Using an Estimation of Distribution Algorithm to Achieve Multitasking Semantic Web Service Composition

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

***Abstract*—Web service composition composes existing Web services to accommodate users' requests for required functionalities with the best possible quality of services (QoS). Due to the computational complexity of this problem, evolutionary computation (EC) techniques have been employed to efficiently find composite services with near-optimal functional quality (i.e., quality of semantic matchmaking, QoSM for short) or nonfunctional quality (i.e., QoS) for each composition request individually. With a rapid increase in composition requests from a growing number of users, solving one composition request at a time can hardly meet the efficiency target anymore. Driven by the idea that the solutions obtained from solving one request can be highly useful for tackling other related requests, multitasking service composition approaches have been proposed to efficiently deal with multiple composition requests concurrently. However, existing attempts have not been effective in learning and sharing knowledge among solutions for multiple requests. In this article, we model the problem of collectively handling multiple service composition requests as a new multitasking service composition problem and propose a new permutation-based multifactorial evolutionary algorithm based on an estimation of distribution algorithm (EDA), named PMFEA-EDA, to effectively and efficiently solve this problem. In particular, we introduce a novel method for effective knowledge sharing across different service composition requests. For that, we develop a new sampling mechanism to increase the chance of identifying high-quality service compositions in both the single-tasking and multitasking contexts. Our experiment shows that our proposed approach, PMFEA-EDA, takes much less time than existing approaches that process each service request separately, and also outperforms them in terms of both QoSM and QoS.**

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

Manuscript received 28 August 2021; revised 29 December 2021 and 27 February 2022; accepted 18 April 2022. Date of publication 29 April 2022; date of current version 31 May 2023. 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 HPC and Data Science Department, National Institute of Water and Atmospheric Research, Wellington 6021, New Zealand (e-mail: chen.wang@niwa.co.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.2022.3170899.

Digital Object Identifier 10.1109/TEVC.2022.3170899

I. Introduction

SERVICE-ORIENTED computing employs the concept of Web services, i.e., self-describing Web-based applications that can be invoked over the Internet. Since a single Web service often fails to accommodate users' complex requirements, *Web service composition* [1] aims to loosely couple independent Web services in the form of service execution workflows, providing value-added functionalities to end-users. Web service composition is a promising research area and is highly desirable given the increasing number of services available in GIS [2], manufacturing [3], smartphone applications [4], [5], oil and gas industry [6], IoT applications [7], [8], logistics [9], and E-learning [10].

Since the service execution workflows are often unknown or not given in advance, many researchers have been interested in *fully automated service composition* that automatically constructs workflows with required functionalities while optimizing the overall quality of composite services. This overall quality usually refers to the functional quality (i.e., quality of semantic matchmaking, QoSM for short) or the nonfunctional quality (i.e., quality of service, QoS for short) of the composite services that stand for the service composition solutions [11]–[20].

The Web service composition problem has been proven to be *NP-hard* [21]. To tackle such a difficult problem, evolutionary computation (EC) techniques have been widely used to efficiently find near-optimal composition solutions in a cost-effective manner [12]–[20], [22]–[25]. These EC-based approaches are mainly designed to solve one service request at a time by improving users' quality preferences quantified in the form of either a single optimization objective [12], [14], [16]–[20], [25] or multiple objectives [13], [15], [22]–[24]. With the significant increase in the amount of service composition requests, a common disadvantage of these methods is that many service requests have to be dealt with repetitively and independently. In fact, similarities across these service requests that could be dealt with collectively have been consistently ignored by existing methods.

Many service requests have identical functional requirements on inputs and outputs but may vary due to different preferences on QoSM and/or QoS [26]. In a market-oriented environment, service composers often strategically group relevant service composition requests into several user segments (e.g., platinum, gold, silver, and bronze user segments), and

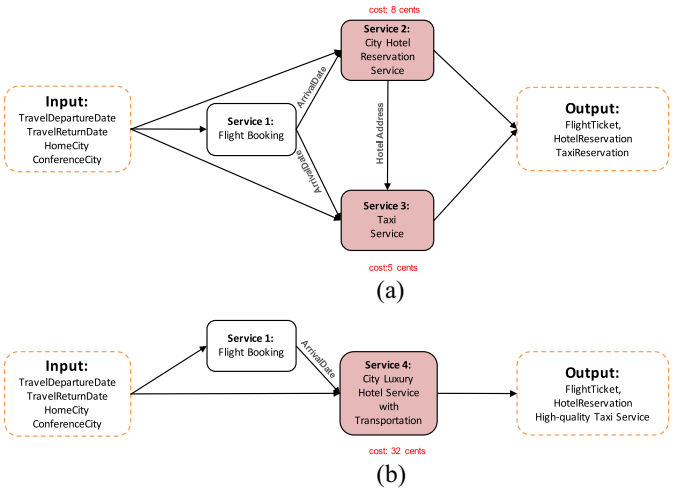

Fig. 1. Two composite booking services produced by TripPlanner. (a) Composite booking service A. (b) Composite booking service B.

each user segment presents distinguishable preferences over the service composition requests. Therefore, one composite service (i.e., a service composition solution) for a user segment can comfortably satisfy requirements from all users belonging to the same segment. In other words, any new service requests arising from the same segment will be immediately served by the same composite service designed *a priori* for that segment.

Herein, we use an example to demonstrate composite services for different user segments. TripPlanner is a service composition design system that produces composite booking services for many travel companies. See an example of two composite booking services utilized by TripPlanner in Fig. 1. Both composite services can be used to book airlines, hotels, and local transportation for travelers. Both are also composed by existing Web services from thousands of available Web services over the Internet. In Fig. 1, some services composed in composite booking service A (i.e., Service 2: City Hotel Reservation Service and Service 3: Taxi Service) are different from those composed in composite booking service B (i.e., Service 4: City Luxury Hotel Service with Transportation). In particular, Service 4 aggregates the functionalities of Service 2 and Service 3, providing high-quality hotel and taxi services. Apart from that, the cost of Service 4 (i.e., 32 cents) is much higher than that of Service 2 and Service 3 (i.e., $8 + 5 = 13$ cents). Apparently, these two composite booking services differ in QoSM and QoS. This is important to cater for different users with varied QoSM and QoS requirements. For example, large international travel companies (i.e., platinum segment users of TripPlaner) often care about their customers' needs more than small local travel companies (i.e., bronze segment users of TripPlaner), by providing high-quality services. These high-quality services contribute to a reliable and accurate user experience. In other words, composite services with high QoSM as employed by composite service B in Fig. 1 are preferably considered by the platinum segment users. In contrast, composite services with low QoSM with a tradeoff in cost, such as the composite booking service A, is preferred by bronze segment users of small local travel companies. From the perspective of service providers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segments.

The problem demonstrated above is clearly a multitasking problem. In line with this problem, we specifically consider multiple related service composition tasks, each of which corresponds to a separate user segment with different preferences (e.g., QoSM preference). A very recent work [26] proposed to handle such problems with multiple user segments collectively, where each segment captures the vital preference differences in terms of QoSM. They also adopted an emerging EC computing paradigm, namely, the multifactorial evolutionary algorithm (MFEA) [27]. Building on MFEA, a permutation-based MFEA (PMFEA) was developed to support intertask solution sharing via *assortative mating*. This is particularly achieved by using crossover and mutation operators. See Algorithm 3 in Appendix A for technical details.

However, empirical studies showed that the performance gain achievable by PMFEA is not prominent in comparison to single-tasking algorithms, indicating that assortative mating has limited effectiveness in promoting constructive intertask knowledge sharing. To tackle this limitation, we will propose a new technique to extract knowledge jointly from promising solutions to different tasks in the form of a series of related node histogram matrices (NHMs). The learned NHMs can be further utilized by the estimation of distribution algorithm (EDA) to search for promising regions of the solution space effectively. Note that existing EDA-based approaches for service composition have never been designed to extract and utilize knowledge from multiple tasks. In this article, we will propose the first PMFEA based on EDA (PMFEA-EDA) to simultaneously solve several fully automated service composition tasks for multiple user segments. PMFEA-EDA features the use of innovative intertask knowledge sharing techniques and solution sampling methods, all designed to improve the effectiveness and efficiency of the algorithm for multitasking service composition. The contributions of this article are as follows.

1) We propose a new algorithm (named PMFEA-EDA) to handle multiple service requests with respect to several predetermined user segments jointly. Our algorithm significantly outperforms existing methods by explicitly learning and sharing knowledge across solutions for different service requests (i.e., tasks). Particularly, PMFEA-EDA iteratively builds a set of single-tasking NHMs. Each NHM captures the knowledge of good solutions with respect to one task. Meanwhile, to facilitate knowledge sharing across different tasks, PMFEA-EDA also learns multitasking NHMs in association with every two tasks with similar preferences on QoSM (i.e., adjacent tasks). To the best of our knowledge, this is the first time that a combined set of single-tasking and multitasking NHMs is utilized for effectively handling the multitasking optimization problem.
2) We propose a new sampling mechanism over a combined set of single-tasking and multitasking NHMs to construct new composite services from these NHMs. This new sampling mechanism inspired by the principle of assortative mating in [27] is introduced in PMFEA-EDA to overcome the difficulty of using the node histogram-based sampling

(NHBSA) algorithm in a multitasking context. By using this mechanism, we can also effectively maintain high population diversity and prevent our method from premature convergence. To the best of our knowledge, this is the first time that sampling over a combined set of single-tasking and multitasking NHMs is used to balance between exploration and exploitation of the evolutionary search process in a multitasking context.

3) We evaluate the effectiveness and efficiency of our proposed approach by conducting experiments to compare it against state-of-the-art approaches, including a pure multitasking one, a pure single-tasking one, and a nonevolutionary one. We also analyze the effectiveness of knowledge sharing across adjacent tasks in terms of its impact on the aggregated quality of obtained solutions for the different tasks. This is achieved by experimentally comparing PMFEA-EDA without knowledge sharing (named PMFEA-EDA-WOT) with PMFEA-EDA.

The remainder of this article is organized as follows. Section II reviews some related work on semiautomated/fully automated Web service composition problems in single-tasking and multitasking contexts. Section III formulates service composition problems in single-tasking and multitasking contexts. Section IV presents an overview of the proposed PMFEA-EDA method and illustrates important components of this method. Section V demonstrates the effectiveness and efficiency of our PMFEA-EDA by comparing it against recent algorithms. Section VI discusses the main conclusions reached by our contribution and insights that guide the future work.

II. Related Work

Web service composition is performed using two strategies: 1) *semiautomated Web service composition* and 2) *fully automated Web service composition*. The first one assumes a predefined service composition workflow, which consists of abstract service slots that specify the required functionalities for atomic Web services, is known and selects an atomic service for each of the abstract service slots. The second one does not assume a workflow of abstract service slots is known and constructs a workflow simultaneously with atomic service selection. Apparently, compared to semiautomated Web service composition, fully automated Web service composition is more difficult, but it also opens new opportunities to improve QoS and QoSM without being restricted to a predefined workflow.

A. Literature on Single-Tasking Semiautomated Approaches

Non-EC service composition techniques try to identify optimal composite services by using some general optimization techniques, such as integer linear programming (ILP), dynamic programming, and reinforcement learning. However, due to the larger number of decision variables, non-EC service composition techniques, such as ILP, may lead to exponentially increased complexity and cost in computation [28]. Besides that, QoS of composite services in ILP-based approaches is calculated by summing up the individual QoS score of every component service. Such a QoS calculation is not always appropriate. Machine learning approaches have been proposed for service composition. Wang *et al.* [29], [30] and Liang *et al.* [31] developed service composition approaches based on deep reinforcement learning to adaptively construct composite services at the execution stage in response to QoS changes. However, these approaches only work for semiautomated service composition, where a composite service workflow must be provided in advance by users. As we have seen above, none of these existing works on automated Web service composition can solve multiple semantic service requests with different QoSM constraints. Therefore, effective and efficient approaches are needed to solve multitasking semantic service composition.

A variety of EC techniques has been demonstrated to be highly promising in solving single-tasking semiautomated Web service composition. This is because EC techniques are particularly useful in practice as they can efficiently find "good enough" (i.e., near-optimal) composite services. Based on the number of objectives to be optimized via these EC techniques, two subgroups of works are classified, i.e., single-tasking single-objective and single-tasking multiobjective EC-based semiautomated Web service composition. One subgroup aims to find composite services with an optimized unified score. For example, some works jointly optimize QoS and QoSM as an unified score [32]–[34]. The other subgroup aims to produce a set of tradeoff composite services with different objectives, e.g., time and cost [35].

The single objective and multiple objectives are optimized using various EC techniques, e.g., genetic algorithm (GA) [33], [35]–[37] and particle swarm optimization (PSO) [23], [38]. For example, Wada *et al.* [36] investigates a semiautomated approach with a vector-based representation in multiobjective GA. Two multiobjective GAs (called E^3-MOGA and X-E^3) are proposed in this work. Particularly, E^3-MOGA is designed to search for equally distributed Pareto-optimal solutions in the multiobjective space, while X-E^3 is designed to search for Pareto-optimal solutions that can reveal the maximum range of tradeoffs, covering extreme solutions in the search space.

B. Literature on Multitasking Semiautomated Approaches

A new EC computing paradigm, namely, MFEA [27], has been introduced recently. MFEA can solve multiple combinatorial optimization tasks concurrently and produce multiple solutions, with one for each task. MFEA searches a unified search space based on a unified random-key representation over multiple tasks and transfers implicit knowledge of promising solutions through the use of simple genetic operators across multiple tasks. The implicit knowledge transformation is achieved by performing crossover on two randomly selected parents solutions from two different tasks. This mechanism is called *assortative mating*. Apart from that, offspring is only evaluated on one task that is determined by its parents based on *vertical cultural transmission*. See Algorithm 3 in Appendix A and Algorithm 4 in Appendix B for technical details.

MFEA has shown its efficiency and effectiveness in several problem domains [11], [39]–[41]. To meet the efficiency and cost requirements, Bao *et al.* [11] reported the first attempt that employs MFEA to solve multiple service composition tasks together. Bao *et al.* [11] optimized QoS for two unrelated service requests simultaneously using MFEA,

achieving competitive results compared to single-objective EC techniques. However, this work cannot support fully automated service composition, where the service execution workflow is unknown or not given by the users. Furthermore, the number of tasks to be optimized concurrently is relatively small (i.e., two tasks). In this article, we will propose an MFEA (PMFEA) to solve more than two fully automated service composition tasks concurrently.

C. Literature on Single-Tasking Fully Automated Approaches

Graph search [42]–[47] is an alternative approach to fully automated service composition. Graph search works on searching composite services, which are constructed by subgraphs or paths from a service dependency graph. Constructing such a service dependency graph may suffer from the scalability issue when dealing with a large service repository with complex service dependencies. This issue can get even worse when QoS optimization is considered [48]. To consider multiple quality criteria in QoS, a recent work, named PathSearch [43], proposes an improved path-based search method over a graph [42]. Particularly, a node (i.e., an atomic service) associated with a higher rank is preferred in a path construction, and nodes are ranked based on the concept of dominance over multiple QoS quality criteria. In this article, we will compare PMFEA-EDA with the state-of-the-art graph search technique, i.e., PathSearch [43].

Evolutionary single-tasking fully automated service composition has been well studied in the majority of existing EC-based works. In particular, each service composition request is processed independently by using single-objective [12], [14], [16], [17], [19], [25] or multiobjective EC techniques [13], [15].

In the single-objective single-tasking setting, most of the existing service composition approaches used conventional EC techniques, which rely on the use of the implicit knowledge of promising solutions based on one or more variations of genetic operators on parent individuals. For example, tree-based composite solutions in [12], [14], [19], and [25] are produced using implicit knowledge defined by one or more variations of GP-based genetic operators on parent individuals. Apart from these conventional EC techniques, other approaches, such as [20], sample high-quality composite solutions using explicit knowledge that is learned by a distribution model, e.g., NHM. Their experiment demonstrates that learning an NHM of promising solutions does help to find near-optimal solutions. In the multiobjective single-tasking settings, there are very limited works, to the best of our knowledge, [13], [15], and [24] are the three recent attempts along this research direction. Very recently, an EDA-guided local search has been proposed that constructs distribution models from suitable Pareto front solutions and other good candidate solutions [24]. This approach can effectively and efficiently produce much better Pareto optimal solutions compared to other state-of-the-art methods [13], [15].

D. Literature on Multitasking Fully Automated Approaches

As discussed in Section II-B, Bao *et al.* [11] reported the first attempt to optimize QoS for two unrelated service requests simultaneously in semiautomated service composition. To overcome the limitations in [11], [26] proposed an MFEA (PMFEA) to solve more than two fully automated service composition tasks concurrently. Compared to single-tasking approaches, this method requires only a fraction of time. However, this work did not significantly outperform single-tasking approaches in finding high-quality solutions through the use of implicit learning. Motivated by the existing attempts to address multitasking service composition problems, with the aim to jointly find high-quality solutions for all tasks. In this article, we will propose a PMFEA-EDA to support explicit knowledge learning and explicit knowledge sharing across different tasks.

III. Preliminaries

A. Single-Tasking Semantic Web Service Composition

We review the formulation of the single-tasking semantic Web service composition problem. The following definitions are also given in [20].

A *semantic Web service* (*service*, for short) is considered 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 $QoS_S = \{t_S, ct_S, r_S, a_S\}$ 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* to describe the level of a match between outputs and inputs [50]. For concepts $\mathfrak{a}$ and $\mathfrak{b}$ in $\mathcal{O}$, the *matchmaking* returns *exact* if $\mathfrak{a}$ and $\mathfrak{b}$ are equivalent ($\mathfrak{a} \equiv \mathfrak{b}$), *plugin* if $\mathfrak{a}$ is a subconcept of $\mathfrak{b}$ ($\mathfrak{a} \sqsubseteq \mathfrak{b}$), *subsume* if $\mathfrak{a}$ is a superconcept of $\mathfrak{b}$ ($\mathfrak{a} \sqsupseteq \mathfrak{b}$), and *fail* if none of the previous matchmaking types is returned. In this article, we are only interested in *exact* and *plugin* matches for robust compositions. As argued in [34], *plugin* matches are less preferable than *exact* matches due to the overheads associated with data processing. For *plugin* matches, the semantic similarity of concepts is suggested to be considered when comparing different *plugin* matches.

A *robust causal link* [51] is a link between two matched services S and S', denoted as $S \rightarrow S'$, if an output $\mathfrak{a}$ ($\mathfrak{a} \in O_S$) of S serves as the input $\mathfrak{b}$ ($\mathfrak{b} \in O_{S'}$) of S' satisfying either $\mathfrak{a} \equiv \mathfrak{b}$ or $\mathfrak{a} \sqsubseteq \mathfrak{b}$. For concepts $\mathfrak{a}$ and $\mathfrak{b}$ in $\mathcal{O}$, the *semantic similarity* $\text{sim}(\mathfrak{a}, \mathfrak{b})$ is calculated based on the edge counting method in a taxonomy like WorldNet [52]. Advantages of this method are simple calculation and accurate measure [52]. Therefore, the *matchmaking type* and *semantic similarity* of a robust causal

TABLE I
QoS CALCULATION FOR A COMPOSITE SERVICE EXPRESSION C

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

link are defined as follows:

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

$$\text{sim}_{\text{link}}=\text{sim}(\mathfrak{a},\mathfrak{b})=\frac{2N_{\mathfrak{c}}}{N_{\mathfrak{a}}+N_{\mathfrak{b}}} \tag{2}$$

with a suitable parameter p, $0<p<1$, and with $N_{\mathfrak{a}}$, $N_{\mathfrak{b}}$, and $N_{\mathfrak{c}}$, which measure the distances from concept $\mathfrak{a}$, concept $\mathfrak{b}$, and the closest common ancestor $\mathfrak{c}$ of $\mathfrak{a}$ and $\mathfrak{b}$ to the top concept of the ontology $\mathcal{O}$, respectively. However, if more than one pair of matched output and input exist from service S to service S', type_e and sim_e will take on their average values.

The QoSM of a composite service is obtained by aggregating over all the robust causal links as follows:

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

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

Formal expressions as in [53] 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 sequential composition, parallel composition, choice, and iteration. 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 composite service, 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, sequence $\bullet$ and parallel $\parallel$, similar as most automated service composition works [14], [54]–[56] do, where composite services are represented as directed acyclic graphs (DAGs). The nodes of the DAG correspond to those services (also called *component services*) in service repository $\mathcal{SR}$ that are used in the composition. Let $\mathcal{G}=(V,E)$ be a DAG-based service composition solution from *Start* to *End*, where nodes correspond to the services and edges correspond to the matchmaking quality between the services. Often, $\mathcal{G}$ does not contain all services in $\mathcal{SR}$. The decoded DAG allows easy calculation of QoS in Table I and presents users with a complete workflow of service execution [20]. For example, the response time of a composite service is the time of the most time-consuming path in the DAG.

When multiple quality criteria are involved in decision making, the fitness of a solution is defined as a weighted sum of all individual criteria in (5), assuming the preference of each quality criterion based on its relative importance is provided by the user [57]

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

with $\sum_{k=1}^{6} w_k=1$ ($w_k\geq 0$). This objective function is defined as a *comprehensive quality model* for service composition. 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 a minimum value of 0 and a maximum value of 1. The minimum and maximum value of Q_3, Q_4, Q_5, and Q_6 are calculated across all the relevant services, which are discovered using a greedy search technique in [14] and [54]

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

The goal of single-tasking Web 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 solution for a given composition task T.

B. Multitasking Semantic Web Service Composition

In this article, we study the semantic Web service composition problem for multiple user segments with different QoSM preferences (henceforth, referred to as WSC-MQP). This problem is also defined in [26]. WSC-MQP is perceived as an evolutionary multitasking problem that aims to optimize K composition tasks concurrently with respect to K user segments.

Different from the composition task defined in the single-tasking context, a *composition task* in the multitasking context

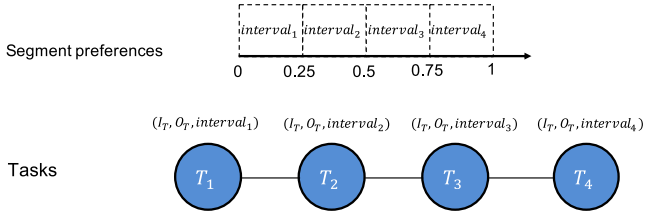

Fig. 2. Example of the neighborhood structure over four tasks.

is a tuple $T_j = (I_T, O_T, \text{interval}_j)$ where I_T is a set of task inputs, O_T is a set of task outputs, and interval_j is an interval based on QoSM for $j \in \{1, 2, \ldots, K\}$. The inputs in I_T and outputs in O_T are parameters that are semantically described by concepts in an ontology $\mathcal{O}$. The interval $\text{interval}_j = (\text{QoSM}_j^a, \text{QoSM}_j^b], j \in \{1, 2, \ldots, K\}$ and QoSM_j^a and QoSM_j^b are lower and upper bounds of QoSM for each user segment. Different user segments can be distinguished by their preferences on QoSM. The preferences of each user segment are defined as an interval, such as $QoSM \in (0.75, 0.1]$.

Wang *et al.* [26] introduced a neighborhood structure over T_j, where $j \in \{1, 2, \ldots, K\}$. This neighborhood structure is determined based on the tasks whose segment preferences on QoSM are adjacent to each other. For example, in Fig. 2, we consider $K = 4$ and let T_1, T_2, T_3, and T_4 be the four composition tasks corresponding to $\text{interval}_1 \in (0, 0.25]$, $\text{interval}_2 \in (0.25, 0.5]$, $\text{interval}_3 \in (0.5, 0.75]$, and $\text{interval}_4 \in (0.75, 1]$, respectively. Therefore, the adjacent tasks of T_2 are T_1 and T_3, whose segment preference on QoSM (i.e., interval_1 and interval_3) is adjacent to that of T_2 (i.e., interval_2).

The goal of multitasking semantic Web service composition is to find the K best solutions concurrently with one for each user segment.

C. Multifactorial Optimization

MFEA is a new evolutionary paradigm that considers K optimization tasks concurrently, where each task affects the evolution of a single population. In MFEA, a unified representation for the K tasks allows a unified search space made for all the K tasks. This unified representation of solutions can be decoded into solutions of the individual tasks. The following definitions are also given in [27] and capture the key attributes associated with each individual Π. For simplicity, we assume that all the tasks are maximization problems (see details in Section III-B).

Definition 1: The *factorial cost* f_j^Π of individual Π measures the fitness value with respect to the K tasks, where $j \in \{1, 2, \ldots, K\}$.

Definition 2: The *factorial rank* r_j^Π of individual Π on task T_j, where $j \in \{1, 2, \ldots, K\}$, is the position of Π in the population sorted in descending order according to their factorial cost with respect to task T_j.

Definition 3: The *scalar fitness* φ^Π of individual Π is calculated based on its best factorial rank over the K tasks, which is given by $\varphi^\Pi = 1/\min_{j \in \{1,2,\ldots,K\}} r_j^\Pi$.

Definition 4: The *skill factor* of individual Π denotes the most effective task of the K tasks, and is given by $\tau^\Pi = \text{argmin}_j\{r_j^\Pi\}$, where $j \in \{1, 2, \ldots, K\}$.

Based on the scalar fitness, evolved solutions in a population can be compared across the K tasks. In particular, an individual associated with a higher scalar fitness is considered to be better. Therefore, *multifactorial optimality* is defined as follows.

Definition 5: An individual $\Pi^\star$ associated with factorial cost $\{f_1^\star, f_2^\star, \ldots, f_K^\star\}$ is optimal iff $\exists j \in \{1, 2, \ldots, K\}$ such that $f_j^\star \geq f_j(\Pi)$, where Π denotes any feasible solution on task T_j.

IV. PMFEA-EDA Method

We first present an outline of PMFEA-EDA for WSC-MQP in Section IV-A. Subsequently, we will discuss the two main innovations of this method: 1) constructing and learning NHMs for effective exploration of the solution space over multiple tasks and 2) a new sampling mechanism to balance the tradeoff between exploration and exploitation in a multitasking context.

To learn a single-tasking NHM with respect to each task, we assign composite solutions to different solution pools based on their skill factors. Therefore, every solution pool stores promising solutions for one task. On the other hand, as shown in [26], solutions that are promising for one task can be used to evolve new solutions for its adjacent tasks (whose QoSM preferences are close). Due to this reason, we also prepare additional solution pools to store solutions that are promising for every two adjacent tasks. Every two adjacent tasks are identified as the most suitable tasks for knowledge sharing. Therefore, learning multitasking NHMs of these additional pools allows knowledge to be shared across adjacent tasks (see details in Section IV-C).

Moreover, we propose a sampling mechanism to balance exploration and exploitation. Particularly, a random sampling probability (rsp) is predefined to determine which NHM will be used to build new solutions. This mechanism is inspired by assortative mating in [27], where a random probability is defined for the occurrence of crossover on two parent solutions from the same skill factor or different skill factors.

The generation updates used in PMFEA-EDA are illustrated in Fig. 3. From the current population in Fig. 3, one sampled offspring population is created and further combined with the current population to produce the next population that only keeps the fittest solutions. Particularly, this sampled offspring population is formed from new solutions that are sampled from both single-tasking and multitasking NHMs. These NHMs are learned from multiple solution pools that consist of solutions assigned based on their skill factors.

A. Outline of PMFEA-EDA

The outline of PMFEA-EDA is shown in Algorithm 1. We first randomly initialize m permutation-based Π_k^g solutions, where $0 \leq k < m$ and $g = 0$. Each solution is represented as a random sequence of service indexes ranging from 0 to $|\mathcal{SR}| - 1$, and $\mathcal{SR}$ is a service repository containing registered Web services. For example, a permutation is represented as $\Pi = (\pi_1, \ldots, \pi_t, \ldots, \pi_n)$ such that $\pi_b \neq \pi_d$ for all $b \neq d$. Every permutation-based solution will be decoded into a DAG-based solution $\mathcal{G}_k^g$ for interpreting its service execution

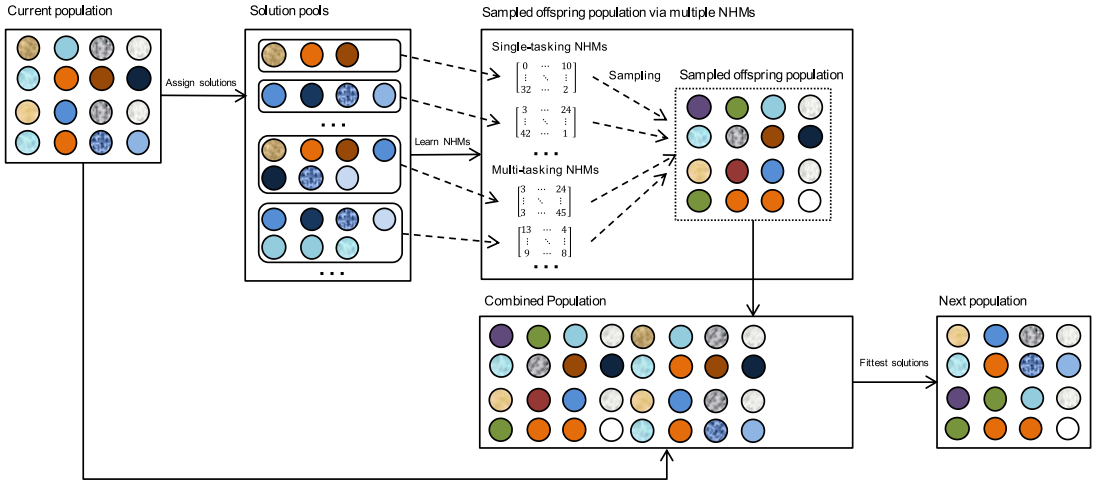

Fig. 3. Generation updates in PMFEA-EDA.

Algorithm 1: PMFEA-EDA for WSC-MQP

Input : T_j, K, and $g_{\max}$
Output: A set of composition solutions

1: Randomly initialize population $\mathcal{P}^g$ of m permutations Π_k^g as solutions (where $g = 0$ and $k = 1, \ldots, m$);
2: Decode each Π_k^g into DAG $\mathcal{G}_k^g$ using the decoding method;
3: Calculate $f_j^{\Pi_k^g}$, $r_j^{\Pi_k^g}$, $\varphi^{\Pi_k^g}$ and $\tau^{\Pi_k^g}$ of Π_k^g over T_j, where $j \in \{1, 2, \ldots, K\}$;
4: Encode each solution Π_k^g in $\mathcal{P}^g$ with another permutation Π'^g_k;
5: **while** $g < g_{\max}$ **do**
6: Generate offspring population $\mathcal{P}_a^{g+1}$ via multiple NHMs learning and sampling using ALGORITHM 2 ;
7: Decode solutions in $\mathcal{P}_a^{g+1}$ into DAG $\mathcal{G}_k^{g+1}$ using the decoding method;
8: Calculate $f^{\Pi_k^{g+1}}$ of solutions in $\mathcal{P}_a^{g+1}$ on the selected tasks related to the skill factors determined in its corresponding NHM;
9: Encode each solution Π_k^g in $\mathcal{P}^g$ with an another permutation Π'^g_k;
10: $\mathcal{P}^{g+1} = \mathcal{P}^g \cup \mathcal{P}_a^{g+1}$;
11: Update $r_j^{\Pi_k^{g+1}}$, $\varphi^{\Pi_k^{g+1}}$ and $\tau^{\Pi_k^{g+1}}$ of offspring in $\mathcal{P}^{g+1}$;
12: Keep top half the fittest individuals in $\mathcal{P}^{g+1}$ based on $\varphi^{\Pi_k^{g+1}}$;
13: Return the best $\Pi_j^\star$ over all the generations for T_j;

workflow using a decoding method proposed in [17]. Based on $\mathcal{G}_k^g$, we can easily determine $f_j^{\Pi_k^g}$, $r_j^{\Pi_k^g}$, $\varphi^{\Pi_k^g}$, and $\tau^{\Pi_k^g}$ of Π_k^g over task T_j, where $j \in \{1, 2, \ldots, K\}$. Afterward, we encode each solution Π_k^g in $\mathcal{P}^g$ into another permutation Π'^g_k based on its decoded DAG form $\mathcal{G}_k^g$ (see details in Section IV-B). This encoding step is essential and enables reliable and accurate learning of an NHM [20]. The iterative part of PMFEA-EDA comprises lines 6–12, which are repeated until a maximum generation $g_{\max}$ is reached. During each iteration, we generate an offspring population $\mathcal{P}_a^{g+1}$ via multiple NMHs using Algorithm 2 (see details in Section IV-C). Again, the same decoding and encoding techniques are employed to these solutions in $\mathcal{P}_a^{g+1}$. Afterward, we evaluate the fitness $f^{\Pi_k^{g+1}}$ of solutions in $\mathcal{P}_a^{g+1}$ on the task related to the imitated tasks skill factor, which is determined based on the principle of vertical culture transmission [27]. In particular, the skill factor of every produced solution is determined based on its corresponding NHM, where it is sampled from. We then produce the next population $\mathcal{P}^{g+1}$ by combining the current population $\mathcal{P}^g$ and the offspring population $\mathcal{P}_a^g$. Consequently, we update $r_j^{\Pi_k^{g+1}}$, $\varphi^{\Pi_k^{g+1}}$, and $\tau^{\Pi_k^{g+1}}$ of the combined population $\mathcal{P}^{g+1}$, and keep half of the population $\mathcal{P}^{g+1}$ based on $\varphi^{\Pi_k^{g+1}}$. When the maximum generation $g_{\max}$ is reached, the algorithm returns the best $\Pi_j^\star$ over all the generations for T_j.

B. Permutation-Based Representation

Permutations were utilized in the domain of fully automated service composition to indirectly represent a set of service composition solutions [16], [26]. Such a permutation, however, needs to be interpreted. For that, a forward graph building algorithm [17] is used to map a permutation to a DAG.

Since different permutations could be mapped to the same DAG, these permutations can lead to conflicts in learning the knowledge of service positions for one composition solution in NHM. As suggested in [20], we encode the permutation into a nearly unique and more reliable service permutation based on the decoded DAG, compared to its original permutation. Particularly, we produce this new permutation by combining two parts. One part comprises indexes of component services in the DAG, sorted in ascending order based on the longest distance from *Start* to every component service of the DAG while the second part comprises indexes of the remaining services in the permutation not utilized by the DAG (see details in [20]).

Example 1: Let us consider a composition task $T = (\{a, b\}, \{e, f\})$ and a service repository $\mathcal{SR}$ consisting of six atomic services: $S_0 = (\{e, f\}, \{g\}, QoS_{S_0})$, $S_1 = (\{b\}, \{c, d\},$

Algorithm 2: Multiple NHMs Learning and Sampling Over K Tasks

Input : $\mathcal{P}^g$
Output: $\mathcal{P}_a^{g+1}$

1: Initialize a set of empty $\mathcal{A}_q$ for each task and every two adjacent tasks;
2: Assign each solution $\Pi_k'^g$ in $\mathcal{P}^g$ to $\mathcal{A}_q$ based on its skill factor $\varphi^{\Pi_k'^g}$;
3: Learn $2K-1$ NHMs $\mathcal{NHM}_q^g$ from the $2K-1$ $\mathcal{A}_q$;
4: **while** $|\mathcal{P}^{g+1}| \leq m$ **do**
5: $rand \leftarrow Rand(0,1)$;
6: **if** $rand < rsp$ **then**
7: Select one NHM from multitasking NHMs randomly;
8: **else**
9: Select one NHM from single-tasking NHMs randomly;
10: Sample one solution Π_k^{g+1} from the selected NHM and put the solution into $\mathcal{P}^{g+1}$;
11: Π_k^{g+1} inherts the skill factor based on the selected NHM;
12: Return offspring population $\mathcal{P}_a^{g+1}$;

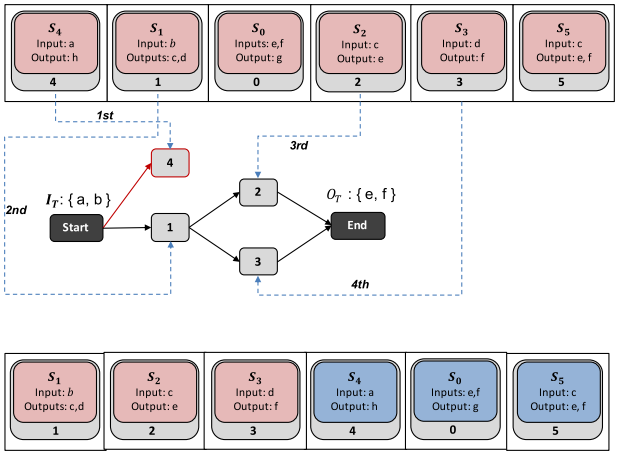

Fig. 4. Decoding a permutation into a DAG.

QoS_{S_1}), $S_2 = (\{c\}, \{e\}, QoS_{S_2})$, $S_3 = (\{d\}, \{f\}, QoS_{S_3})$, $S_4 = (\{a\}, \{h\}, QoS_{S_4})$, and $S_5 = (\{c\}, \{e,f\}, QoS_{S_5})$. The two special services Start $= (\emptyset, \{a,b\}, \emptyset)$ and $End = (\{e,f\}, \emptyset, \emptyset)$ are defined by a given composition task T. Fig. 4 illustrates an example of producing a DAG from decoding a given permutation [4, 1, 0, 2, 3, 5] and producing another permutation [1, 2, 3, 4, 0, 5].

In the example, we check the satisfaction on the inputs of services in the permutation from left to right. If any services can be immediately satisfied by the provided inputs of composition task I_T, we remove it from the permutation and add it to the DAG with a connection to *Start*. Afterward, we continue checking on services' inputs by using I_T and outputs of the services, and add satisfied services to the DAG. We continue this process until we can add *End* to the graph. In the last phase of the decoding process, some redundant services, such as 4, whose outputs contribute nothing to *End*, will be removed. In addition, this DAG is encoded as a new permutation [1, 2, 3, 4, 0, 5] consisting of two parts: one part [1, 2, 3] corresponds to a service discovered by the discussed sorted method on the DAG and another part [4, 0, 5] corresponds to the remaining atomic services in $\mathcal{SR}$, but not in the DAG. Furthermore, we also permit the encoding [1, 2, 3, 0, 4, 5], as no information can be extracted from the DAG to determine the order of 0, 4, and 5.

C. NHMs Learning and Sampling

Considering K composition tasks in PMFEA-EDA, we learn $2K-1$ NHMs based on promising solutions for sampling new candidate solutions. Every entry of NMHs roughly counts the number of times that a service index appears in the position of the permutation over all promising solutions in the pool. Among the NHMs, there are K single-tasking NHMs and $K-1$ multitasking NHMs. With respect to each NHM, a separate solution pool will be maintained by PMFEA-EDA to keep track of useful solutions for building the corresponding NHM. For example, considering the example of the four composition tasks discussed in Section III-B, i.e., T_1, T_2, T_3, and T_4, seven pools must be initialized for the four composition tasks and three adjacent task pairs (i.e., T_1 and T_2, T_2 and T_3, and T_3 and T_4).

Moreover, a parameter rsp is used to determine whether multitasking or single-tasking NHMs are selected for sampling. Particularly, a value of rsp close to 0 implies that single-tasking NHMs are more frequently used to build new solutions, while a value close to 1 implies that multitasking NHMs are used with high probability to build new solutions for two adjacent tasks.

The outline of multiple NHMs learning and sampling over K tasks is summarized in Algorithm 2. We first initialize a set of empty solution pools $\mathcal{A}_q$, where $1 \leq q \leq (2K-1)$. Afterward, we assign these encoded solutions to these pools based on the solutions' skill factors $\tau^{\Pi_k'^g}$. For example, if $\tau^{\Pi_k'^g} = 1$, this solution $\Pi_k'^g$ is assigned to two pools, one for task T_1, and the other is for both tasks T_1 and T_2. Afterward, we learn $2K-1$ NHMs from the $2K-1$ pools, respectively (see details in Section IV-D). The iteration part comprises lines 5–12. This iteration will not stop until m new solutions are constructed to form the offspring population $\mathcal{P}_a^{g+1}$. During the iteration, rsp is used to determine whether one NHM is randomly selected from the $2K-1$ single-tasking NHMs or multitasking NHMs. The selected NHM is used to build one solution. Hence, the skill factor of the newly created solution will also be determined by the associated tasks with the chosen NHM, inspired by the principle of vertical culture transmission [27]. After all iterations have been completed, Algorithm 2 returns the newly produced population $\mathcal{P}_a^{g+1}$ required in line 6 of Algorithm 1.

D. Application of Node Histogram-Based Sampling

We employ node histogram-based sampling [58] as a tool to create new permutations from the selected NHMs in step 7 or 9 in Algorithm 2. NHBSA can effectively sample new and

good candidate composite services from every node histogram matrix learned in each generation. This is because the learned node histogram can capture the explicit knowledge of a set of promising composite services in every generation with respect to each task and every adjacent task.

An NHM learned from solutions in each pool $\mathcal{A}_q$ at generation g, denoted by $\mathcal{NHM}_q^g$, is an $n \times n$-matrix with entries $e_{i,r}^g$ as follows:

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

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

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

Example 2: Let us consider a pool $\mathcal{A}_q$ at generation g. This pool is assigned with m permutations. For $m = 6$, an example of $\mathcal{A}_q^g$ may look as follows:

$$\mathcal{A}_q^g = \begin{bmatrix} \Pi_0^g \\ \Pi_1^g \\ \Pi_2^g \\ \Pi_3^g \\ \Pi_4^g \\ \Pi_5^g \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 0 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 & 5 \\ 4 & 3 & 0 & 1 & 2 & 5 \\ 4 & 3 & 0 & 1 & 2 & 5 \\ 2 & 1 & 3 & 0 & 4 & 5 \end{bmatrix}.$$

Consider $b_{\text{ratio}} = 0.2$, $m = 6$, and $n = 6$, then $\varepsilon = 0.24$. Thus, we can calculate $\mathcal{NHM}_q^g$ as follows:

$$\mathcal{NHM}_q^g = \begin{bmatrix} 2.24 & 1.24 & 1.24 & 0.24 & 2.24 & 0.24 \\ 0.24 & 3.24 & 1.24 & 2.24 & 0.24 & 0.24 \\ 2.24 & 0.24 & 2.24 & 2.24 & 0.24 & 0.24 \\ 2.24 & 2.24 & 0.24 & 2.24 & 0.24 & 0.24 \\ 0.24 & 0.24 & 2.24 & 0.24 & 4.24 & 6.24 \end{bmatrix}.$$

We use one entry $e_{0,0}^g = 2.24$ as an example to explain the meaning behind this value. The integer part 2 states that service S_0 appears twice in the first position over all the permutations in $\mathcal{A}_q^g$. The decimal part $0.24 = 6 * 0.2/(6-1)$ is the bias ε.

Once we have computed $\mathcal{NHM}_q^g$, we use NHBSA [58] to sample new candidate solutions Π_k^{g+1} for the population $\mathcal{P}_a^{g+1}$ (see Algorithm 5 in Appendix C for technical details). Afterward, the same decoding part discussed in Section IV-B will be employed on each newly sampled permutation to ensure its functional validity in its corresponding DAG form.

E. Fitness Evaluations for K Tasks

It is essential to include infeasible individuals (i.e., composite solutions that violate interval$_j$ of task T_j) into each population since infeasible composite solutions may help to find optimal solutions of other tasks. For example, we take an arbitrary example of a composite service whose QoSM equals 0.3. Based on the segment preferences in Fig. 2, this composite service is only feasible for just one task (i.e., T_2), since it complies with interval$_2$. However, this solution is infeasible for the other tasks (i.e., T_1, T_3, and T_4) as it violates interval$_1$, interval$_3$, and interval$_4$, respectively. We allow infeasible individuals in the population, but their fitness (i.e., factorial cost in a multitasking context) must be penalized for tasks T_1, T_3, and T_4 (see details in (9)). According to the fitness function in (9) with respect to T_j, we guarantee that f_j^Π of an infeasible individual falls below 0.5 while f_j^Π of a feasible individual stays above 0.5. Equation (11) quantifies the violation of interval$_j$ by measuring how far it is from QoSM(Π) in (10). In particular, an infeasible individual that violates interval$_j$ more should be penalized more

$$f_j^\Pi = \begin{cases} 0.5 + 0.5 * F(\Pi), & \text{if QoSM}(\Pi) \in \text{interval}_j \\ 0.5 * F(\Pi) - 0.5 * V_j(\Pi), & \text{otherwise.} \end{cases} \tag{9}$$

$$\text{QoSM}(\Pi) = w_7 \hat{\text{MT}} + w_8 \hat{\text{SIM}} \tag{10}$$

$$V_j(\Pi) = \begin{cases} \text{QoSM}_j^a - \text{QoSM}(\Pi), & \text{if QoSM}(\Pi) \le \text{QoSM}_j^a \\ \text{QoSM}(\Pi) - \text{QoSM}_j^b, & \text{otherwise} \end{cases} \tag{11}$$

with $\sum_{k=7}^{8} w_k = 1$. We can adjust the weights according to the preferences of user segments. $\hat{\text{MT}}$ and $\hat{\text{SIM}}$ are normalized values calculated within the range from 0 to 1 using (6).

To find the K best solutions with one for each task, the goal of multitasking semantic Web service composition is to maximize the objective function in (9) concerning the K tasks.

V. Experimental Evaluation

To demonstrate the effectiveness and efficiency of our PMFEA-EDA, we conduct experiments to compare it against four recent works: one evolutionary multitasking approach developed in [26]; two works [16], [20] that employed EC-based single-tasking techniques; one recent non-EC work [43] based on a graph traversal technique. Moreover, we also study the effectiveness of knowledge sharing across adjacent tasks in PMFEA-EDA to understand its impact on the quality of obtained solutions for all the tasks. This is achieved by experimentally comparing PMFEA-EDA without knowledge sharing (named PMFEA-EDA-WOT) with PMFEA-EDA.

We employ a quantitative evaluation approach for studying the effectiveness and efficiency of PMFEA-EDA[1] with augmented benchmark datasets[2] (i.e., WSC08-1 to WSC08-8 and WSC09-1 to WSC09-5 with increasing service repository $\mathcal{SR}$) used by the very recent studies [20], [26], [43], [59]. The benchmark datasets originally come from WSC 08 [60] and WSC09 [61] and were extended with real QoS attributes in QWS [62]. In WSC08 and WSC09, the semantics of service inputs and outputs are described by the OWL-S language. This language allows a high degree of automation in discovering, invoking, composing, and monitoring Web resources. Other Web service description languages, such as WSDL, RSDL, and OpenAI, can be transformed into OWL-S [63]–[65]. In other words, these different description languages can be supported by our algorithm technically.

[1]The code of PMFEA-EDA for automated Web service composition is available from https://github.com/chenwangnida/PMFEA-EDA-Code.

[2]The two augmented benchmarks for automated Web service composition are available from https://github.com/chenwangnida/Dataset.

TABLE II
MEAN FITNESS VALUES OF SOLUTIONS PER TASK FOR OUR APPROACHES IN COMPARISON TO PMFEA [26], EDA [20], FL [16], AND PATHSEARCH [43] FOR WSC08 (NOTE: THE HIGHER THE FITNESS THE BETTER)

Task T_1						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC08-1	0.164706 ± 0.002087	0.159196 ± 0.002385	0.163224 ± 0.001179	0.165277 ± 0.000297	0.163771 ± 0.00133	0.150044
WSC08-2	0.190545 ± 0	0.186381 ± 0.003009	0.183607 ± 0.005647	0.190513 ± 0.000119	0.186696 ± 0.004012	0.148601
WSC08-3	0.135325 ± 0.000218	0.132964 ± 0.000276	0.133842 ± 0.000569	0.134644 ± 0.00019	0.13494 ± 0.000261	0.130825
WSC08-4	0.181683 ± 0	0.175812 ± 0.001998	0.180604 ± 0.001476	0.181683 ± 0	0.180149 ± 0.001553	0.168962
WSC08-5	0.158257 ± 0.000409	0.140024 ± 0.002234	0.150427 ± 0.005761	0.154543 ± 0.001248	0.148781 ± 0.004893	0.146512
WSC08-6	0.13877 ± 0.000784	0.136855 ± 0.000457	0.137388 ± 0.000878	0.137572 ± 0.000251	0.138903 ± 0.000851	0.162561
WSC08-7	0.155868 ± 0.001971	0.145899 ± 0.00104	0.147377 ± 0.002597	0.151623 ± 0.001254	0.150155 ± 0.002666	0.140096
WSC08-8	0.140659 ± 0.00028	0.138724 ± 0.000381	0.139543 ± 0.000493	0.140014 ± 0.000162	0.140249 ± 0.000375	0.140389
Task T_2						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC08-1	0.783246 ± 0.004123	0.77537 ± 0.005286	0.78094 ± 0.003956	0.780501 ± 0.005908	0.779276 ± 0.003184	0.275044
WSC08-2	0.810462 ± 0.003182	0.792539 ± 0.008051	0.796973 ± 0.011255	0.804669 ± 0.005148	0.798272 ± 0.007808	0.745933
WSC08-3	0.73858 ± 0.000144	0.736444 ± 0.000254	0.737742 ± 0.000522	0.737772 ± 0.000163	0.737822 ± 0.000394	0.736360
WSC08-4	0.778908 ± 0	0.775306 ± 0.002071	0.77849 ± 0.001029	0.778908 ± 0	0.777783 ± 0.001245	0.774642
WSC08-5	0.761759 ± 0.00042	0.74463 ± 0.003025	0.754992 ± 0.006117	0.757012 ± 0.001323	0.752598 ± 0.004912	0.727467
WSC08-6	0.74079 ± 0.000159	0.738981 ± 0.000254	0.740372 ± 0.000701	0.739996 ± 0.000153	0.740168 ± 0.000354	0.707353
WSC08-7	0.761402 ± 0.000417	0.748287 ± 0.001869	0.754869 ± 0.004231	0.757697 ± 0.00085	0.754984 ± 0.003319	0.735627
WSC08-8	0.748496 ± 0.00029	0.738195 ± 0.001043	0.743635 ± 0.002428	0.74168 ± 0.000886	0.742857 ± 0.002489	0.722568
Task T_3						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC08-1	0.786634 ± 0.103967	0.783322 ± 0.005268	0.798704 ± 0.008522	0.803764 ± 0.008859	0.798732 ± 0.005459	0.803575
WSC08-2	0.878406 ± 0	0.876893 ± 0.002883	0.876137 ± 0.005002	0.87791 ± 0.001854	0.876441 ± 0.002861	0.796941
WSC08-3	0.22369 ± 0.000218	0.219439 ± 0.000521	0.222456 ± 0.001023	0.221868 ± 0.000288	0.222154 ± 0.000612	0.217205
WSC08-4	0.253553 ± 0	0.248326 ± 0.002299	0.253038 ± 0.001563	$0.253549 \pm 2e-05$	0.252369 ± 0.001297	0.263426
WSC08-5	0.24297 ± 0.000782	0.222143 ± 0.001811	0.236261 ± 0.006324	0.234794 ± 0.002012	0.230628 ± 0.00512	0.187677
WSC08-6	0.226387 ± 0.000262	0.222613 ± 0.000784	0.2259 ± 0.001693	0.225165 ± 0.000223	0.225222 ± 0.000924	0.127144
WSC08-7	0.249391 ± 0.00029	0.232944 ± 0.001898	0.241935 ± 0.005391	0.243889 ± 0.001904	0.240343 ± 0.003232	0.211617
WSC08-8	0.231538 ± 0.000315	0.21998 ± 0.000939	0.22664 ± 0.002304	0.223648 ± 0.000975	0.225382 ± 0.002224	0.180967
Task T_4						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC08-1	0.216365 ± 0.018324	0.175843 ± 0.012607	0.204841 ± 0.019299	0.21709 ± 0.015455	0.204375 ± 0.013322	0.223000
WSC08-2	0.362814 ± 0	0.360104 ± 0.004077	0.357579 ± 0.01212	0.362814 ± 0	0.357846 ± 0.010539	0.211233
WSC08-3	0.098541 ± 0.000257	0.094654 ± 0.000391	0.097468 ± 0.000966	0.096868 ± 0.000288	0.097154 ± 0.000612	0.092205
WSC08-4	0.128553 ± 0	0.123964 ± 0.002133	0.128012 ± 0.001311	$0.128549 \pm 2e-05$	0.127289 ± 0.001313	0.138426
WSC08-5	0.117822 ± 0.000908	0.097303 ± 0.002245	0.111215 ± 0.006349	0.109794 ± 0.002012	0.105628 ± 0.00512	0.062677
WSC08-6	0.101324 ± 0.000315	0.097761 ± 0.000633	0.10097 ± 0.001524	0.100165 ± 0.000223	0.100187 ± 0.000957	0.002144
WSC08-7	0.1244 ± 0.000302	0.107739 ± 0.001348	0.117175 ± 0.005221	0.118889 ± 0.001904	0.115343 ± 0.003232	0.086617
WSC08-8	0.106516 ± 0.000354	0.09512 ± 0.000775	0.101716 ± 0.002423	0.098648 ± 0.000975	0.100382 ± 0.002224	0.055967

Wang *et al.* [26] defined four composition tasks for each dataset, which has identical I_T and O_T but four different QoSM preferences introduced at the beginning of Section III-B. We evaluate three multitasking methods: PMFEA-EDA, PMFEA-WTO, and PMFEA [26], and three single-tasking methods: EDA [20], FL [16], and PathSearch [43] (see the comparison results in Sections V-A and V-B). In particular, the three multitasking methods are utilized to optimize the four composition tasks concurrently, while the three single-tasking methods are utilized to optimize each task one by one, and the execution time is the aggregation of time spent on all tasks. We run 30 times of each EC-based method independently for all the datasets while we run 1 time of the deterministic non-EC method, i.e., PathSearch [43].

To make fair comparisons over all the methods, we use the same number of evaluations in PMFEA-EDA, PMFEA-WTO, PMFEA [26], EDA [20], and FL [16] for each run, i.e., the population size is 30 with 200 generations. We define rsp as 0.2 so that every single-tasking NHM and every multitasking NHM are expected to create 6 and 2 solutions, respectively, for the population size of 30. Therefore, each task has roughly the same number of solutions from the sampling. b_{ratio} is 0.0002 according to EDA [20]. Other parameters of PMFEA [26], EDA [20], FL [16], and PathSearch [43] follow the common settings reported in the literature. For PathSearch [43], the parameter k (i.e., the number of services considered in the path construction at each step) associated with this algorithm is set to 7, which reported the highest quality in their paper. All the weights in (5) and (10) follow PMFEA [26]: w_1 and w_2 are set equally to 0.25, and w_3, w_4, w_5, and w_6 are all set to 0.125, these weights are set to properly balance QoSM and QoS; w_7 and w_8 are set to 0.5, these weights are set to balance all quality criteria in QoSM. In general, weight settings are decided to reflect user segments' preferences. We have conducted tests with other weights and observed similar results to those reported below.

All the methods are run on a grid engine system (i.e., N1 Grid Engine 6.1 software) that performs tasks via a collection of computing resources, i.e., Linux PCs and each PC with an Intel Core i7-4770 CPU (3.4 GHz) and 8-GB RAM. This hardware configuration is used for all the methods compared in this article.

A. Comparison of the Fitness

Wilcoxon rank-sum test is employed at a significance level of 5% to verify the observed differences in fitness values. Particularly, pairwise comparisons of all the competing methods are carried out to count the number of times they are found

TABLE III
MEAN FITNESS VALUES OF SOLUTIONS PER TASK FOR OUR APPROACHES IN COMPARISON TO PMFEA [26], EDA [20], FL [16], AND PATHSEARCH [43] FOR WSC09 (NOTE: THE HIGHER THE FITNESS THE BETTER)

Task T_1						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC09-1	0.196195 ± 0.000547	0.193173 ± 0.002266	0.192864 ± 0.003543	0.196316 ± 0.000314	0.193947 ± 0.003276	0.136890
WSC09-2	0.15621 ± 0.000806	0.141811 ± 0.000646	0.148709 ± 0.00488	0.145844 ± 0.001347	0.146254 ± 0.002946	0.137212
WSC09-3	0.158664 ± 0.001038	0.147505 ± 0.001923	0.150053 ± 0.003885	0.156733 ± 0.001425	0.15355 ± 0.002688	0.140160
WSC09-4	0.142097 ± 0.000467	0.139684 ± 0.000359	0.140255 ± 0.00073	0.140256 ± 0.000673	0.141451 ± 0.000515	0.135814
WSC09-5	0.145391 ± 0.000337	0.143159 ± 0.000389	0.143447 ± 0.000946	0.145045 ± 0.000278	0.144942 ± 0.000705	0.137544
Task T_2						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC09-1	0.815627 ± 0.002673	0.808235 ± 0.003689	0.809369 ± 0.007696	0.816144 ± 0	0.807483 ± 0.005398	0.261890
WSC09-2	0.761226 ± 0.00064	0.74019 ± 0.001354	0.752848 ± 0.006956	0.74704 ± 0.005348	0.74895 ± 0.006425	0.732288
WSC09-3	0.77392 ± 0.003505	0.755821 ± 0.003864	0.760746 ± 0.006192	0.772646 ± 0.001529	0.761924 ± 0.006194	0.727531
WSC09-4	0.741242 ± 0.000261	0.738214 ± 0.000567	0.739866 ± 0.000853	0.739946 ± 0.000233	0.739826 ± 0.000692	0.733738
WSC09-5	0.74173 ± 0.000756	0.738334 ± 0.000293	0.73936 ± 0.001217	0.739431 ± 0.000255	0.739467 ± 0.000735	0.734080
Task T_3						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC09-1	0.806034 ± 0.097563	0.820217 ± 0.003495	0.820107 ± 0.007241	0.823483 ± 0.000978	0.819418 ± 0.003768	0.788172
WSC09-2	0.242136 ± 0.000241	0.222519 ± 0.001753	0.234557 ± 0.00651	0.232177 ± 0.00283	0.22968 ± 0.004616	0.205492
WSC09-3	0.791989 ± 0	0.787464 ± 0.002324	0.789012 ± 0.002968	0.791938 ± 0.000146	0.788726 ± 0.002576	0.190768
WSC09-4	0.227768 ± 0.000446	0.221226 ± 0.000789	0.224278 ± 0.001957	0.224629 ± 0.00063	0.224127 ± 0.001467	0.206797
WSC09-5	0.224546 ± 0.000719	0.219729 ± 0.000897	0.221169 ± 0.002244	0.2218 ± 0.000243	0.221102 ± 0.001248	0.206344
Task T_4						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC09-1	0.226227 ± 0.011127	0.22145 ± 0.009191	0.219863 ± 0.013342	0.22731 ± 0.003251	0.221582 ± 0.00946	0.206402
WSC09-2	0.117137 ± 0.000241	0.097486 ± 0.001454	0.109708 ± 0.00659	0.107177 ± 0.00283	0.10468 ± 0.004616	0.080492
WSC09-3	0.222379 ± 0	0.215091 ± 0.005245	0.217783 ± 0.005575	0.222212 ± 0.00034	0.216698 ± 0.00533	0.065768
WSC09-4	0.102637 ± 0.000634	0.096245 ± 0.001065	0.099276 ± 0.001935	0.099674 ± 0.000596	0.099127 ± 0.001467	0.081797
WSC09-5	0.099487 ± 0.000716	0.094344 ± 0.000876	0.096085 ± 0.002181	0.096785 ± 0.000271	0.096102 ± 0.001248	0.081344

TABLE IV
MEAN EXECUTION TIME (IN SECONDS) OVER ALL THE TASKS FOR OUR APPROACHES IN COMPARISON TO PMFEA [26], EDA [20], FL [16], AND PATHSEARCH [43] FOR WSC08 (NOTE: THE SHORTER THE TIME THE BETTER)

Tasks T_1, T_2, T_3 and T_4						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC08-1	66 ± 15	151 ± 14	79 ± 23	310 ± 103	228 ± 230	8
WSC08-2	31 ± 4	62 ± 8	35 ± 20	131 ± 67	64 ± 56	15
WSC08-3	901 ± 90	1483 ± 123	1956 ± 531	3682 ± 338	8084 ± 3657	43
WSC08-4	39 ± 5	85 ± 9	84 ± 22	132 ± 63	351 ± 265	18
WSC08-5	763 ± 100	1516 ± 184	1548 ± 596	3516 ± 351	7128 ± 3632	48
WSC08-6	11356 ± 1040	15714 ± 1305	16486 ± 3464	36824 ± 2664	65212 ± 30075	320
WSC08-7	1140 ± 172	2463 ± 210	2972 ± 1637	5536 ± 444	10862 ± 8071	306
WSC08-8	1856 ± 144	3183 ± 364	2998 ± 800	7842 ± 652	12424 ± 5387	908

to be better, similar, or worse than the others. Consequently, we can rank all the competing methods and highlight the top performance in green color.

Tables II and III show the mean value of the solution fitness and the standard deviation over 30 repetitions for each task solved by PMFEA-EDA, PMFEA-EDA-WOT, PMFEA, EDA, and FL, and deterministic fitness value over one run for each task solved by PathSearch. We observe that the quality (i.e., QoSM and QoS) of solutions produced by using our PMFEA-EDA and EDA [20] is generally higher than those obtained by PMFEA and FL [16]. This corresponds well with our expectation that learning the knowledge of promising solutions explicitly can effectively improve the quality of composite services.

Furthermore, PMFEA-EDA performs better than single-tasking EDA [20]. This observation indicates that addressing multiple tasks collectively is often more effective than addressing each task individually, through the use of NHM. Particularly, compared to single-tasking EDA, multitasking methods are more likely to evolve a well-diversified population of solutions. Consequently, we can easily prevent the evolutionary process from converging prematurely.

In addition, PMFEA-EDA also outperforms PMFEA-EDA-WTO significantly and is labeled as top performance. This corresponds well with our expectation that explicit knowledge sharing through multitasking NHMs can significantly improve its ability in finding high-quality solutions.

Finally, PathSearch [43] achieves the worst performance in finding high-quality solutions, despite 5 out of 52 composition tasks are marked in green. It is due to that PathSearch [43] was designed to make the locally best choice over the k services at each step and gradually build a path-based composite solution.

B. Comparison of the Execution Time

Wilcoxon rank-sum test at a significance level of 5% is also employed to verify the observed differences in values of execution time (in seconds). Tables IV and V show the mean value of the execution time and the standard deviation over 30 repetitions for all tasks solved by PMFEA-EDA, PMFEA-EDA-WOT, PMFEA, EDA, and FL, and the value of execution time over 1 run for all tasks solved by PathSearch.

TABLE V
MEAN EXECUTION TIME (IN SECONDS) OVER ALL THE TASKS FOR OUR APPROACHES IN COMPARISON TO PMFEA [26], EDA [20], FL [16], AND PATHSEARCH [43] FOR WSC09 (NOTE: THE SHORTER THE TIME THE BETTER)

Tasks T_1, T_2, T_3 and T_4						
Method	PMFEA-EDA	PMFEA-EDA-WTO	PMFEA [26]	EDA [20]	FL [16]	PathSearch [43]
WSC09-1	54 ± 8	52 ± 11	79 ± 87	184 ± 12	150 ± 151	20
WSC09-2	1571 ± 181	1533 ± 218	2371 ± 804	7058 ± 369	8479 ± 3002	463
WSC09-3	1085 ± 186	975 ± 122	1821 ± 740	5057 ± 885	5926 ± 3199	728
WSC09-4	57788 ± 6902	50310 ± 7535	71903 ± 19042	202464 ± 9366	250146 ± 55355	3894
WSC09-5	9671 ± 1092	8834 ± 819	13689 ± 6723	39257 ± 1885	47879 ± 16126	3138

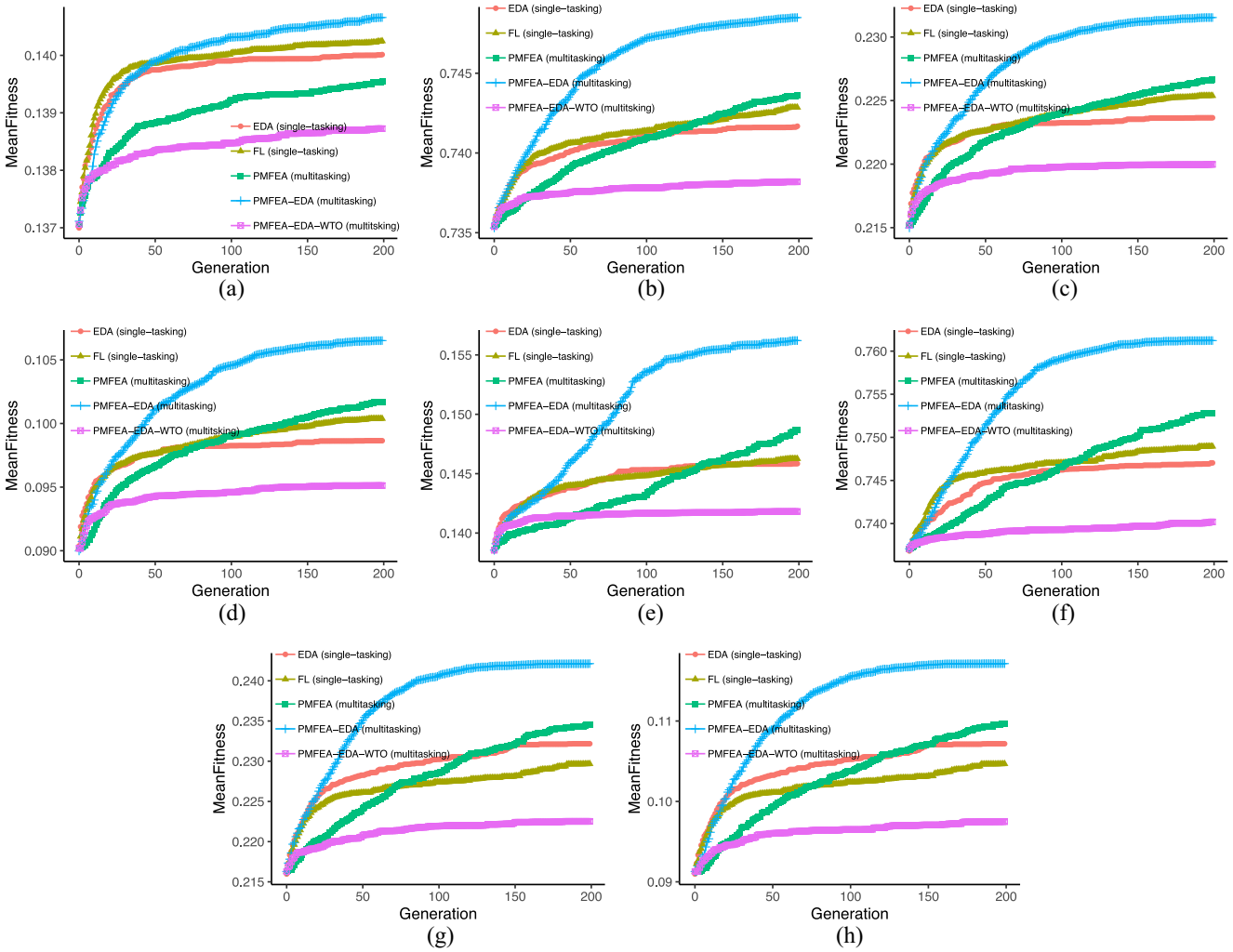

Fig. 5. Mean fitness over generations for tasks 1–4, for WSC08-8 and WSC09-2 (Note: the larger the fitness the better). (a) WSC08-8 Task 1. (b) WSC08-8 Task 2. (c) WSC08-8 Task 3. (d) WSC08-8 Task 4. (e) WSC09-2 Task 1. (f) WSC09-2 Task 2. (g) WSC09-2 Task 3. (h) WSC09-2 Task 4.

First, PathSearch [43] requires the least execution time. This is because PathSearch [43] only searches the constructed path based on the k best services from a prestored service dependency graph. However, efficiency is not the focus of this article because finding high-quality composite services at the design stage is our focus.

Apart from PathSearch [43], PMFEA-EDA, PMFEA-EDA-WTO, and PMFEA appear to be more efficient than EDA [20] and FL [16]. Although the same number of evaluations is assigned for each run of every method, EDA [20] and FL [16] are single-tasking methods that have to solve each composition task one by one.

Finally, PMFEA-EDA-WTO requires slightly less execution time for all the tasks since PMFEA-EDA demands more time for learning NHMs when service repository $\mathcal{SR}$ becomes larger and larger. However, the extra time incurred in PMFEA-EDA is not substantial compared to other multitasking methods.

C. Comparison of the Convergence Rate

We also study the convergence rate of PMFEA-EDA, PMFEA-EDA-WTO, PMFEA, EDA [20], and FL [16]. Using WSC08 and WSC09-2 as two examples, we show the behaviors of the effectiveness of all the methods in Fig. 5.

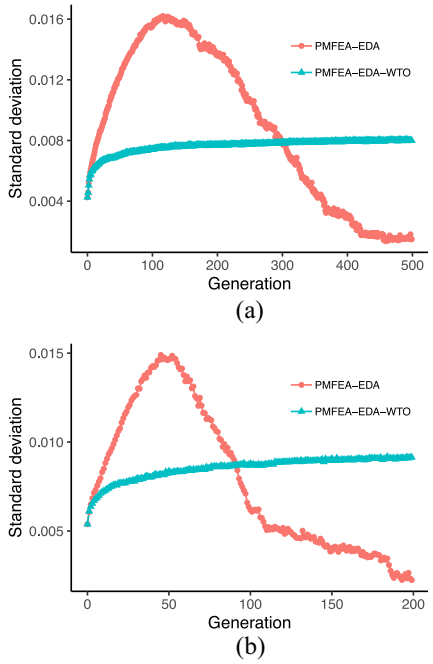

Fig. 6. Population diversity measured by standard deviation over generations. (a) WSC08-8. (b) WSC09-2.

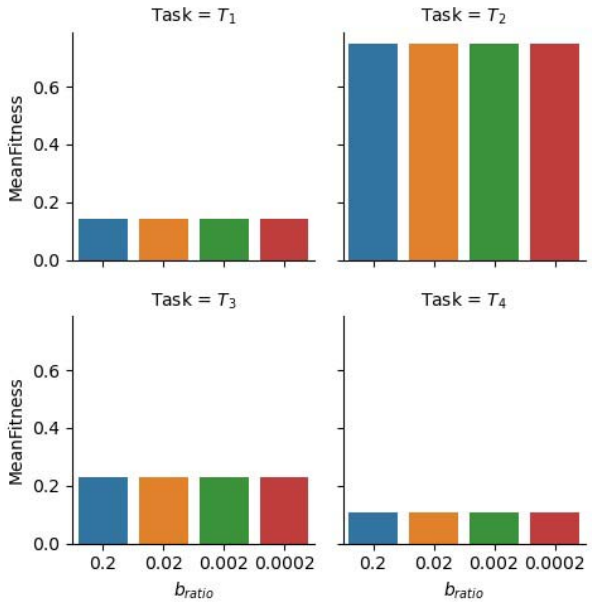

Fig. 7. Mean fitness values of PMFEA-EDA with different b_{ratio} over four tasks in WSC08-8.

Fig. 5 shows the evolution of the mean fitness value of the best solutions found so far along 200 generations for all the approaches. We can see that PMFEA-EDA converges much faster than all the other methods in all the tasks (except task 1 on WSC 08-08). Besides that, PMFEA-EDA converges faster than PMFEA-EDA-WTO, and eventually reaches the highest plateau. This observation matches well with our expectation that knowledge sharing across tasks is very effective.

D. Comparison of the Population Diversity

To explore the effectiveness of our proposed sampling strategy from multitasking NHMs with knowledge sharing, we investigated the diversity of the sampled population using 30 independent runs. We have used WSC08-8 and WSC09-2 as examples to illustrate the population diversity of the two methods, i.e., PMFEA-EDA and PMFEA-EDA-WTO. To examine the population diversity of these two methods over WSC08-8 and WSC09-2, we run 500 generations, instead of 200 generations, for WSC08-8 because the size of the service repository in WSC08-8 (i.e., 16238) is much bigger (with larger searching space) than that of WSC09-2 (i.e., 8258). Fig. 6 shows the population diversity, measured by the standard deviation of fitness values in (5) across 500 and 200 generations for WSC08-8 and WSC09-2, respectively.

In Fig. 6(a) and (b), PMFEA-EDA focuses more on exploration than PMFEA-EDA-WOT at the beginning of the evolutionary process, with the standard deviation of fitness values reaching its peak at generation 120 and 50 for WSC08-8 and WSC09-2, respectively. Starting from generation 350 and 100 for WSC08-8 and WSC09-2, respectively, PMFEA-EDA focuses comparatively more on exploitation than PMFEA-EDA-WOT, and the corresponding fitness standard deviation continues to decrease to a low level. This observation matches well with our expectation that more exploration is performed in the beginning, and more exploitation happens in later phases of the evolution. On the other hand, PMFEA-EDA-WOT performs exploitation all the time as the standard deviation of fitness values stays at roughly the same levels.

E. Sensitivity Analysis of the Model Parameter

In the literature, b_{ratio} was set to 0.0002 in the single-tasking context [20], [58]. To study its sensitivity in a multitasking context, we study the performance of PMFEA-EDA with varying values of b_{ratio}. Particularly, we use WSC08-8 as an example to test the sensitivity of b_{ratio} under a wide range of settings, i.e., 0.2, 0.02, 0.002, and 0.0002.

Fig. 7 shows the mean fitness values of the best solutions found after 200 generations for tasks T_1, T_2, T_3, and T_4 with respect to four different b_{ratio} values over 30 runs. As shown in Fig. 7, we observed no significant differences in the mean fitness values for different values of b_{ratio} in each task. This finding indicates that the performance of PMFEA-EDA is not sensitive to the settings of b_{ratio}.

VI. Conclusion

In this article, we introduced a new PMFEA-EDA to solve service composition tasks from multiple user segments with different QoSM preferences in the context of fully automated Web service composition. In particular, single-tasking and multitasking NHMs are constructed to learn explicit knowledge of promising solutions for each task and every two adjacent tasks, respectively. This explicit learning mechanism is expected to perform knowledge learning and sharing better with an aim to find high-quality composite services for multiple tasks simultaneously. In addition, we also allow explicit knowledge to be effectively shared across every two adjacent tasks through the use of multitasking NHMs.

Furthermore, a sampling mechanism is proposed to balance the exploration and exploitation of the evolutionary search process for multiple tasks. Our experimental evaluations show that our proposed method outperforms two state-of-the-art single-tasking and one recent multitasking EC-based approaches for finding high-quality solutions. Besides that, the execution time of our approach is comparable to the recent multitasking approach and outperforms two state-of-the-art single-tasking EC approaches by saving a large fraction of time. Future work can investigate the adaptions of NMHs for handling a dynamically updated service repository in an online fashion and study our EDA-based multitasking techniques to handle the dynamic and multitasking semantic Web service composition problem.

REFERENCES

[1] F. Curbera, W. Nagy, and S. Weerawarana, "Web services: Why and how," in *Proc. Workshop Object-Orient. Web Serv. OOPSLA*, 2001, pp. 1–7.

[2] "Geographic Information Systems (GIS): MIT Geodata Repository." [Online]. Available: https://libguides.mit.edu/gis/Geodata

[3] Y. Wu, G. Peng, H. Wang, and H. Zhang, "A heuristic algorithm for optimal service composition in complex manufacturing networks," *Complexity*, vol. 2019, Apr. 2019, Art. no. 7819523.

[4] G. Mesfin, G. Ghinea, T.-M. Grønli, and S. Alouneh, "REST4Mobile: A framework for enhanced usability of REST services on smartphones," *Concurrency Comput. Pract. Exp.*, vol. 32, no. 1, p. e4174, 2020.

[5] G. Mesfin, G. Ghinea, T.-M. Grønli, and M. Younas, "Web service composition on smartphones: The challenges and a survey of solutions," in *Proc. Int. Conf. Mobile Web Intell. Inf. Syst.*, 2018, pp. 126–141.

[6] A. M. Saettler, K. R. Llanes, P. Ivson, D. L. Nascimento, E. T. Corseuil, and G. M. da Silva, "An ontology-driven framework for data integration and dynamic service composition: Case study in the oil & gas industry," in *Proc. IADIS Int. Conf. WWW/Internet*, 2017, pp. 79–86.

[7] M. Hamzei and N. J. Navimipour, "Toward efficient service composition techniques in the Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3774–3787, Oct. 2018.

[8] A. Krishna, M. Le Pallec, R. Mateescu, L. Noirie, and G. Salaün, "IoT composer: Composition and deployment of IoT applications," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. Companion (ICSE-Companion)*, 2019, pp. 19–22.

[9] S. Chitra and A. Bhuvaneswari, "Application of perfect domination in logistics services using Web service composition," *J. Comput. Math. Sci.*, vol. 10, no. 1, pp. 207–214, 2019.

[10] P. K. Keserwani, S. G. Samaddar, and P. Kumar, "e-Learning Web services and their composition strategy in SOA," in *Smart Computing Paradigms: New Progresses and Challenges*. Singapore: Springer, 2020, pp. 291–302.

[11] L. Bao *et al.*, "An evolutionary multitasking algorithm for cloud computing service composition," in *Proc. World Congr. Serv.*, 2018, pp. 130–144.

[12] 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.

[13] A. S. da Silva, H. Ma, Y. Mei, and M. Zhang, "A hybrid memetic approach for fully automated multi-objective Web service composition," in *Proc. IEEE ICWS*, 2018, pp. 26–33.

[14] A. S. da Silva, H. Ma, and M. Zhang, "Genetic programming for QoS-aware Web service composition and selection," *Soft Comput.*, vol. 20, pp. 3851–3867, Oct. 2016.

[15] 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. GECCO*, 2017, pp. 353–360.

[16] A. S. da Silva, Y. Mei, H. Ma, and M. Zhang, "Evolutionary computation for automatic Web service composition: An indirect representation approach," *J. Heuristics*, vol. 24, pp. 425–456, Jun. 2018.

[17] C. Wang, H. Ma, A. Chen, and S. Hartmann, "Comprehensive quality-aware automated semantic Web service composition," in *Proc. Aust. Joint Conf. Artif. Intell.*, 2017, pp. 195–207.

[18] C. Wang, H. Ma, G. Chen, and S. Hartmann, "Towards fully automated semantic Web service composition based on estimation of distribution algorithm," in *Proc. Aust. Joint Conf. Artif. Intell.*, 2018, pp. 458–471.

[19] C. Wang, H. Ma, G. Chen, and S. Hartmann, "GP-based approach to comprehensive quality-aware automated semantic Web service composition," in *Proc. Asia–Pacific Conf. Simul. Evol. Learn.*, 2017, pp. 170–183.

[20] C. Wang, H. Ma, A. Chen, and S. Hartmann, "Knowledge-driven automated Web service composition—An EDA-based approach," in *Proc. Int. Conf. Web Inf. Syst. Eng.*, 2018, pp. 135–150.

[21] J. Rao and X. Su, "A survey of automated Web service composition methods," in *Proc. Semantic Web Serv. Web Process Composition*, 2005, pp. 43–54.

[22] Y. Chen, J. Huang, and C. Lin, "Partial selection: An efficient approach for QoS-aware Web service composition," in *Proc. IEEE Int. Conf. Web Serv.*, 2014, pp. 1–8.

[23] H. Yin, C. Zhang, B. Zhang, Y. Guo, and T. Liu, "A hybrid multiobjective discrete particle swarm optimization algorithm for a SLA-aware service composition problem," *Math. Problems Eng.*, vol. 2014, Jan. 2014, Art. no. 252934.

[24] C. Wang, H. Ma, and G. Chen, "Using EDA-based local search to improve the performance of NSGA-II for multiobjective semantic Web service composition," in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2019, pp. 434–451.

[25] 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.

[26] C. Wang, H. Ma, G. Chen, and S. Hartmann, "Evolutionary multitasking for semantic Web service composition," in *Proc. IEEE Congr. Evol. Comput.*, 2019, pp. 2490–2497.

[27] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: Toward evolutionary multitasking," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 343–357, Jun. 2016.

[28] J. Li, Y. Yan, and D. Lemire, "Full solution indexing for top-K Web service composition," *IEEE Trans. Services Comput.*, vol. 11, no. 3, pp. 521–533, May/Jun. 2018.

[29] H. Wang *et al.*, "Integrating reinforcement learning and skyline computing for adaptive service composition," *Inf. Sci.*, vol. 519, pp. 141–160, May 2020.

[30] H. Wang *et al.*, "Adaptive and large-scale service composition based on deep reinforcement learning," *Knowl. Based Syst.*, vol. 180, pp. 75–90, Sep. 2019.

[31] H. Liang, X. Wen, Y. Liu, H. Zhang, L. Zhang, and L. Wang, "Logistics-involved QoS-aware service composition in cloud manufacturing with deep reinforcement learning," *Robot. Comput.-Integr. Manuf.*, vol. 67, 2021, Art. no. 101991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584520302027

[32] V. R. Chifu, C. B. Pop, I. Salomie, D. S. Suia, and A. N. Niculici, "Optimizing the semantic Web service composition process using cuckoo search," in *Intelligent Distributed Computing V*. Berlin, Germany: Springer, 2011, pp. 93–102.

[33] Y.-Y. FanJiang and Y. Syu, "Semantic-based automatic service composition with functional and non-functional requirements in design time: A genetic algorithm approach," *Inf. Softw. Technol.*, vol. 56, no. 3, pp. 352–373, 2014.

[34] F. Lécué, "Optimizing QoS-aware semantic Web service composition," in *Proc. Int. Semantic Web Conf.*, 2009, pp. 375–391.

[35] S. Liu, Y. Liu, N. Jing, G. Tang, and Y. Tang, "A dynamic Web service selection strategy with QoS global optimization based on multi-objective genetic algorithm," in *Proc. Int. Conf. Grid Cooperative Comput.*, 2005, pp. 84–89.

[36] H. Wada, J. Suzuki, Y. Yamano, and K. Oba, "E^3: A multiobjective optimization framework for SLA-aware service composition," *IEEE Trans. Services Comput.*, vol. 5, no. 3, pp. 358–372, 3rd Quart., 2011.

[37] H. Wu, S. Deng, W. Li, M. Fu, J. Yin, and A. Y. Zomaya, "Service selection for composition in mobile edge computing systems," in *Proc. IEEE Int. Conf. Web Serv. (ICWS)*, 2018, pp. 355–358.

[38] J. Liao, Y. Liu, J. Wang, J. Wang, and Q. Qi, "Lightweight approach for multi-objective Web service composition," *IET Softw.*, vol. 10, no. 4, pp. 116–124, 2016.

[39] L. Feng, Y.-S. Ong, A.-H. Tan, and I. W. Tsang, "Memes as building blocks: A case study on evolutionary optimization + transfer learning for routing problems," *Memetic Comput.*, vol. 7, no. 3, pp. 159–180, 2015.

[40] Y. Yuan, Y.-S. Ong, A. Gupta, P. S. Tan, and H. Xu, "Evolutionary multitasking in permutation-based combinatorial optimization problems: Realization with TSP, QAP, LOP, and JSP," in *Proc. TENCON*, 2016, pp. 3157–3164.

[41] L. Zhou, L. Feng, J. Zhong, Y.-S. Ong, Z. Zhu, and E. Sha, "Evolutionary multitasking in combinatorial search spaces: A case study in capacitated vehicle routing problem," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, 2016, pp. 1–8.
[42] S. Chattopadhyay, A. Banerjee, and N. Banerjee, "A scalable and approximate mechanism for Web service composition," in *Proc. IEEE Int. Conf. Web Serv.*, 2015, pp. 9–16.
[43] S. Chattopadhyay, A. Banerjee, and N. Banerjee, "A fast and scalable mechanism for Web service composition," *ACM Trans. Web*, vol. 11, no. 4, pp. 1–36, 2017.
[44] M. Chen and Y. Yan, "QoS-aware service composition over graphplan through graph reachability," in *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*, 2014, pp. 544–551.
[45] P. Hennig and W.-T. Balke, "Highly scalable Web service composition using binary tree-based parallelization," in *Proc. IEEE Int. Conf. Web Serv.*, 2010, pp. 123–130.
[46] 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 Serv.*, 2010, pp. 42–49.
[47] Y. Yan and M. Chen, "Anytime QoS-aware service composition over the graphplan," *Serv. Orient. Comput. Appl.*, vol. 9, no. 1, pp. 1–19, 2015.
[48] A. Klein, F. Ishikawa, and S. Honiden, "Efficient heuristic approach with improved time complexity for QoS-aware service composition," in *Proc. IEEE Int. Conf. Web Serv.*, 2011, pp. 436–443.
[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, A. Delteil, and A. Léger, "Optimizing causal link based Web service composition," in *Proc. ECAI*, 2008, pp. 45–49.
[52] K. Shet and U. D. Acharya, "A new similarity measure for taxonomy based on edge counting," 2012, *arXiv:1211.4709*.
[53] H. Ma, K.-D. Schewe, B. Thalheim, and Q. Wang, "A formal model for the interoperability of service clouds," *Serv. Orient. Comput. Appl.*, vol. 6, no. 3, pp. 189–205, 2012.
[54] H. Ma, A. Wang, and M. Zhang, "A hybrid approach using genetic programming and greedy search for QoS-aware Web service composition," *Transactions on Large-Scale Data- and Knowledge-Centered Systems XVIII*, vol. 18. Berlin, Germany: Springer, 2015, pp. 180–205.
[55] A. S. da Silva, H. Ma, and M. Zhang, "GraphEvol: A graph evolution technique for Web service composition," in *Proc. DEXA*, 2015, pp. 134–142.
[56] 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. Comb. Optim.*, 2016, pp. 202–218.
[57] 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), vol. 164. Berlin, Germany: Springer, 1981.
[58] 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.
[59] S. Sadeghiram, H. Ma, and G. Chen, "Composing distributed data-intensive Web services using distance-guided memetic algorithm," in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2019, pp. 411–422.
[60] 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 E-Commerce Technol. 5th IEEE Conf. Enterprise Comput. E-Commerce E-Serv.*, s2008, pp. 351–354.
[61] 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. Commerce Enterprise Comput.*, 2009, pp. 487–490.
[62] 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.
[63] J. Ling and L. Jiang, "Semantic description of IoT services: A method of mapping WSDL to OWL-S," *Comput. Sci*, vol. 46, no. 4, pp. 89–94, 2019.
[64] S. Schwichtenberg, C. Gerth, and G. Engels, "From open API to semantic specifications and code adapters," in *Proc. IEEE Int. Conf. Web Serv. (ICWS)*, 2017, pp. 484–491.
[65] W. Zhang, L. Jiang, and H. Cai, "An ontology-based resource-oriented information supported framework towards restful service generation and invocation," in *Proc. 5th IEEE Int. Symp. Serv. Orient. Syst. Eng.*, 2010, pp. 107–112.

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

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

Hui Ma (Member, IEEE) received the B.E. degree from Tongji University, Shanghai, China, in 1989, and the Ph.D. degree from Massey University, Palmerston North, New Zealand, in 2008.

She is currently an Associate Professor of Software Engineering with Victoria University of Wellington, Wellington, New Zealand. She has more than 120 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 90 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 Beijing Institute of Technology, Beijing, China, in 2000, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2006.

He is currently a Senior Lecturer with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He has more than 120 publications, including leading journals and conferences in machine learning, evolutionary computation, and distributed computing areas, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, *Journal of Autonomous Agents and Multi-Agent Systems*, *ACM Transactions on Autonomous and Adaptive Systems*, IEEE ICWS, and IEEE SCC. His research interests include evolutionary computation, reinforcement learning, multiagent systems, and cloud and service computing.

Dr. Chen is serving as a PC member of many prestigious conferences, including ICLR, ICML, NeurIPS, IJCAI, and AAAI, and the Co-Chair for Australian AI 2018 and CEC 2019.

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

From 2002 to 2007, he worked first as an Associate Professor and 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 Clausthal University of Technology, Clausthal-Zellerfeld, Germany, where he is also serving as the Academic Dean with the Faculty of Mathematics, Informatics and Mechanical Engineering. He has more than 150 publications.

Prof. Hartmann served as a PC member for more than 80 conferences, including ten times as a PC chair. His research interests include database systems, big data management, conceptual modeling, and combinatorial optimization.