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 non-functional 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 paper, we model the problem of collectively handling multiple service requests as a new multitasking service composition problem and propose a new Permutation-based Multi-factorial 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—Web service composition, QoS optimization, Combinatorial optimization, Evolutionary Multitasking, Estimation of Distribution Algorithm

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 non-functional quality (i.e., quality of service, QoS for short) of the composite services that stand for the service composition solutions [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21].

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], [13], [14], [15], [16], [17], [18], [19], [20], [22], [23], [24], [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], [17], [18], [19], [20], [25] or multiple objectives [13], [15], [22], [23], [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 QoS 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 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 ser-
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 QoS and QoSM. This is important to cater for different users with varied QoS and QoSM requirements. For example, large international travel companies (i.e., platinum segment users of TripPlanner) often care about their customers’ needs more than small local travel companies (i.e., bronze segment users of TripPlanner), by providing high-quality services. These high-quality services contribute to a reliable and accurate user experience. In other words, composite services with high QoS as employed by composite service B in Fig. 1 are preferably considered by the platinum segment users. In contrast, composite services with low QoS with a trade-off 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., QoS and QoSM). 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 QoS and QoSM. They also adopted an emerging EC computing paradigm, namely, the multi-factorial evolutionary algorithm (MFEA) [27]. Building on MFEA, a permutation-based multi-factorial evolutionary algorithm (PMFEA) was developed to support inter-task 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 inter-task 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 paper, we will propose the first Permutation-based Multi-Factorial Evolutionary Algorithm 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 inter-task 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 paper are as follows:

1) We propose a new algorithm (named PMFEA-EDA) to handle multiple service requests with respect to several pre-determined 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 QoS (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 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 non-evolutionary 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) and knowledge sharing (named PMFEA-EDA-WOT) with PMFEA-EDA.

The remainder of the paper is organized as follows. Sect. II reviews some related work on semi-automated/fully automated web service composition problems in single-tasking and multitasking contexts. Sect. III formulates service composition problems in single-tasking and multitasking contexts. Sect. IV presents an overview of the proposed PMFEA-EDA method and illustrates important components of this method. Sect. V demonstrates the effectiveness and efficiency of our PMFEA-EDA by comparing it against recent algorithms. Sect. 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: semi-automated web service composition and fully-automated web service composition. The first one assumes a pre-defined 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 semi-automated 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 pre-defined workflow.

A. Literature on single-tasking semi-automated 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. [29], [30], [31] develop 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 semi-automated 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 have been demonstrated to be highly promising in solving single-tasking semi-automated 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 multi-objective EC-based semi-automated 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], [33], [34]. The other subgroup aims to produce a set of trade-off 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], [36], [37] and Particle Swarm Optimization (PSO) [23], [38]. For example, [36] investigates a semi-automated approach with a vector-based representation in multi-objective GA. Two multi-objective GAs (called $E^3$-MOGA and $X^E$) are proposed in this work. Particularly, $E^3$-MOGA is designed to search for equally distributed Pareto-optimal solutions in the multi-objective space, while $X^E$ is designed to search for Pareto-optimal solutions that can reveal the maximum range of trade-offs, covering extreme solutions in the search space.

B. Literature on multitasking semi-automated approaches

A new EC computing paradigm, namely multi-factorial evolutionary algorithm (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], [40], [41]. To meet the efficiency and cost requirements, [11] reports the first attempt that employs MFEA to solve multiple service composition tasks together. [11] optimizes 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 paper, we will propose a multi-factorial evolutionary
algorithm (PMFEA) to solve more than two fully automated
service composition tasks concurrently.

C. Literature on single-tasking fully automated approaches
Graph search [42], [43], [44], [45], [46], [47] is an alterna-
tive approach to fully automated service composition. Graph
search works on searching composite services, which are
constructed by subgraphs or paths from a service dependency
table. 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 consid-
ered [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 paper, we will compare PMFEA-EDA with the state-
of-the-art graph search technique, i.e., PathSearch [43].

Evolutionary single-tasking fully automated service composi-
tion 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], [17],
[14], [16], [19], [25] or multi-objective 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 ge-
netic operators on parent individuals. For example, tree-based
composite solutions in [12], [14], [19], [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., Node
Histogram Matrix (NHM). Their experiment demonstrates that
learning an NHM of promising solutions does help to find
near-optimal solutions. In the multi-objective single-tasking
settings, there are very limited works, to the best of our
knowledge, [13], [15], [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-art methods [15], [13].

D. Literature on multitasking fully automated approaches
As discussed in Sect. II-B, [11] reported the first attempt
to optimize QoS for two unrelated service requests simulta-
neously in semi-automated service composition. To overcome
the limitations in [11], [26] proposed a multi-factorial evolu-
tionary algorithm (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 composi-
tion problems, with the aim to jointly find high-quality solutions
for all tasks. In this paper, we will propose a PMFEA-EDA
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 \( QoS_S = \{t_S, cts_S, r_S, a_S\} \) is a
set of non-functional attributes of \( S \). The inputs in \( I_S \) and
outputs in \( O_S \) are parameters modeled through concepts in
a domain-specific ontology \( O \). The attributes \( t_S, cts, r_S, a_S \)
refer to the response time, cost, reliability, and availability of
service \( S \), respectively, which are four commonly used QoS
attributes [49].

A service repository \( SR \) is a finite collection of services
supported by a common ontology \( O \).

A composition task (also called service request) over a
given \( SR \) 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
defined by concepts in the ontology \( O \). Two special atomic
services \( Start = (\emptyset, I_T, \emptyset) \) and \( End = (O_T, \emptyset, \emptyset) \) are always
included in \( SR \) 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 \( a, b \) in \( O \) the
matchmaking returns \( \text{exact} \) if \( a \) and \( b \) are equivalent (\( a \equiv b \)),
\( \text{plugin} \) if \( a \) is a sub-concept of \( b \) (\( a \sqsubseteq b \)), \( \text{subsume} \) if \( a \) is a
super-concept of \( b \) (\( a \sqsupset b \)), and \( \text{fail} \) if none of the previous
matchmaking types is returned. In this paper, we are only in-
terested in \( \text{exact} \) and \( \text{plugin} \) matches for robust compositions. As
argued in [34], \( \text{plugin} \) matches are less preferable than \( \text{exact} \)
matches due to the overheads associated with data processing.
For \( \text{plugin} \) matches, the semantic similarity of concepts is
suggested to be considered when comparing different \( \text{plugin} \)
matches.

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

\[
type_{\text{link}} = \begin{cases} 
1 & \text{if } a \equiv b \text{ (exact match)} \\
p & \text{if } a \not\equiv b \text{ (plugin match)} 
\end{cases} 
\] (1)

\[
sim_{\text{link}} = \frac{2N_a}{N_a + N_b} 
\] (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. 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 \( \text{QoSM} \) of a composite service is obtained by aggregating over all the robust causal links as follows:

\[
MT = \prod_{j=1}^{m} type_{\text{link},j} 
\] (3)

\[
SIM = \frac{1}{m} \sum_{j=1}^{m} sim_{\text{link},j} 
\] (4)

Formal expressions as in [53] are used to represent service compositions. The constructors \( \bullet \), \( \| \), \( + \) and \( * \) are used to denote sequential composition, parallel composition, choice, and iteration, respectively. The set of \( \text{composite service expressions} \) 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 \( \bullet (C_1, \ldots, C_d), \| (C_1, \ldots, C_d), \) \( + (C_1, \ldots, C_d) \), and \( * C_0 \) are in \( \mathcal{SC} \), too. Let \( C \) be a composite service expression. If \( C \) denotes an atomic service \( S \) then its \( \text{QoS} \) is given by \( \text{QoSM}_S \). Otherwise the \( \text{QoSM} \) 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 \( \ell \) denotes the average number of iterations. Therefore, \( \text{QoSM} \) of a composite service, i.e., availability (\( A \)), reliability (\( R \)), execution time (\( T \)), and cost (\( C \)) can be obtained by aggregating \( a_C, r_C, t_C \) and \( c_C \) as in Table I.

In the presentation of this paper, we mainly focus on two constructors, sequence \( \bullet \) and parallel \( \| \), similar as most automated service composition works [54], [14], [55], [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 \( \text{Start} \) to \( \text{End} \), 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 \( \text{QoS} \) 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 Eq. (5), assuming the preference of each quality criterion based on its relative importance is provided by the user [57]:

\[
F(C) = w_1 MT + w_2 SIM + w_3 \hat{A} + w_4 \hat{R} + w_5 (1 - \hat{T}) + w_6 (1 - \hat{C})T
\] (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. \( MT, SIM, \hat{A}, \hat{R}, \hat{T}, \) and \( \hat{C} \) are normalized values calculated within the range from 0 to 1 using Eq. (6). To simplify the presentation, we also use the notation \( (Q_1, Q_2, Q_3, Q_4, Q_5, Q_6) = (MT, SIM, A, R, T, CT) \). \( Q_1 \) 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 [54], [14].

The goal of single-tasking web semantic service composition is to find a composite service expression \( C^* \) that maximizes the objective function in Eq. (5). \( C^* \) is hence considered as the best solution for a given composition task \( T \).

### B. Multi-tasking Semantic Web Service Composition

In this paper, we study the semantic Web Service Composition problem for Multiple user segments with different \( \text{QoS} \) preferences (henceforth referred to as \( \text{WSC-MQP} \)). 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 is a tuple \( T_j = (IT_j, OT_j, interval_j) \) where \( IT_j \) is a set of task inputs, \( OT_j \) is a set of task outputs, \( interval_j \) is an interval based on \( \text{QoS} \) for \( j \in \{1, 2, \ldots, K\} \). The inputs in \( IT_j \) and outputs in \( OT_j \) are parameters that are semantically described by concepts in an ontology \( \mathcal{O} \). The interval \( interval_j = (QoSM_{j}^{min}, QoSM_{j}^{max}), j \in \{1, 2, \ldots, K\} \) and \( QoSM_{j}^{min}, QoSM_{j}^{max} \) are lower and upper bounds of \( \text{QoS} \) for each user segment. Different user segments can be distinguished by their preferences on \( \text{QoS} \). The preferences of each user segment is defined as an interval, such as \( \text{QoS} \in (0.75, 0.1] \).

[26] introduces 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 \( \text{QoS} \) 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 \( interval_1 \in (0, 0.25], interval_2 \in (0.25, 0.5], interval_3 \in (0.5, 0.75] \) and \( interval_4 \in (0.75, 1] \), respectively. Therefore, the adjacent tasks of \( T_2 \) are \( T_1 \) and
TABLE I: QoS calculation for a composite service expression C.

<table>
<thead>
<tr>
<th>C = (C_1, \ldots, C_d)</th>
<th>r_C = \prod_{k=1}^{d} r_{C_k}</th>
<th>a_C = \prod_{k=1}^{d} a_{C_k}</th>
<th>c_{t_C} = \sum_{k=1}^{d} c_{t_{C_k}}</th>
<th>c_C = \sum_{k=1}^{d} c_{C_k}</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C_1, \ldots, C_d)</td>
<td>(C_1, \ldots, C_d)</td>
<td>(C_1, \ldots, C_d)</td>
<td>(C_1, \ldots, C_d)</td>
<td>(C_1, \ldots, C_d)</td>
</tr>
</tbody>
</table>

IV. PMFEA-EDA Method

We first present an outline of PMFEA-EDA for WSC-MQP in Sect. IV-A. Subsequently, we will discuss the two main innovations of this method: constructing and learning NHMs for effective exploration of the solution space over multiple tasks; and a new sampling mechanism to balance the trade-off 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 QoS 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 Sect. 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 $\Pi_0^k$ 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
Current population

Solution pools

Sampled offspring population via multiple NHMs

Single-tasking NHMs

Multi-tasking NHMs

Combined Population

Next population

Fig. 3: Generation updates in PMFEA-EDA

**Algorithm 1. PMFEA-EDA for WSC-MQP**

**Input**: $T_j, K$, and $g_{\text{max}}$

**Output**: A set of composition solutions

1. Randomly initialize population $P^g$ of $m$ permutations $\Pi_k^g$ as solutions (where $g = 0$ and $k = 1, \ldots, m$);
2. Decode each $\Pi_k^g$ into DAG $G_k^g$ using the decoding method;
3. Calculate $f_{i,j}^{\Pi_k^g}$, $r_{i,j}^{\Pi_k^g}$, $\varphi_{i,j}^{\Pi_k^g}$ and $\tau_{i,j}^{\Pi_k^g}$ of $\Pi_k^g$ over $T_j$, where $j \in \{1, 2, \ldots, K\}$;
4. Encode each solution $\Pi_k^g$ in $P^g$ with another permutation $\Pi_k^{g+1}$;
5. while $g < g_{\text{max}}$ do
6. Generate offspring population $P^{g+1}$ via multiple NHMs learning and sampling using Algorithm 2;
7. Decode solutions in $P^{g+1}$ into DAG $G_k^{g+1}$ using the decoding method;
8. Calculate $f_{i,j}^{G_k^{g+1}}$, $r_{i,j}^{G_k^{g+1}}$, $\varphi_{i,j}^{G_k^{g+1}}$ and $\tau_{i,j}^{G_k^{g+1}}$ of solutions in $P^{g+1}$ on the selected tasks related to the skill factors determined in its corresponding NHM;
9. Encode each solution $\Pi_k^{g+1}$ in $P^{g+1}$ with another permutation $\Pi_k^{g+1}$, $P^{g+1} = P^g \cup P^{g+1}$;
10. Update $r_{i,j}^{\Pi_k^{g+1}}$, $\varphi_{i,j}^{\Pi_k^{g+1}}$ and $\tau_{i,j}^{\Pi_k^{g+1}}$ of offspring in $P^{g+1}$;
11. Keep half the fittest individuals in $P^{g+1}$ based on $\varphi_{i,j}^{\Pi_k^{g+1}}$;
12. Return the best $\Pi_j^g$ over all the generations for $T_j$;

registered web services. For example, a permutation is represented as $\Pi = (\pi_1, \ldots, \pi_d)$ such that $\pi_0 \neq \pi_d$ for all $b \neq d$. Every permutation-based solution will be decoded into a DAG-based solution $G_k^g$ for interpreting its service execution workflow using a decoding method proposed in [17]. Based on $G_k^g$, we can easily determine $f_{i,j}^{G_k^g}$, $r_{i,j}^{G_k^g}$, $\varphi_{i,j}^{G_k^g}$ and $\tau_{i,j}^{G_k^g}$ of $\Pi_k^g$ over task $T_j$, where $j \in \{1, 2, \ldots, K\}$. Afterwards, we encode each solution $\Pi_k^g$ in $P^g$ into another permutation $\Pi_k^{g+1}$ based on its decoded DAG form $G_k^g$ (see details in Sect. 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 to 12, which are repeated until a maximum generation $g_{\text{max}}$ is reached. During each iteration, we generate an offspring population $P^{g+1}$ via multiple NHMs using Algorithm 2 (see details in Section IV-C). Again, the same decoding and encoding techniques are employed to these solutions in $P^{g+1}$. Afterwards, we evaluate the fitness $f_{i,j}^{\Pi_k^{g+1}}$ of solutions in $P^{g+1}$ 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 $P^{g+1}$ by combining the current population $P^g$ and the offspring population $P^{g+1}$. Consequently, we update $r_{i,j}^{\Pi_k^{g+1}}$, $\varphi_{i,j}^{\Pi_k^{g+1}}$ and $\tau_{i,j}^{\Pi_k^{g+1}}$ of the combined population $P^{g+1}$, and keep half of the population $P^{g+1}$ based on $\varphi_{i,j}^{\Pi_k^{g+1}}$. When the maximum generation $g_{\text{max}}$ is reached, the algorithm returns the best $\Pi_j^g$ 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 \( T = \{ \{a, b\}, \{c, f\} \} \) and a service repository \( SR \) consisting of six atomic services: \( S_0 = \{ \{e, f\}, \{g\}, QoS_{S_0} \} \), \( S_1 = \{ \{b, c, d\}, QoS_{S_1} \} \), \( S_2 = \{ \{c\}, \{e, QoS_{S_2}\} \} \), \( S_3 = \{ \{d\}, \{f\}, QoS_{S_3} \} \), \( S_4 = \{ \{a\}, \{b\}, QoS_{S_4} \} \) and \( S_5 = \{ \{e\}, \{f\}, QoS_{S_5} \} \). 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 \([1, 2, 3, 4, 0, 5]\).

**Algorithm 2.** Multiple NHMs learning and sampling over \( K \) tasks

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

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 \( A_q \), where \( 1 \leq q \leq (2K - 1) \). Afterwards, we assign these encoded solutions to these pools based on the solutions’ skill factors \( \varphi^{\Pi_k^g} \). For example, if \( \varphi^{\Pi_k^g} = 1 \), 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 \( T_2 \). Afterwards, we learn \( 2K - 1 \) NHMs from the \( 2K - 1 \) pools respectively (see details in Subsection IV-D). The iteration part comprises lines 5 to 12. This iteration will not stop until \( m \) new solutions are constructed to form the offspring population \( P_{\text{a}}^{g+1} \). During the iteration, \( \text{rsp} \) 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 \( P_{\text{a}}^{g+1} \) required in line 6 of Algorithm 1.

---

**Fig. 4:** Decoding a permutation into a DAG

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 \( T \), we remove it from the permutation and add it to the DAG with a connection to \( Start \). Afterwards, we continue checking on services’ inputs by using the \( T \) 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 \([0, 4, 5]\) corresponds to the remaining atomic services in \( SR \), 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 NHMs 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 Sect. III-B, i.e., \( T_1, T_2, T_3 \), and \( T_4 \), 7 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 \( \text{rsp} \) is used to determine whether multitasking or single-tasking NHMs are selected for sampling. Particularly, a value of \( \text{rsp} \) 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.
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. Node Histogram-Based Sampling can effectively sample new and good candidate composite services from every Node Histogram Matrix learnt in each generation. This is because the learnt 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 $A_q$ at generation $g$, denoted by $N\mathcal{H}M_q^g$, is an $n \times n$-matrix with entries $e_{i,r}^g$ as follows:

$$e_{i,r}^g = \sum_{k=0}^{m-1} \delta_{i,r}(\Pi_q^g) + \varepsilon \quad (7)$$

$$\delta_{i,r}(\Pi_q^g) = \begin{cases} 1 & \text{if } \pi_k = r \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

where $i, r = 0, 1, \ldots, n-1$, $\varepsilon = \frac{m}{n} b_{ratio}$ is a predetermined bias, and $n = |SR|$. Roughly speaking, entry $e_{i,r}^g$ counts the number of times that service index $\pi_k$ appears in position $r$ of the permutation over all solutions in pool $A_q$.

Example 2. Let's consider a pool $A_q$ at generation $g$. This pool is assigned with $m$ permutations. For $m = 6$, an example of $A_q^g$ may look as follows.

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

Consider $b_{ratio} = 0.2$, $m = 6$, and $n = 6$, then $\varepsilon = 0.24$.

Thus, we can calculate $N\mathcal{H}M_q^g$ as follows:

$$N\mathcal{H}M_q^g = \begin{bmatrix} 2.24 & 1.24 & 1.24 & 0.24 & 2.24 & 0.24 \\ 0.24 & 3.24 & 1.24 & 2.24 & 0.24 & 0.24 \\ 2.24 & 2.24 & 0.24 & 2.24 & 0.24 & 0.24 \\ 0.24 & 0.24 & 2.24 & 0.24 & 4.24 & 0.24 \end{bmatrix}$$

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 $A_q^g$. The decimal part 0.24 = 0.2/($6 - 1$) is the bias $\varepsilon$.

Once we have computed $N\mathcal{H}M_q^g$, we use node histogram-based sampling (NHBSA) [58] to sample new candidate solutions $\Pi_k^{g+1}$ for the population $P_a^{g+1}$, see Algorithm 5 in Appendix C for technical details. Afterwards, the same decoding part discussed in Sect. 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 Eq. (9)). According to the fitness function in Eq. (9) with respect to $T_j$, we guarantee that $f_{j}^g$ of an infeasible individual falls below 0.5 while $f_{j}^{\Pi}$ of a feasible individual stays above 0.5. Eq. (11) quantifies the violation of $interval_j$ by measuring how far it is from $QoSM_j$ in Eq. (10). In particular, an infeasible individual that violates $interval_j$ more should be penalized more.

$$f_{j}^g = \begin{cases} 0.5 + 0.5 * F(II) & \text{if } QoSM_j \in interval_j, \\ 0.5 * F(II) - 0.5 * V_j(II) & \text{otherwise}. \end{cases} \quad (9)$$

$$V_j(II) = \begin{cases} QoSM_j - QoSM_j & \text{if } QoSM_j \leq QoSM_j, \\ QoSM_j - QoSM_j & \text{otherwise}. \end{cases} \quad (11)$$

with $\sum_{k=1}^{n} w_k = 1$. We can adjust the weights according to the preferences of user segments. $MT$ and $SIM$ are normalized values calculated within the range from 0 to 1 using Eq. (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 Eq. (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 with augmented benchmark datasets 2 (i.e., WSC08-1 to WSC08-8 and WSC09-1 to WSC09-5 with increasing service repository

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

2The two augmented benchmarks for automated web service composition are available from https://github.com/chenwangnida/Dataset
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 $r_{sp}$ 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 Eq. (5) and Eq. (10) follow PMFEA [26]: $w_1$ and $w_2$ are set equal to 0.25, and $w_3$, $w_4$, $w_5$, $w_6$ are all set to 0.125, these weights are set to properly balance QoS and QoS: $w_7$ and $w_8$ are set to 0.5, these weights are set to balance all quality criteria in QoS. 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

\begin{table}[h]
\centering
\caption{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)}
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\textbf{Method} & \textbf{PMFEA-EDA} & \textbf{PMFEA-EDA-WTO} & \textbf{EDA [20]} & \textbf{FL [16]} & \textbf{PathSearch [43]} \\
\hline
WS08-1 & 0.163244 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 \\
WS08-2 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 \\
\hline
WS08-3 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 \\
WS08-4 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 \\
\hline
WS08-5 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 \\
WS08-6 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 \\
WS08-7 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 \\
WS08-8 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 & 0.16324 ± 0.000179 \\
\hline
\end{tabular}
\end{table}
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)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC09-1</td>
<td>0.198185 ± 0.000037</td>
<td>0.198177 ± 0.000286</td>
<td>0.198640 ± 0.000545</td>
<td>0.196700 ± 0.000314</td>
<td>0.194974 ± 0.000276</td>
<td>0.188890</td>
</tr>
<tr>
<td>WSC09-2</td>
<td>0.156212 ± 0.000008</td>
<td>0.144181 ± 0.000646</td>
<td>0.148790 ± 0.000848</td>
<td>0.154844 ± 0.000137</td>
<td>0.146254 ± 0.000294</td>
<td>0.137122</td>
</tr>
<tr>
<td>WSC09-3</td>
<td>0.586664 ± 0.000103</td>
<td>0.147050 ± 0.000193</td>
<td>0.150053 ± 0.000885</td>
<td>0.156733 ± 0.000142</td>
<td>0.153555 ± 0.000268</td>
<td>0.141060</td>
</tr>
<tr>
<td>WSC09-4</td>
<td>0.142097 ± 0.000047</td>
<td>0.139684 ± 0.000559</td>
<td>0.140255 ± 0.000673</td>
<td>0.141026 ± 0.000673</td>
<td>0.141451 ± 0.000515</td>
<td>0.135814</td>
</tr>
<tr>
<td>WSC09-5</td>
<td>0.145391 ± 0.000037</td>
<td>0.143159 ± 0.000389</td>
<td>0.143447 ± 0.000946</td>
<td>0.145045 ± 0.000227</td>
<td>0.144942 ± 0.000705</td>
<td>0.137544</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC08-1</td>
<td>0.254625 ± 0.000574</td>
<td>0.206235 ± 0.003689</td>
<td>0.809369 ± 0.004796</td>
<td>0.810490 ± 0.002694</td>
<td>0.810843 ± 0.002556</td>
<td>0.261980</td>
</tr>
<tr>
<td>WSC08-2</td>
<td>0.761226 ± 0.000654</td>
<td>0.74019 ± 0.001315</td>
<td>0.752848 ± 0.006956</td>
<td>0.74704 ± 0.005348</td>
<td>0.75489 ± 0.006475</td>
<td>0.722883</td>
</tr>
<tr>
<td>WSC08-3</td>
<td>0.77392 ± 0.000505</td>
<td>0.755821 ± 0.002864</td>
<td>0.760774 ± 0.006192</td>
<td>0.778246 ± 0.001529</td>
<td>0.761924 ± 0.000815</td>
<td>0.727531</td>
</tr>
<tr>
<td>WSC08-4</td>
<td>0.741242 ± 0.000261</td>
<td>0.738214 ± 0.000567</td>
<td>0.738966 ± 0.000853</td>
<td>0.739946 ± 0.000232</td>
<td>0.739826 ± 0.000652</td>
<td>0.733738</td>
</tr>
<tr>
<td>WSC08-5</td>
<td>0.747183 ± 0.000759</td>
<td>0.738834 ± 0.000293</td>
<td>0.739396 ± 0.001217</td>
<td>0.739431 ± 0.000255</td>
<td>0.739407 ± 0.000735</td>
<td>0.730400</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC09-1</td>
<td>0.226227 ± 0.001127</td>
<td>0.221145 ± 0.000919</td>
<td>0.219863 ± 0.001344</td>
<td>0.229341 ± 0.000369</td>
<td>0.221582 ± 0.000946</td>
<td>0.206492</td>
</tr>
<tr>
<td>WSC09-2</td>
<td>0.111737 ± 0.000241</td>
<td>0.097486 ± 0.001454</td>
<td>0.109708 ± 0.00659</td>
<td>0.107177 ± 0.00283</td>
<td>0.10468 ± 0.00046</td>
<td>0.080942</td>
</tr>
<tr>
<td>WSC09-3</td>
<td>0.222370 ± 0.000197</td>
<td>0.215991 ± 0.000525</td>
<td>0.217783 ± 0.005575</td>
<td>0.222212 ± 0.00034</td>
<td>0.216098 ± 0.000535</td>
<td>0.088376</td>
</tr>
<tr>
<td>WSC09-4</td>
<td>0.102367 ± 0.000634</td>
<td>0.099225 ± 0.001065</td>
<td>0.099976 ± 0.006056</td>
<td>0.0999127 ± 0.001467</td>
<td>0.0981797</td>
<td></td>
</tr>
<tr>
<td>WSC09-5</td>
<td>0.099847 ± 0.000716</td>
<td>0.094434 ± 0.000876</td>
<td>0.096685 ± 0.002181</td>
<td>0.096785 ± 0.000271</td>
<td>0.096102 ± 0.000248</td>
<td>0.081344</td>
</tr>
</tbody>
</table>

An Intel Core i7-4770 CPU (3.4GHz) and 8 GB RAM. This hardware configuration is used for all the methods compared in this paper.

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 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.

Table 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 1 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] are 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.

Lastly, 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). Table 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.

Firstly, 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 pre-stored service dependency graph. However, efficiency is not the focus of this paper 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.

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

![Figures showing mean fitness over generations for tasks 1-4, for WSC08-8 and WSC09-2.](image)

(Note: the larger the fitness the better)
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 behaviours of the effectiveness of all the methods in Fig. 5.

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 Eq. (5) across 500 and 200 generations for WSC08-8 and WSC09-2, respectively.

In Fig. 6 (a) and (b), PMFEA-EDA focuses more on exploitation 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 vary 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. Conclusions

In this paper, we introduced a new permutation-based multifactorial evolutionary algorithm based on Estimation of Distribution Algorithm 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 NMHs 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 NMHs. 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-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-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.

971

972 References


Chen Wang received his B.Eng degree from Jiangsu University, China (2010), and his MBA degree from National Institute of Development Administration, Thailand (2015). He received his PhD degree in Engineering from Victoria University of Wellington, New Zealand (2020). He is currently a lecturer at the School of Engineering and Computer Science at Victoria University of Wellington. His research interests include service composition, resource allocation in cloud, distributed computing, evolutionary computation, and conceptual modelling. He has served as a PC member for about 90 international conferences, including seven times as a PC chair for conferences such as DE, DEXA, and APCCM.

Hui Ma received her B.E degree from Tongji University, China (1989) and her Ph.D degree from Massey University, New Zealand (2008). She is currently an Associate Professor in Software Engineering at Victoria University of Wellington, New Zealand. Her research interests include service composition, resource allocation in cloud, distributed computing, evolutionary computation, and conceptual modelling. She has also served as a PC member for about 90 international conferences, including leading journals and conferences in databases, service computing, cloud computing, evolutionary computation, and conceptual modelling. She has also served as a PC chair for conferences such as ER, DEXA, and APCCM.

Gang Chen obtained his B.Eng degree from Beijing Institute of Technology in China and PhD degree from Nanyang Technological University (NTU) in Singapore, respectively. He is currently a senior lecturer in the School of Engineering and Computer Science at Victoria University of Wellington. His research interests include evolutionary computation, reinforcement learning, multi-agent systems, and cloud and service computing. He has more than 120 publications, including leading journals and conferences in databases, service computing, cloud computing, evolutionary computation, and conceptual modelling. He has also 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.

Sven Hartmann received his Ph.D in 1996 and his D.Sc. in 2001, both from the University of Rostock (Germany). From 2002 to 2007 he worked first as an associate professor, then full professor for information systems at Massey University (New Zealand). Since 2008 he is a full professor of computer science and chair for databases and information systems at Clausthal University of Technology (Germany). There he is also serving as academic dean at the Faculty of Mathematics, Informatics and Mechanical Engineering. Sven has more than 150 publications.

He served as a PC member for more than 80 conferences, including 10 times as PC chair. His research interests include database systems, big data management, conceptual modelling, and combinatorial optimization.
A. Assortative Mating

The procedure of assortative mating for breeding offspring for K composition tasks is outlined in Algorithm 3. As addressed in [27], the principle of assortative mating is that individuals are more likely to mate those associated with the same skill factors. Meanwhile, implicit knowledge of promising individuals is allowed to be transferred across tasks by crossover. Apart from that, rand is predefined to balance exploitation and exploration.

Algorithm 3. Assortative Mating [27]

1: Randomly select two parents \( \Pi^g_a \) and \( \Pi^g_b \) from \( P^g \);
2: \( \text{rand} \leftarrow \text{Rand}(0, 1) \);
3: if \( \tau^\Pi^g_a = \tau^\Pi^g_b \) or \( \text{rand} < \text{rmp} \) then
4: Perform crossover on \( \Pi^g_a \) and \( \Pi^g_b \) to generate two children \( \Pi^g_c \) and \( \Pi^g_d \);
5: else
6: Perform mutation on \( \Pi^g_a \) to generate one child \( \Pi^g_e \);
7: Perform mutation on \( \Pi^g_b \) to generate one child \( \Pi^g_f \);

B. Vertical Cultural Transmission

In MFEA [27], only one task is evaluated for any child produced by assortative mating. This task is determined by the vertical cultural transmission that is outlined in Algorithm 4, which allows cultural (i.e., skill factor) to be inherited from parents. Therefore, any produced child will only be evaluated on the inherited task.

Algorithm 4. Vertical Cultural Transmission [27]

1: if \( \Pi^g_k \) is produced by two parents \( \Pi^g_a \) and \( \Pi^g_b \) then
2: Generate a random \( \text{rand} \) between 0 and 1;
3: if \( \text{rand} < 0.5 \) then
4: \( \Pi^g_k \) imitates the skill factor \( \tau^\Pi^g_a \) of \( \Pi^g_a \);
5: \( \Pi^g_k \) is only evaluated on task \( T_{\tau^\Pi^g_a} \);
6: else
7: \( \Pi^g_k \) imitates the skill factor \( \tau^\Pi^g_b \) of \( \Pi^g_b \);
8: \( \Pi^g_k \) is only evaluated on task \( T_{\tau^\Pi^g_b} \);
9: else
10: Let \( \Pi^g_\text{e} \) be the only one parent of \( \Pi^g_k \);
11: \( \Pi^g_k \) imitates the skill factor \( \tau^\Pi^g_\text{e} \) of \( \Pi^g_\text{e} \);
12: \( \Pi^g_k \) is only evaluated on task \( T_{\tau^\Pi^g_\text{e}} \);

C. Node Histogram-Based Sampling Algorithm

Node Histogram-Based Sampling Algorithm (NHBSA) [58] is proposed to sample new candidate solutions from a learned \( NHM^g \). Particularly, NHBSA starts with sampling an element for a random position of a permutation with a probability calculated based on elements of \( NHM^g \), and recursively continue sampling other elements of other positions in the permutation.

Algorithm 5. NHBSA [58]

Input : \( NHM^g \)
Output: a sequence of service index \( \Pi^g_k \)
1: Generate a random position index permutation \( r[] \) of \( [0, 1, ..., n-1] \);
2: Generate a candidate list \( C = [0, 1, ..., n-1] \);
3: Set the position counter \( p \leftarrow 0 \);
4: while \( p < n - 1 \) do
5: Sample node \( x \) with probability \( \frac{e_x^{\Pi^g_k}}{\sum_{j \in C} e_j^{\Pi^g_k}} \);
6: Set \( c[r[p]] \leftarrow x \) and remove node \( x \) from \( C \);
7: \( p \leftarrow p + 1 \);
8: \( \Pi^g_{k+1} \leftarrow c[] \);
9: return \( \Pi^g_{k+1} \);