772	85813 ± 37895	-	176152 ± 46321	1154
WSC09-5	16160 ± 1123	18446 ± 1776	45039 ± 4534	15249 ± 978	11117 ± 1150	14807 ± 5605	635637 ± 151975	29991 ± 4867	915

TABLE V
SUMMARY OF STATISTICAL SIGNIFICANCE TESTS FOR EXECUTION TIME (IN S), WHERE EACH COLUMN SHOWS WIN/DRAW/LOSS SCORE OF AN APPROACH AGAINST OTHERS

Dataset	Method	MEEDA-OP	MEEDA-TP	MEEDA-OB	MEEDA-LOP	EDA	PSO	MEGA	GA	PathSearch
WSC-08 (8 tasks)	MEEDA-OP	-	0/0/8	0/0/8	6/2/0	8/0/0	1/1/6	0/0/8	2/0/6	8/0/0
	MEEDA-TP	8/0/0	-	0/0/8	8/0/0	8/0/0	2/1/5	0/0/8	2/0/6	8/0/0
	MEEDA-OB	8/0/0	8/0/0	-	8/0/0	8/0/0	5/1/2	0/0/8	4/3/1	8/0/0
	MEEDA-LOP	0/2/6	0/0/8	0/0/8	-	8/0/0	0/2/6	0/0/8	2/0/6	8/0/0
	EDA	0/0/8	0/0/8	0/0/8	0/0/8	-	0/0/8	0/0/8	0/1/7	8/0/0
	PSO	6/1/1	5/1/2	2/1/5	6/2/0	8/0/0	-	0/0/8	2/4/2	8/0/0
	MEGA	8/0/0	8/0/0	8/0/0	8/0/0	8/0/0	8/0/0	-	8/0/0	8/0/0
	GA	6/0/2	6/0/2	1/3/4	6/0/2	7/1/0	2/4/2	0/0/8	-	8/0/0
	PathSearch	0/0/8	0/0/8	0/0/8	0/0/8	0/0/8	0/0/8	0/0/8	0/0/8	-
WSC-09 (5 tasks)	MEEDA-OP	-	0/0/5	0/0/5	2/3/0	5/0/0	0/1/4	0/0/4	3/1/1	5/0/0
	MEEDA-TP	5/0/0	-	0/0/5	5/0/0	5/0/0	4/1/0	0/0/4	4/1/0	5/0/0
	MEEDA-OB	5/0/0	5/0/0	-	5/0/0	5/0/0	5/0/0	0/0/4	5/0/0	5/0/0
	MEEDA-LOP	0/3/2	0/0/5	0/0/5	-	5/0/0	0/5/0	0/0/4	1/1/3	5/0/0
	EDA	3/2/0	2/1/2	0/0/5	3/2/0	-	3/2/0	0/0/4	1/2/2	5/0/0
	PSO	4/1/1	0/1/4	0/0/5	0/5/0	5/0/0	-	0/0/4	1/2/2	5/0/0
	MEGA	4/0/0	4/0/0	4/0/0	4/0/0	5/0/0	4/0/0	-	4/0/0	5/0/0
	GA	1/1/3	0/1/4	0/0/5	3/1/1	5/0/0	2/2/1	0/0/4	-	5/0/0
	PathSearch	0/0/5	0/0/5	0/0/5	0/0/5	0/0/5	0/0/5	0/0/5	0/0/5	-

method outperforms, equals, or is outperformed by the competing method. This testing and comparison methods are also used in Section V-C. Note that any "−" in the tables means that results cannot be collected when the corresponding method has been running for seven days.

One of the experiments' objectives is to evaluate MEEDA algorithms' effectiveness compared to EDA, PSO, GA, MEGA, and PathSearch. Table II shows the mean value of the fitness and the standard deviation for all EC-based approaches over 30 repetitions and the fitness value obtained by PathSearch over one run. The pairwise comparison results of the fitness value are summarized in Table III. From Tables II and III, we observe some interesting behaviors of these approaches in finding high-quality solutions. Based on these observations, we can conclude the observations about the effectiveness of these methods as follows.

First, all the memetic EDA algorithms, including MEEDA-OP, MEEDA-TP, MEEDA-OB, and MEEDA-LOP, significantly outperform the baseline method EDA. This observation corresponds well with our expectation that the exploitation ability of EDA can be enhanced by hybridizing it with local search.

Second, all EDA-based approaches (with and without local search) consistently outperform three baseline methods that include PSO, GA, and PathSearch. This observation indicates that all EDA-based approaches are more competent at improving the quality of composite services by effectively utilizing the knowledge via NHMs, compared to other methods. In contrast, PathSearch achieves the worst performance in finding high-quality solutions. This is because PathSearch is designed to make a locally best choice over the K services at each step, toward a gradually built path-based composite service.

In addition, MEEDA-LOP is identified as the best performer in finding high-quality composite services. This observation corresponds well with our assumption that the layer-based constrained one-point swap is more effective than other swap operators. Meanwhile, MEEDA-LOP has achieved extremely stable performance in most runs with 0 standard deviations.

Furthermore, MEEDA-OP and MEEDA-TP outperform MEGA, while MEEDA-OB and MEGA are comparable to each other. This is because EDA (i.e., the baseline method of MEEDA-OP, MEEDA-TP) is more effective than GA (i.e., the baseline method of MEGA). The effectiveness of MEEDA-OB

TABLE II
MEAN FITNESS VALUES FOR OUR APPROACH IN COMPARISON TO EDA, PSO, MEGA, GA, AND PATHSEARCH. (NOTE: THE HIGHER THE FITNESS THE BETTER)

Dataset	MEEDA-OP	MEEDA-TP	MEEDA-OB	MEEDA-LOP	EDA	PSO	MEGA	GA	PathSearch
WSC08-1	0.613745 ± 0	0.613745 ± 0	0.613745 ± 0	0.613745 ± 0	0.613745 ± 0	0.610182 ± 0.003748	0.613745 ± 0	0.613439 ± 0.000693	0.607149
WSC08-2	0.756812 ± 0	0.756812 ± 0	0.756812 ± 0	0.756812 ± 0	0.756812 ± 0	0.756779 ± 0.000175	0.756812 ± 0	0.756812 ± 0	0.597588
WSC08-3	0.477864 ± 5.1e − 05	0.477866 ± 4e − 05	0.477803 ± 6.7e − 05	0.47791 ± 1.7e − 05	0.477802 ± 5.2e05	0.476086 ± 0.000528	0.477768 ± 0.00015	0.477447 ± 0.000228	0.473133
WSC08-4	0.557815 ± 0	0.557815 ± 0	0.557815 ± 0	0.557815 ± 0	0.557815 ± 0	0.557416 ± 0.000666	0.557815 ± 0	0.55781 ± 3.1e − 05	0.549285
WSC08-5	0.52453 ± 0.00038	0.52474 ± 0.000388	0.524482 ± 0.000369	0.524412 ± 0.000329	0.524492 ± 0.000415	0.517912 ± 0.00544	0.525586 ± 0.000639	0.523463 ± 0.002078	0.455233
WSC08-6	0.482698 ± 0.000107	0.482684 ± 0.000143	0.482622 ± 0.000131	0.482703 ± 0.000225	0.482671 ± 0.000139	0.481723 ± 0.000505	0.482759 ± 0.000253	0.482142 ± 0.000452	0.414705
WSC08-7	0.523588 ± 0	0.523588 ± 0	0.523577 ± 5.8e − 05	0.523588 ± 0	0.523584 ± 2.2e05	0.516141 ± 0.00519	0.523559 ± 0.00014	0.521694 ± 0.002248	0.471800
WSC08-8	0.497676 ± 1.3e − 05	0.497676 ± 1.3e − 05	0.497646 ± 9.4e − 05	0.497679 ± 0	0.497638 ± 0.000126	0.489581 ± 0.003219	0.497455 ± 0.000357	0.496347 ± 0.000861	0.445169
WSC09-1	0.64902 ± 0.003747	0.650861 ± 0.00383	0.64852 ± 0.003005	0.650444 ± 0.00412	0.647988 ± 0.002939	0.648605 ± 0.004328	0.650737 ± 0.003863	0.649612 ± 0.003688	0.597177
WSC09-2	0.521817 ± 0.003855	0.520562 ± 0.004943	0.520966 ± 0.008258	0.52309 ± 0.000443	0.517561 ± 0.012522	0.506701 ± 0.011045	0.52257 ± 0.003246	0.521424 ± 0.003011	0.464576
WSC09-3	0.583978 ± 0	0.583978 ± 0	0.583978 ± 0	0.583978 ± 0	0.583978 ± 0	0.583358 ± 0.001182	0.583978 ± 0	0.583954 ± 9.1e − 05	0.455317
WSC09-4	0.484428 ± 0.000191	0.484427 ± 0.00032	0.484295 ± 0.00014	0.485533 ± 0.002073	0.484315 ± 0.000104	0.481741 ± 0.000985	-	0.483277 ± 0.000367	0.467477
WSC09-5	0.484832 ± 2.6e − 05	0.484818 ± 9.5e − 05	0.484788 ± 0.000179	0.484825 ± 0.000117	0.484818 ± 4e − 05	0.480539 ± 0.001308	0.484603 ± 0.000294	0.483278 ± 0.001185	0.468230

does not meet our expectations. This is because swapping building blocks can potentially ruin the learned knowledge of promising solutions, resulting in poor searching behavior.

In summary, we sort all the competing approaches based on the effectiveness in descending order: MEEDA-LOP > MEGA > MEEDA-TP = MEEDA-OP > MEEDA-OB > EDA > GA > PSO > PathSearch. Apparently, the layer-based constrained one-point swap operator is the most effective swap for enhancing our EDA-based approach in terms of effectiveness.

C. Comparison of the Execution Time

The second objective of our experiment is to study the efficiency of MEEDA algorithms compared to EDA, PSO, GA, MEGA, and PathSearch. Table IV shows the mean value of the execution time and the standard deviation over 30 repetitions for all EC-based approaches, and execution time consumed by PathSearch over one run. The pairwise comparison results for the execution time are summarized Table V. From the two tables above, we make some analysis and possible conclusions about the execution time of these approaches as below.

First, PathSearch requires the least execution time. This is because PathSearch prunes prestored service dependency graphs, on which it only searches K best services at each step, toward a gradually built path (i.e., a composite service). Despite the highest efficiency achieved by PathSearch, the effectiveness of this method is the worst. In contrast, MEGA requires the highest execution time because its local search is performed on all the candidate solutions based on a predefined probability. Besides, its swap operator exclusively searches the whole neighborhood of candidate solutions. This poor execution time confirms that the local search strategy in MEGA is very time consuming.

Second, in most instances of WSC-08 and WSC-09, we observe that EDA consumes much less execution time compared to other EC-based approaches without local search. This might be due to two reasons. The first reason is that solutions evolved by EDA are likely to have all useful services required to build a suitable DAG placed at the very front of the service queue. The second reason is that the archive utilized in EDA stores promising solutions, saving execution time from preventing many promising solutions to be evaluated.

Third, among all the memetic methods, we observe that MEEDA-LOP requires consistently less execution time than other memetic approaches. This remarkable observation further confirms the best effectiveness of MEEDA-LOP, resulting in sampled permutations that are likely to have useful services to be put in the front. Such permutations can be decoded in DAGs much faster than those produced by other approaches.

Finally, MEEDA-OB is very computation-intensive among all the memetic EDA-based approaches. It is due to that one-block swap retards accurate distributions to be learned as local improvements of one-block swap is less effective, so required services for service composition are less likely to be put at the front of a service queue. In addition, building blocks in one-block swaps consume extra time in MEEDA-OB.

In summary, we sort all the competing approaches based on the execution time in ascending order: PathSearch > EDA >

MEEDA-LOP > MEEDA-OP > MEEDA-TP > PSO > GA > MEEDA-OB > MEGA. Despite the least execution time consumed by PathSearch, PathSearch cannot outperform any memetic method. Therefore, MEEDA-LOP becomes the most suitable method since it consumes the least execution time in all the memetic algorithms and achieves the best effectiveness.

VI. Conclusion

In this article, we proposed effective and efficient memetic EDA-based approaches to QoS-aware fully automated semantic service composition. In particular, we proposed several neighborhood structures of composite services by different local search operators. Meanwhile, a uniform distribution scheme and a stochastic strategy were jointly utilized to select a small set of suitable candidate solutions for performing an effective and efficient local search. The experiments showed that MEEDA-LOP achieves the best performance significantly, compared to some state-of-the-art EC-based and non-EC-based approaches in this article. Future work can investigate variable neighborhoods with combinations of more than one local search operators in one evolutionary process and study memetic EDA to handle multiobjective service composition problems.

References

[1] A. Arsanjani, *Service-Oriented Modeling and Architecture*, vol. 1, IBM, Armonk, NY, USA, 2004, p. 15.

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architectures and Applications*, Springer, 2004,

[3] A. S. da Silva, H. Ma, Y. Mei, and M. Zhang, "A survey of evolutionary computation for Web service composition: A technical perspective," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 4, no. 4, pp. 538–554, Aug. 2020.

[4] V. Gabrel, M. Manouvrier, and C. Murat, "Web services composition: Complexity and models," *Discr. Appl. Math.*, vol. 196, pp. 100–114, Dec. 2015.

[5] H. Ma, A. Wang, and M. Zhang, "A hybrid approach using genetic programming and greedy search for QoS-aware Web service composition," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XVIII*, vol. 18. Heidelberg, Germany: Springer, 2015, pp. 180–205.

[6] S. Peng, H. Wang, and Q. Yu, "Estimation of distribution with restricted Boltzmann machine for adaptive service composition," in *Proc. Int. Conf. Web Services*, 2017, pp. 114–121.

[7] P. Rodriguez-Mier, M. Mucientes, M. Lama, and M. I. Couto, "Composition of Web services through genetic programming," *Evol. Intell.*, vol. 3, nos. 3–4, pp. 171–186, 2010.

[8] A. S. Da Silva, H. Ma, and M. Zhang, "Genetic programming for QoS-aware Web service composition and selection," *Soft Comput.*, vol. 20, no. 10, pp. 3851–3867, 2016.

[9] A. S. da Silva, Y. Mei, H. Ma, and M. Zhang, "Evolutionary computation for automatic Web service composition: An indirect representation approach," *J. Heurist.*, vol. 24, no. 3, pp. 425–456, 2018.

[10] C. Wang, H. Ma, A. Chen, and S. Hartmann, "Comprehensive quality-aware automated semantic Web service composition," in *Advances in Artificial Intelligence*. Cham, Switzerland: Springer, 2017, pp. 195–207.

[11] C. Wang, H. Ma, G. Chen, and S. Hartmann, "GP-based approach to comprehensive quality-aware automated semantic Web service composition," in *Simulated Evolution and Learning*. Cham, Switzerland: Springer, 2017, pp. 170–183.

[12] C. Wang, H. Ma, G. Chen, and S. Hartmann, "Knowledge-driven automated Web service composition—An EDA-based approach," in *Web Information Systems Engineering (WISE)*. Cham, Switzerland: Springer, 2018, pp. 135–150.

[13] Y. Yu, H. Ma, and M. Zhang, "An adaptive genetic programming approach to QoS-aware Web services composition," in *Proc. IEEE CEC*, 2013, pp. 1740–1747.

[14] J. Rao and X. Su, "A survey of automated Web service composition methods," in *Proc. Int. Workshop Semant. Web Services Web Process Composition*, 2004, pp. 43–54.

[15] S. Chattopadhyay, A. Banerjee, and N. Banerjee, "A fast and scalable mechanism for Web service composition," *ACM Trans. Web*, vol. 11, no. 4, p. 26, 2017.

[16] J. Peer, *Web Service Composition as AI Planning: A Survey*. St. Gallen, Switzerland: Univ. St. Gallen, 2005.

[17] L. Qi, Y. Tang, W. Dou, and J. Chen, "Combining local optimization and enumeration for QoS-aware Web service composition," in *Proc. Int. Conf. Web Services*, 2010, pp. 34–41.

[18] C. Wang, H. Ma, and G. Chen, "EDA-based approach to comprehensive quality-aware automated semantic Web service composition," in *Proc. Genet. Evol. Comput. Conf. Compan.*, 2018, pp. 147–148.

[19] H. Tong, J. Cao, S. Zhang, and M. Li, "A distributed algorithm for Web service composition based on service agent model," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 12, pp. 2008–2021, Dec. 2011.

[20] D. Whitley, "A genetic algorithm tutorial," *Stat. Comput.*, vol. 4, no. 2, pp. 65–85, 1994.

[21] J. R. Koza and J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT press, 1992.

[22] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Netw.*, vol. 4, 1995, pp. 1942–1948.

[23] S. Tsutsui, "A comparative study of sampling methods in node histogram models with probabilistic model-building genetic algorithms," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, vol. 4, 2006, pp. 3132–3137.

[24] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano, "A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems," *Progr. Artif. Intell.*, vol. 1, no. 1, pp. 103–117, 2012.

[25] K. Pichanaharee and T. Senivongse, "QoS-based service provision schemes and plan durability in service composition," in *Proc. IFIP Int. Conf. Distrib. Appl. Interoperable Syst.*, 2008, pp. 58–71.

[26] S.-Y. Wang and L. Wang, "An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem," *IEEE Trans. Syst.*, vol. 46, no. 1, pp. 139–149, Jan. 2016.

[27] J. Wang, K. Tang, J. A. Lozano, and X. Yao, "Estimation of the distribution algorithm with a stochastic local search for uncertain capacitated arc routing problems," *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 96–109, Feb. 2016.

[28] A. S. Da Silva, H. Ma, and M. Zhang, "GraphEvol: A graph evolution technique for Web service composition," in *Database and Expert Systems Applications*. Cham, Switzerland: Springer, 2015, pp. 134–142.

[29] A. S. Da Silva, Y. Mei, H. Ma, and M. Zhang, "Particle swarm optimisation with sequence-like indirect representation for Web service composition," in *Proc. Eur. Conf. Evol. Comput. Combinatorial Optim.*, 2016, pp. 202–218.

[30] S. Acid and L. M. de Campos, "Searching for Bayesian network structures in the space of restricted acyclic partially directed graphs," *J. Artif. Intell. Res.*, vol. 18, no. 1, pp. 445–490, 2003.

[31] P. Lacomme, C. Prins, and W. Ramdane-Cherif, "Competitive memetic algorithms for arc routing problems," *Ann. Oper. Res.*, vol. 131, nos. 1–4, pp. 159–185, Oct. 2004.

[32] A. S. da Silva, Y. Mei, H. Ma, and M. Zhang, "A memetic algorithm-based indirect approach to Web service composition," in *Proc. IEEE Congr. Evol. Comput.*, 2016, pp. 3385–3392.

[33] S. Kona, A. Bansal, M. B. Blake, S. Bleul, and T. Weise, "WSC-2009: A quality of service-oriented Web services challenge," in *Proc. IEEE Conf. Commer. Enterprise Comput.*, 2009, pp. 487–490.

[34] A. Bansal, M. B. Blake, S. Kona, S. Bleul, T. Weise, and M. C. Jaeger, "WSC-08: Continuing the Web services challenge," in *Proc. 10th IEEE Conf. E-Commerce Technol. 5th IEEE Conf. Enterprise Comput. E-Commer. E-Services*, 2008, pp. 351–354.

[35] E. Al-Masri and Q. H. Mahmoud, "QoS-based discovery and ranking of Web services," in *Proc. Int. Conf. Comput. Commun. Netw.*, 2007, pp. 529–534.

[36] E. Al-Masri and Q. H. Mahmoud, "Investigating Web services on the world wide Web," in *Proc. 17th Int. Conf. World Wide Web*, 2008, pp. 795–804.

[37] E. Al-Masri and Q. H. Mahmoud, "Discovering the best Web service," in *Proc. 16th Int. Conf. World Wide Web*, 2007, pp. 1257–1258.

[38] S. Chattopadhyay, A. Banerjee, and N. Banerjee, "A scalable and approximate mechanism for Web service composition," in *Proc. IEEE Int. Conf. Web Services*, 2015, pp. 9–16.

[39] M. Chen and Y. Yan, "QoS-aware service composition over graphplan through graph reachability," in *Proc. IEEE Int. Conf. Services Comput.*, 2014, pp. 544–551.
[40] S.-L. Fan, Y.-B. Yang, and X.-X. Wang, "Efficient Web service composition via knapsack-variant algorithm," in *Proc. Int. Conf. Services Comput.*, 2018, pp. 51–66.
[41] P. Hennig and W.-T. Balke, "Highly scalable Web service composition using binary tree-based parallelization," in *Proc. Int. Conf. Web Services*, 2010, pp. 123–130.
[42] W. Jiang, C. Zhang, Z. Huang, M. Chen, S. Hu, and Z. Liu, "QSynth: A tool for QoS-aware automatic service composition," in *Proc. IEEE Int. Conf. Web Services*, 2010, pp. 42–49.
[43] Y. Yan and M. Chen, "Anytime QoS-aware service composition over the GraphPlan," *Service Oriented Comput. Appl.*, vol. 9, no. 1, pp. 1–19, 2015.
[44] R. Stuart and N. Peter, *Artificial Intelligence: A Modern Approach*, S. Russell and P. Nowig, Eds. Upper Saddle River, NJ, USA: Prentice Hall, 2003.
[45] P. Rodriguez-Mier, M. Mucientes, and M. Lama, "Automatic Web service composition with a heuristic-based search algorithm," in *Proc. Int. Conf. Web Services*, 2011, pp. 81–88.
[46] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 591–607, Oct. 2011.
[47] Y. Yu, H. Ma, and M. Zhang, "A hybrid GP-Tabu approach to QoS-aware data intensive Web service composition," in *Proc. Asia-Pac. Conf. Simulat. Evol. Learn.*, 2014, pp. 106–118.
[48] C. Mao, J. Chen, and X. Yu, "An empirical study on meta-heuristic search-based Web service composition," in *Proc. Int. Conf. e-Bus. Eng.*, 2012, pp. 117–122.
[49] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality driven Web services composition," in *Proc. 12th Int. Conf. World Wide Web*, 2003, pp. 411–421.
[50] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic matching of Web services capabilities," in *Proc. Int. Semantic Web Conf.*, 2002, pp. 333–347.
[51] F. Lécué, "Optimizing QoS-aware semantic Web service composition," in *Int. Semant. Web Conf.*, 2009, pp. 375–391.
[52] F. Lécué, A. Delteil, and A. Léger, "Optimizing causal link based Web service composition," in *Proc. Eur. Coordinating Committee Artif. Intell.*, 2008, pp. 45–49.
[53] M. K. Shenoy, K. Shet, and U. D. Acharya, "A new similarity measure for taxonomy based on edge counting," *Int. J. Web Semant. Technol.*, vol. 3, no. 4, p. 23, 2012.
[54] H. Ma, K.-D. Schewe, B. Thalheim, and Q. Wang, "A formal model for the interoperability of service clouds," *Service Oriented Comput. Appl.*, vol. 6, no. 3, pp. 189–205, 2012.
[55] C.-L. Hwang and K. Yoon, *Multiple Objective Decision Making—Methods and Applications: A State-of-the-Art Survey* (Lecture notes in economics and mathematical systems). Heidelberg, Germany: Springer, 1981.
[56] A. S. Da Silva, Y. Mei, H. Ma, and M. Zhang, "Fragment-based genetic programming for fully automated multi-objective Web service composition," in *Proc. Genet. Evol. Comput. Conf.*, 2017, pp. 353–360.
[57] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artif. Intell.*, vol. 90, no. 1, pp. 281–300, 1997.

Chen Wang received the B.Eng. degree from Jiangsu University, Zhenjiang, China, in 2010, the M.B.A. degree from the National Institute of Development Administration, Bangkok, Thailand, in 2015, and the Ph.D. degree in engineering from the Victoria University of Wellington, Wellington, New Zealand, in 2020.

He is currently a Data Scientist with the National Institute of Water and Atmospheric Research, Wellington, and a Visiting Scholar with the Victoria University of Wellington. His research interests include evolutionary computation and machine learning for combinatorial optimization.

Hui Ma (Member, IEEE) received the Bachelor of Engineering degree from Tongji University, Shanghai, China, in 1989, the Bachelor of Information Sciences degree (with First Class Honours), the Master of Information Sciences (with First Class Honours), and the Ph.D. degree from Massey University, Palmerston North, New Zealand, in 2002, 2003, and 2008, respectively.

She is currently an Associate Professor of Software Engineering with the Victoria University of Wellington, Wellington, New Zealand. She has more than 100 publications, including leading journals and conferences in databases, service computing, cloud computing, evolutionary computation, and conceptual modeling. Her research interests include service composition, resource allocation in cloud, conceptual modeling, database systems, resource allocation in clouds, and evolutionary computation in combinatorial optimization.

Dr. Ma has served as a PC member for about 80 international conferences, including seven times as a PC Chair for conferences, such as ER, DEXA, and APCCM.

Gang Chen (Member, IEEE) received the B.Eng. degree from the Beijing Institute of Technology, Beijing, China, in 2000, and the Ph.D. degree from Nanyang Technological University (NTU), Singapore, in 2006.

From 2000 to 2001, he worked as a Software Engineer with Founder Electronics Private Ltd., Beijing. From 2005 to 2006, he worked as a Software Engineer with Crimsonlogic Private Ltd., Singapore. From 2006 to 2007, he worked as a Research Fellow with the Information Communication Institute of Singapore, School of Electrical and Electronic Engineering (EEE), NTU. From 2007 to 2010, he worked as a Teaching Fellow and a Visiting Assistant Professor with School of EEE, NTU. From 2010 to 2012, he worked as a Lecturer and Postgraduate Programme Leader with the Department of Computing, Unitec Institute of Technology, Auckland, New Zealand. Since 2012, he has been a Senior Lecturer with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand.

Sven Hartmann (Member, IEEE) received the Ph.D. and D.Sc. degrees from the University of Rostock, Rostock, Germany, in 1996 and 2001, respectively.

From 2002 to 2007, he worked first as an Associate Professor, then a Full Professor of Information Systems with Massey University, Palmerston North, New Zealand. Since 2008, he has been a Full Professor of Computer Science and the Chair of Databases and Information Systems with the Clausthal University of Technology, Clausthal-Zellerfeld, Germany, where he is also serving as an Academic Dean with the Faculty of Mathematics, Informatics and Mechanical Engineering. He has more than 150 publications. His research interests include database systems, big data management, conceptual modeling, and combinatorial optimization.

Prof. Hartmann served as a PC member for more than 80 conferences, including ten times as the PC Chair.