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 relevant 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 of multiple requests. In this paper, we model the problem of collectively handling multiple service composition requests as a new multi-tasking 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 QoS and QoSM.

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 form of service execution workflows, providing value-added functionalities to end users. Web service composition is a promising research area and is highly desirable with increasing number of services available in GIS services [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 composite services that stand for the service composition solutions [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [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 either a single optimization objective [17], [18], [19], [20], [12], [14], [16], [25] or multiple objectives [22], [23], [13], [15], [24]. With the significant increase in service composition requests, one common disadvantage of these methods is that many service requests have to be dealt with repetitively and independently. In fact, similarities across those service requests that could be dealt with collectively have been consistently ignored among 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 each user segment present 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 compositions services for different user segment. TripPlanner is a service composition design system that produces composite booking services for many traveling companies. See an example of two composite booking services utilized by TripPlanner in Figure 1. Both two 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 Figure 1, some services composed in composite booking service A (i.e.,
The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

![Fig. 1: Two composite booking services produced by TripPlanner](image)

(a) Composite booking service A

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 QoM. This is important to cater for different users with varied QoS and QoM requirements. For example, large international travel companies (i.e., platinum segment users of TripPlanner), often care about their customers’ needs more than small local traveling 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 are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.

The problem demonstrated above is clearly a multi-tasking service composition problem. In line with this problem, we specifically consider how to handle such problems with multiple user segments collectively, by providing high-quality services. These high-quality services often cater to a reliable and accurate user experience. In other words, composite services with high QoS are usually provided by composite service B in Figure 1. In contrast, composite services with low QoS are usually provided by composite service A, preferred by bronze segment users of small local traveling companies. For example, service developers, they should distinguish different types of companies and provide different segment offers (i.e., composite services) to different segment.
the second one does not strictly follow any workflow, and it constructs workflows simultaneously with atomic service selections. These two strategies result in two groups of works: semi-automated web service composition and fully web service composition, respectively. In this section, we review some recent works in these two groups. Fig. 2 shows a diagram to guide our discussion of the related works.

Fig. 2: An overview of related work.

A. Literature on semi-automated web service composition

Semi-automated web service composition assumes that an abstract service composition workflow is given, and all the composite services produced by the composition system must strictly obey this workflow. Therefore, semi-automated QoS-aware web service composition turns to select concrete services for each abstract service in the given workflow to achieve the best possible QoS. For example, [29] investigated different QoS estimation models for service selection. Herein we give a short review on the semi-automated web service composition because fully automated service composition is the focus of our paper.

1) Literature on single-tasking non-EC semi-automated web service composition: Non-EC service composition techniques do not rely on bio-inspired techniques. They target optimal composite services by some methods, such as Integer Linear Programming (ILP), dynamic programming, and local search. ILP is used to achieve single-objective semi-automated web service composition. Generally, an ILP model is created with three inputs: a set of decision variables, an objective function and a set of constraints. The outputs of ILP are decision variables and values of maximized/minimized objective function. ILP is flexible for handling QoS constraints and optimizing problems for semi-automated QoS-aware service composition [30], [31], [32]. For example, Yoo et al. [32] formulate the web service composition problem based on a zero-one ILP model introduced in [31]. They take both QoS and constraints on QoS into account. However, due to the larger number of decision variables, ILP may lead to exponentially increased complexity and cost in computation [33]. Besides that, QoS of composite services in ILP-based approaches is calculated by summing up the individual QoS score of every component services. Such a QoS calculation is not always appropriate because the availability of composite services should be calculated by multiplying the availability of every component service.

2) Literature on single-tasking EC-based semi-automated web 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-objective and multi-objective single-tasking EC-based semi-automated web service composition. The first subgroup aims to find composite services with an optimized united score, which is often computed using a simple additive weighting (SAW) technique [34]. For example, some works jointly optimize QoS and QoS as unified score [35], [36], [37]. On the other hand, the second subgroup aims to produce a set of trade-off composite services over multiple objectives. For example, two trade-off objectives, i.e., time and cost, are independently optimized in [38].

The single objective and multiple objectives are optimized using different EC algorithms, e.g., Genetic Algorithm (GA) [39], [36], [38], [40], [41] and Particle Swarm Optimization (PSO) [23], [42]. Single-objective GA is a classic optimization technique to search for high-quality solutions via a population-based solution improvement framework. It has been popularly applied to single-objective semi-automated web service composition with vector-based representations [39], [36], [41]. On the other hand, multi-objective GA has been widely used for the semi-automated web service composition [38], [40]. For example, Liu et al. [38] propose a service composition model, i.e., multi-constraint and multi-objective optimal path, where only the sequence composition construct is supported. In their work, different paths, i.e., composite services, are searched by GA. Wada et al. [40] investigate a semi-automated approach with a vector-based representation. Each vector presents three composite services for three user groups. Two multi-objective GAs (called $E^2$-MOGA and $X-E^3$) are proposed in this work. Particularly, $E^2$-MOGA is designed to search for equally distributed Pareto-optimal solutions in the multi-objective space, while $X-E^3$ is designed to search for Pareto-optimal solutions that can reveal the maximum range of trade-offs, covering extreme solutions in the search space.

3) Literature on multi-tasking EC-based semi-automated web service composition: A new EC computing paradigm, namely, multi-fatorial evolutionary algorithm (MFEA) [27], is recently introduced by Gupta et al. MFEA is proposed to 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 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], [43], [44], [45]. To meet the efficiency and cost requirements, [11] reported the first attempt that employ MFEA to solve multiple service composition tasks together, [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 paper, we will proposed a multi-factorial evolutionary algorithm (PMFEA) to solve more than two fully automated service composition tasks concurrently.

### B. Literature on fully automated web service composition

Different from semi-automated service composition, fully automated service composition does not rely on any existing workflow. Instead, a composite service workflow will be constructed from scratch while selecting and connecting concrete atomic services from the service repository. 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 predefined workflows.

1) Literature on non-EC based fully automated web service composition: Graph search [46], [28], [47], [48], [49], [50] 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 complexity service dependencies. This issue can get even worse when QoS optimization is considered [51]. A* search [52] is utilized to search composite services presented as paths, which are constructed from a sub-graph of a service dependency graph [53]. This sub-graph is extracted based on service requests. However, this work only focuses on minimizing the number of component services in composite services without considering QoS or QoSM. Besides that, the scalability of this method suffers when the service repository grows. To address this critical issue, [46] proposes QoS-aware service composition via a scalable way of pruning dependency graphs, and a novel path-based construction and selection method. This method can efficiently construct near-optimal composite services. However, it only considers a single quality criterion in QoS. To consider multiple quality criteria in QoS, a recent work, named PathSearch [28], proposes an improved path-based search method based on [46]. Partly, 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 [28].

2) Literature on single-tasking EC-Based fully automated web service composition: Evolutionary single-tasking 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], [19], [25] or multi-objective EC techniques [13], [15]. The first subgroup aims to find composite services with an optimized united score. For example, a comprehensive quality score that combines QoSM and QoS [17]. On the other hand, the second subgroup aims to produce a set of trade-off composite services over multiple objectives. For example, two trade-off objectives are investigated in [13]: one combines cost and time, and the other combines availability and reliability.

In single-objective single-tasking, most of existing service composition approaches use 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, [54] proposes a graph-based evolutionary algorithm to evolve DAG-based composite services directly with DAG-based crossover and mutation operators. Tree-based composite solutions in [12], [14], [19], [32] are also produced using implicit knowledge defined by one or more variations of GP-based genetic operators on parent individuals. [16] works on permutations (i.e., indirect representation of composite services) with permutation-dependent genetic operators to produce high-quality composite solutions. Apart from these conventional EC techniques, other approaches [20], [18] works on Estimation of Distribution Algorithm that uses the explicit knowledge of promising solutions encoded by the distribution of promising solutions. These methods often make the search more effective and efficient.

For example, [20] samples 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 a NHM of promising solutions does help to find near-optimal solutions. The same author also investigates the use of Edge Histogram Matrix (EHM) of service dependencies to learning explicit knowledge of promising solutions. However, this approach suffers from a scalability issue when the size of the service repository is double of the reported size in [18].

In multi-objective single-tasking, there are very limited works on both multi-objective and fully automated service composition, while many works [22], [23] are reported on multi-objective semi-automated service composition. To the best of our knowledge, [13], [15], [24] are the three recent attempts along this research direction. In [15], a fragmented tree-based representation is proposed in NSGA-II with the representation-dependent genetic operators. Later on, the same authors proposes a hybrid approach that combines NSGA-II [55] and MOEA/D [56] with permutation-based representation. This approach enables the use of single-objective local search technique (e.g., swap-based operator) can be applied in many decomposed single-objective subproblems. Very recently, an EDA-guided local search is 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].

In summary, traditional EC techniques have shown their promises in solving fully automated web service composition by implicit knowledge learning that relies genetic operators to produce new candidate solutions from promising parent solutions. On the other hand, EDA has been demonstrated to...
be more effective at solving this problem by explicitly learning distributions of promising solutions and sampling high-quality solutions.

3) Literature on multitasking EC-Based fully automated web service composition: As we discussed in Sect. II-A3, [11] reported the first attempt to optimize QoS for two unrelated service requests simultaneously in semi-automated service composition. To overcome the limitations in [11], [26] proposed a multi-factorial evolutionary algorithm (PMFEA) to solve more than two fully automated service composition tasks concurrently. Compared to single-tasking approaches, this method only requires only a fraction of time. However, this work does 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 paper 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 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 $QoS_S = \{t_S, c_S, r_S, a_S\}$ is a set of non-functional attributes of $S$. The inputs in $I_S$ and outputs in $O_S$ are parameters modeled through concepts in a domain-specific ontology $O$. The attributes $t_S, c_S, r_S, a_S$ refer to the response time, cost, reliability, and availability of service $S$, respectively, which are four commonly used QoS attributes [57].

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 described 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 [58]. For concepts $a, b \in O$ the matching returns exact if $a$ and $b$ are equivalent ($a \equiv b$), plugin if $a$ is a sub-concept of $b$ ($a \sqsubseteq b$), super-concept if $a$ is a super-concept of $b$ ($a \sqsupseteq b$), and fail if none of the previous matchmaking types is returned. In this paper, we are only interested in exact and plugin matches for robust compositions. As argued in [37], 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 [59] is a link between two matched services $S$ and $S'$, denoted as $S \rightarrow S'$, if an output $a (a \in O_S)$ of $S$ serves as the input $b (b \in O_{S'})$ of $S'$ satisfying either $a \equiv b$ or $a \sqsubseteq b$. For concepts $a, b \in O$, the semantic similarity $sim(a, b)$ is calculated based on the edge counting method in a taxonomy like WordNet [60]. Advantages of this method are simple calculation and accurate measure [60]. Therefore, the matchmaking type and semantic similarity of a robust causal link is defined as follows:

$$type_{\text{link}} = \begin{cases} 1 & \text{if } a \equiv b \text{ (exact match)} \\ \frac{p}{1} & \text{if } a \sqsubseteq b \text{ (plugin match)} \end{cases}$$

$$sim_{\text{link}} = \frac{2N_a}{N_a + N_b}$$

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 of $a$ and $b$ to the top concept of the ontology $O$, respectively. However, if more than one pair of matched output and input exist from service $S$ to service $S'$, $type_{\text{link}}$ and $sim_{\text{link}}$ will take on their average values.

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

$$MT = \prod_{j=1}^m type_{\text{link}}$$

$$SIM = \frac{1}{m} \sum_{j=1}^m sim_{\text{link}}$$

Formal expressions as in [61] are used to represent service compositions. The constructors $\bullet$, $| |$, $+$ and $\ast$ are used to denote sequential composition, parallel composition, choice, and iteration, respectively. The set of composite service expressions is the smallest collection $SC$ that contains all atomic services and that is closed under sequential composition, parallel composition, choice, and iteration. That is, whenever $C_0, C_1, \ldots, C_d$ in $SC$, then $C_0 \bullet (C_1, \ldots, C_d)$, $| | (C_1, \ldots, C_d)$, and $+C_0$ are in $SC$, 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.

In the presentation of this paper, we mainly focus on two constructors, sequence $\bullet$ and parallel $| |$, similar as the most automated service composition works [62], [14], [54], [63] do, where composite services are represented as directed acyclic graphs (DAGs). Its nodes correspond to those services (also called component services) in service repository $SR$ that are used in the composition. Let $G = (V, E)$ 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, $G$ does not contain all services in $SR$. The decoded DAG allows easy calculation of QoS in Table I and presents users with a complete workflow of service execution [20]. For example, 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 [34]:

\[ F(C) = w_1 MT + w_2 SIM + w_3 A + w_4 R + w_5 (1 - T) + w_6 (1 - CT) \]  \hspace{1cm} (5)

with \( \sum_{k=1}^{K} w_k = 1 \) (\( w_0 \)). This objective function is defined as a comprehensive quality model for service composition. We can adjust the weights according to the user’s preferences. Different from the composition task defined in the single-tasking semantic web service composition [26], which is discovered using a greedy value of 0 and a maximum value of 1, the minimum and maximum of each quality criterion based on its relative importance is considered as an evolutionary multitasking problem that aims to find the best possible solution for a given composition task \( M \). This problem is also defined in [26]. WSC-MQP 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 of 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 \( \pi \).

The goal of multitasking semantic web service composition is to find the \( K \) best possible solutions concurrently with one for each user segment.

### C. Multifactorial Optimization

MFEA is a new evolutionary paradigm 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 of 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 \( \pi \).

**Definition 1:** The factorial cost \( f^I_j \) of individual \( \pi \) measures the fitness value with respect to the \( K \) tasks, where \( j \in \{1, 2, \ldots, K\} \).

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

**Definition 3:** The scalar fitness \( \varphi^I_j \) of individual \( \pi \) is calculated based on its best factorial rank over the \( K \) tasks, which is given by \( \varphi^I_j = 1/min\{r^I_j(1, 2, \ldots, K)\} \).

**Definition 4:** The skill factor of individual \( \pi \) denotes the most effective task of the \( K \) tasks, and is given by \( \tau^I_j = \arg\min_j\{r^I_j\} \), 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
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 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. These every two adjacent tasks are identified as the most suitable tasks for knowledge sharing. Therefore, learning multitasking NHMs of these additional pools allow 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 to 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 Figure 4. From the current population in Figure 4, 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

We first randomly initialize $m$ permutation-based $\Pi_k^g$ solutions, where $0 \leq k < m$ and $g = 0$. Each solution is represented as a random sequence of service indices ranging from 0 to $|S^R|-1$, and $S^R$ is a service repository containing registered web services. For example, a permutation is represented as $\Pi = (\pi_1, \ldots, \pi_n)$ such that $\pi_k \neq \pi_d$ 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 $f^{\Pi_k^g}$, $r^{\Pi_k^g}$, $\varphi^{\Pi_k^g}$ and $\tau^{\Pi_k^g}$ 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 a NHM [20]. The iterative part of PMFEA-EDA comprises lines 6 to 12, which are repeated until a maximum generation $g_{max}$ 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 $f^{\Pi_k^{g+1}}$ of solutions in $P^{g+1}$ on the task related to the imitated tasks skill factor, which is determined in the principle of vertical culture transmission [27]. In particular, the skill factor of produced every solution is determined based on its corresponding NHM, where it is sampled from. We then produce the next population $P^{g+2}$ by combining the current population $P^{g}$ and the offspring population $P^{g}_{a}$, respectively, and $\Pi_k^{g+1}$ of the combined population $P^{g+1}$. And keep half of the population $P^{g+1}$ based on $\varphi^{\Pi_k^{g+1}}$. When the maximal generation $g_{max}$ is met, the algorithm returns the best $\Pi_j^g$ over all the generations for $T_j$. 

B. Permutation-based representation

Permutations have been 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.
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 of indexes of component services in DAG, sorted in ascending order based on the longest distance from Start to every component services of DAG while the second part is indexes of remaining services in permutation not utilized by the DAG, see details in [20].

Let us consider a composition task $T = \{(a, b), (e, f)\}$ and a service repository $SR$ consisting of six atomic services.

1. $S_0 = \{(e, f), (g), \text{QoS}, S_0\}$
2. $S_1 = \{(b), (c, d), \text{QoS}, S_1\}$
3. $S_2 = \{(c), (e), \text{QoS}, S_2\}$
4. $S_3 = \{(d), (f), \text{QoS}, S_3\}$
5. $S_4 = \{(a), (h), \text{QoS}, \}$
6. $S_5 = \{(e), (f), \text{QoS}, \}$

The two special services Start $= (\emptyset, (a, b), \emptyset)$ and End $= (\{e, f\}, \emptyset, \emptyset)$ are defined by a given composition task $T$.

Figure 5 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 $T_I$, 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 $T_I$ 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. Afterwards, this DAG is encoded as a new permutation $[1, 2, 3, 4, 0, 5]$ consisting of two parts: one part corresponds to $[1, 2, 3]$ 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 $SR$, 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 from promising solutions for sampling new candidate solutions. 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 $r_{sp}$ is used to determine whether multitasking or single-tasking NHMs are selected for sampling. Particularly, a value of $r_{sp}$ 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.
Algorithm 2. Multiple NHMs learning and sampling over K tasks

Input : $P^g$
Output: $P^{g+1}_a$
1: Initialize a set of empty $A_q$ for each task and every two adjacent tasks;
2: Assign each solution $\Pi^g_q$ in $P^g$ to $A_q$ based on its skill factor $\varphi^g_q$;
3: Learn $2K - 1$ NHMs $\mathcal{N}^T M^q_k$ from the $2K - 1 - A_q$;
while $|P^{g+1}_a| \leq m$ do
5: rand $\leftarrow$ Rand(0,1);
6: if rand < $r$sp then
7: Select one NHM from multitasking NHMs randomly;
else
9: Select one NHM from single-tasking NHMs randomly;
10: Sample one solution $\Pi^{q+1}_g$ from the selected NHM and put the solution into $P^{g+1}_a$;
II: $\Pi^{q+1}$ inherits the skill factor based on the selected NHM;
12: Return offspring population $P^{g+1}_a$;

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 $\tau^g_q$. For example, if $\tau^g_q = 1$, this solution $\Pi^g_q$ is assigned to two pools, one for task $T_1$ and the other 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 $P^{g+1}_a$. 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. The dominance of the principal of vertical culture transmission [27]. After all iterations have been completed, Algorithm 2 returns the newly produced population $P^{g+1}_a$ required in line 6 of Algorithm 1.

D. Application of Node Histogram-Based Sampling

We employ the node histogram-based sampling [64] 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.

A NHM learned from solutions in each pool $A_q$ at generation $g$, denoted by $\mathcal{N}^T M^q_g$, is an $n \times n$-matrix with entries $e^q_{i,r}$ as follows:

$$e^q_{i,r} = \sum_{k=0}^{m-1} \delta_{i,r}(\Pi^g_k) + \varepsilon$$

$$\delta_{i,r}(\Pi^g_k) = \begin{cases} 1 & \text{if } \pi_i = r \\ 0 & \text{otherwise} \end{cases}$$

where $i, r = 0, 1, \ldots, n - 1$, $\varepsilon = \frac{m}{|\mathcal{S}|} b_{ratio}$ is a predetermined bias, and $n = |\mathcal{S}|$. Roughly speaking, entry $e^q_{i,r}$ counts the number of times that service index $\pi_i$ appears in position $r$ of the permutation over all solutions in pool $A_q$. Let’s consider a pool $A_q$ at generation $g$. This pool is assigned with $m$ permutations. For $m = 6$, an example of $A^g_q$ may look as follows:

$$A^g_q = \begin{bmatrix} \Pi^g_5 & 1 & 2 & 3 & 4 & 0 \\ \Pi^g_4 & 0 & 1 & 2 & 3 & 4 \\ \Pi^g_3 & 0 & 1 & 2 & 3 & 4 \\ \Pi^g_2 & 4 & 3 & 0 & 1 & 2 \\ \Pi^g_1 & 4 & 3 & 0 & 1 & 2 \\ \Pi^g_0 & 2 & 1 & 3 & 0 & 4 \end{bmatrix}$$

Consider $b_{ratio} = 0.2$, $m = 6$, and $n = 6$, then $\varepsilon = 0.24$. Thus, we can calculate $\mathcal{N}^T M^q_g$ as follows:

$$\mathcal{N}^T M^q_g = \begin{bmatrix} 2.24 & 1.24 & 1.24 & 0.24 & 0.24 & 0.24 \\ 0.24 & 3.24 & 1.24 & 2.24 & 0.24 & 0.24 \\ 2.24 & 2.24 & 0.24 & 2.24 & 0.24 & 0.24 \\ 0.24 & 2.24 & 0.24 & 4.24 & 6.24 \end{bmatrix}$$

We use one entry $e^g_{0,0} = 2.24$ as an example to demonstrate the meaning behind this value. The integer part 2 presents that service $S_0$ appears twice at the first position over all the permutations in $A^g_q$. The decimal part $0.24 = 6 \times 0.2/(6 - 1)$ is the bias $\varepsilon$.

Once we have computed $\mathcal{N}^T M^q_g$, we use node histogram-based sampling (NHBSA) [64] to sample new candidate solutions $\Pi^{q+1}_g$ for the population $P^{g+1}_a$, see Algorithm 5 in Appendix C for technical details. Afterwards, the same decoding part discussed in Sect. IV-B will be employed on any 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 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 QoS equals 0.3. Based on segment preferences discussed in Figure 3, this composite service is only feasible for just one task (i.e., $T_2$), since it complies with intervals of the other tasks (i.e., $T_1$, $T_3$, and $T_4$) as it violates $\text{interval}_1$, $\text{interval}_3$, and $\text{interval}_4$ 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 $f^j_1$ of an infeasible individual falls below 0.5 while $f^j_2$ of a feasible individual stays above 0.5. Eq. (11) quantifies the violations of $\text{interval}_j$ by measuring how far it is from 0.5.
\[ QoSM(\Pi) = w_7 \hat{MT} + w_8 \hat{SIM} \]

\[ V_j(\Pi) = \begin{cases} QoS_M^j - QoSM(\Pi) & \text{if } QoSM(\Pi) \leq QoS_M^j \\ QoS_M^j - QoSM(\Pi) & \text{otherwise.} \end{cases} \]

\[ \sum_{k=7}^{8} w_k = 1 \]

To find the K best possible solutions with one for each task, the goal of multi-tasking semantic web service composition is to maximize the objective function in Eq. (9) concerning the K tasks.

V. EXPERIMENTAL EVALUATION

In this section, we employ a quantitative evaluation approach for studying the effectiveness and efficiency of PMFEA-EDA with augmented benchmark datasets (i.e., WSC08-1 to WSC08-8 and WSC09-1 to WSC09-5 with increasing service repository SR) used by the recent study.

Particularly, compared to single-tasking EDA, multitasking methods are more likely to evolve a well diversified population size of 30. Therefore, each task have 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 [28] follow the common settings reported in the literature. For PathSearch [28], 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 reports 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 equally 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 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 observe similar results to these reported below.

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

A. Comparison of the Fitness

Independent-sample T-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 were found to be better, similar, or worse than the others.

Consequently, we can rank all the competing methods and highlight the top performance in a green color.

In addition, PMFEA-EDA also outperformed PMFEA-WTO, PMFEA, EDA [20], and FL [16] for each run, i.e., 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 have 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 [28] follow the common settings reported in the literature. For PathSearch [28], 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 reports 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 equally 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 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 observe similar results to these reported below.

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

A. Comparison of the Fitness

Independent-sample T-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 were found to be better, similar, or worse than the others.

Consequently, we can rank all the competing methods and highlight the top performance in a 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 perform 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 outperformed PMFEA-EDA-WTO significantly, and is labeled as a 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 [28] 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 [28] is designed to make locally best choice over the \( k \) services at each step, and gradually built a path-based composite solution.
B. Comparison of the Execution Time

Independent-sample T-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 execution time and the standard deviation over 30 repetitions for all tasks solved by PMFEA-EDA, PMFEA-EDA-WOT, PMFEA-EDA, EDA and FL, and the value of execution time over 1 run for all tasks solved by PathSearch.

Firstly, PathSearch [28] requires the least execution time. This is because PathSearch [28] only searches the constructed path based on $k$ best services from a pre-stored service dependency graph. However, the efficiency is not the focus of this manuscript because finding high-quality composite services at the design stage is our focus.

Apart from PathSearch [28], PMFEA-EDA, PMFEA-EDA-WOT, and PMFEA appear to be very 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-WOT requires slightly less execution time for all tasks since PMFEA-EDA demands more time in learning NHMs when service repository $SR$ becomes larger and larger. However, the extra time incurred in PMFEA-EDA-WOT is not substantial compared to other multitasking methods.

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 QoS 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 all tasks.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC08-1</td>
<td>0.190450 ± 0.004210</td>
<td>0.18920</td>
<td>0.190450</td>
<td>0.190450</td>
<td>PathSearch [28]</td>
</tr>
<tr>
<td>WSC08-2</td>
<td>0.207646 ± 0.004210</td>
<td>0.207646</td>
<td>0.207646</td>
<td>0.207646</td>
<td>PathSearch [28]</td>
</tr>
<tr>
<td>WSC08-3</td>
<td>0.190450 ± 0.004210</td>
<td>0.190450</td>
<td>0.190450</td>
<td>0.190450</td>
<td>PathSearch [28]</td>
</tr>
<tr>
<td>WSC08-4</td>
<td>0.207646 ± 0.004210</td>
<td>0.207646</td>
<td>0.207646</td>
<td>0.207646</td>
<td>PathSearch [28]</td>
</tr>
</tbody>
</table>

Table II: Mean fitness values of solutions per task for our approaches in comparison to PMFEA [26], EDA [20], FL [16] and PathSearch [28] (Note: the higher the fitness the better)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC08-1</td>
<td>0.18920</td>
<td>0.18920</td>
<td>0.18920</td>
<td>0.18920</td>
<td>PathSearch [28]</td>
</tr>
<tr>
<td>WSC08-2</td>
<td>0.207646</td>
<td>0.207646</td>
<td>0.207646</td>
<td>0.207646</td>
<td>PathSearch [28]</td>
</tr>
<tr>
<td>WSC08-3</td>
<td>0.190450</td>
<td>0.190450</td>
<td>0.190450</td>
<td>0.190450</td>
<td>PathSearch [28]</td>
</tr>
<tr>
<td>WSC08-4</td>
<td>0.207646</td>
<td>0.207646</td>
<td>0.207646</td>
<td>0.207646</td>
<td>PathSearch [28]</td>
</tr>
</tbody>
</table>

Figure 6 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 PMFEA-EDA-WOT, and eventually reaches the highest plateau. This observation matches well with our expectation that knowledge sharing across tasks is very effective.

C. Comparison of the Convergence Rate

We also studied the convergence rate of PMFEA-EDA, PMFEA-EDA-WTO, PMFEA, EDA [20], and FL [16]. Using WSC08 and WSC09 as two examples, we show the behaviours of effectiveness of all the methods in Figure 6.
### Table III: Mean fitness values of solutions per task for our approaches in comparison to PMFEA [26], EDA [20], FL [16] and PathSearch [28] (Note: the higher the fitness the better)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC09-1</td>
<td>0.198189 ± 0.005760</td>
<td>0.198173 ± 0.002266</td>
<td>0.192864 ± 0.003534</td>
<td>0.196386 ± 0.003613</td>
<td>0.193947 ± 0.003276</td>
<td>0.138690</td>
<td></td>
</tr>
<tr>
<td>WSC09-2</td>
<td>0.19211 ± 0.008066</td>
<td>0.141811 ± 0.006846</td>
<td>0.148379 ± 0.004478</td>
<td>0.140544 ± 0.003147</td>
<td>0.140256 ± 0.002946</td>
<td>0.137212</td>
<td></td>
</tr>
<tr>
<td>WSC09-3</td>
<td>0.158664 ± 0.010383</td>
<td>0.147505 ± 0.019213</td>
<td>0.150503 ± 0.038855</td>
<td>0.156733 ± 0.011425</td>
<td>0.15653 ± 0.002688</td>
<td>0.140610</td>
<td></td>
</tr>
<tr>
<td>WSC09-4</td>
<td>0.142907 ± 0.008467</td>
<td>0.139684 ± 0.003539</td>
<td>0.140255 ± 0.007073</td>
<td>0.140256 ± 0.006673</td>
<td>0.141451 ± 0.005315</td>
<td>0.135814</td>
<td></td>
</tr>
<tr>
<td>WSC09-5</td>
<td>0.134218 ± 0.001307</td>
<td>0.145159 ± 0.008398</td>
<td>0.141417 ± 0.000946</td>
<td>0.140456 ± 0.002797</td>
<td>0.144942 ± 0.007073</td>
<td>0.137544</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC09-1</td>
<td>0.813679 ± 0.002658</td>
<td>0.863253 ± 0.006699</td>
<td>0.893469 ± 0.007696</td>
<td>0.921176 ± 0.001414</td>
<td>0.901746 ± 0.002918</td>
<td>0.629909</td>
<td></td>
</tr>
<tr>
<td>WSC09-2</td>
<td>0.761220 ± 0.006036</td>
<td>0.749199 ± 0.003154</td>
<td>0.752846 ± 0.008508</td>
<td>0.747994 ± 0.005348</td>
<td>0.749595 ± 0.004265</td>
<td>0.732688</td>
<td></td>
</tr>
<tr>
<td>WSC09-3</td>
<td>0.777922 ± 0.003505</td>
<td>0.755821 ± 0.008864</td>
<td>0.769476 ± 0.006192</td>
<td>0.77266 ± 0.001526</td>
<td>0.769124 ± 0.006194</td>
<td>0.727531</td>
<td></td>
</tr>
<tr>
<td>WSC09-4</td>
<td>0.741262 ± 0.002561</td>
<td>0.738314 ± 0.005657</td>
<td>0.739866 ± 0.000653</td>
<td>0.739946 ± 0.000253</td>
<td>0.739826 ± 0.000692</td>
<td>0.737338</td>
<td></td>
</tr>
<tr>
<td>WSC09-5</td>
<td>0.741730 ± 0.007756</td>
<td>0.738344 ± 0.006829</td>
<td>0.73936 ± 0.002177</td>
<td>0.739431 ± 0.000255</td>
<td>0.739406 ± 0.007073</td>
<td>0.734809</td>
<td></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 [28] (Note: the shorter the time the better)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC09-1</td>
<td>rand ± 1092</td>
<td>907 ± 596</td>
<td>510 ± 434</td>
<td>1375 ± 367</td>
<td>124 ± 14</td>
<td>0 ± 9</td>
<td></td>
</tr>
<tr>
<td>WSC09-2</td>
<td>rand ± 2262</td>
<td>1243 ± 87</td>
<td>654 ± 43</td>
<td>1700 ± 367</td>
<td>74 ± 10</td>
<td>0 ± 9</td>
<td></td>
</tr>
<tr>
<td>WSC09-3</td>
<td>rand ± 4102</td>
<td>2089 ± 51</td>
<td>600 ± 43</td>
<td>2700 ± 367</td>
<td>43 ± 10</td>
<td>0 ± 9</td>
<td></td>
</tr>
<tr>
<td>WSC09-4</td>
<td>rand ± 5982</td>
<td>3025 ± 51</td>
<td>900 ± 43</td>
<td>3700 ± 367</td>
<td>36 ± 10</td>
<td>0 ± 9</td>
<td></td>
</tr>
<tr>
<td>WSC09-5</td>
<td>rand ± 7052</td>
<td>4000 ± 51</td>
<td>1300 ± 43</td>
<td>4700 ± 367</td>
<td>30 ± 10</td>
<td>0 ± 9</td>
<td></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 [28] (Note: the shorter the time the better)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC09-1</td>
<td>rand ± 1092</td>
<td>907 ± 596</td>
<td>510 ± 434</td>
<td>1375 ± 367</td>
<td>124 ± 14</td>
<td>0 ± 9</td>
<td></td>
</tr>
<tr>
<td>WSC09-2</td>
<td>rand ± 2262</td>
<td>1243 ± 87</td>
<td>654 ± 43</td>
<td>1700 ± 367</td>
<td>74 ± 10</td>
<td>0 ± 9</td>
<td></td>
</tr>
<tr>
<td>WSC09-3</td>
<td>rand ± 4102</td>
<td>2089 ± 51</td>
<td>600 ± 43</td>
<td>2700 ± 367</td>
<td>43 ± 10</td>
<td>0 ± 9</td>
<td></td>
</tr>
<tr>
<td>WSC09-4</td>
<td>rand ± 5982</td>
<td>3025 ± 51</td>
<td>900 ± 43</td>
<td>3700 ± 367</td>
<td>36 ± 10</td>
<td>0 ± 9</td>
<td></td>
</tr>
<tr>
<td>WSC09-5</td>
<td>rand ± 7052</td>
<td>4000 ± 51</td>
<td>1300 ± 43</td>
<td>4700 ± 367</td>
<td>30 ± 10</td>
<td>0 ± 9</td>
<td></td>
</tr>
</tbody>
</table>

### Appendix

#### 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 exploration and exploitation.

#### Algorithm 3. Assortative Mating [27]

1. Randomly select two parents $\Pi_1^g$ and $\Pi_2^g$ from $\mathcal{P}^g$;
2. $rand \leftarrow \text{Rand}(0, 1)$;
3. if $\frac{rand}{\Pi_1^g} \leq \frac{rand}{\Pi_2^g}$ then
   4. Perform crossover on $\Pi_1^g$ and $\Pi_2^g$ to generate two children $\Pi_1^{g+1}$ and $\Pi_2^{g+1}$;

**else**

6. Perform mutation on $\Pi_1^{g+1}$ to generate one child $\Pi_1^{g+1}$;
7. Perform mutation on $\Pi_2^{g+1}$ to generate one child $\Pi_2^{g+1}$;
Fig. 6: 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

C. Node Histogram-Based Sampling Algorithm

Node Histogram-Based Sampling Algorithm (NHBSA) [64] is proposed to sample new candidate solutions from a learned $NHM^9$. Particularly, NHBSA starts with sampling an element for a random position of a permutation with a probability calculated based on elements of $NHM^9$, and recursively continue sampling other elements of other positions in the permutation.

ALGORITHM 5. NHBSA [64]

Input : $NHM^9$
Output: a sequence of service index $\Pi_{k}^{p}$
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 $\frac{e^{\varphi_{s}[r[p], x]}}{e^{\varphi_{s}[r[p], x]}}$
6: Set $c[r[p]] \leftarrow x$ and remove node $x$ from $C$;
7: $p \leftarrow p + 1$;
8: $\Pi_{k}^{p+1} \leftarrow c[]$;
9: return $\Pi_{k}^{p+1}$;
Wang et al.: Using an estimation of distribution algorithm to achieve multitasking semantic web service composition

Chen Wang received his B.Eng degree from Jiaotong 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 data scientist at the National Institute of Water and Atmospheric Research, New Zealand. His research interests include evolutionary computation and machine learning for combinatorial optimization.

Hui Ma Dr. Hui Ma received her B.E. degree from Tongji University and her Ph.D degrees from Massey University. She is currently an Associate Professor in Software Engineering at Victoria University of Wellington. Her research interests include service composition, resource allocation in cloud, conceptual modelling, database systems, resource allocation in clouds, and evolutionary computation in combinatorial optimization. Hui has more than 150 publications, including leading journals and conferences in databases, service computing, cloud computing, evolutionary computation, and conceptual modelling. She has served as a PC member for more than 10 international conferences, including seven times as a PC chair for conferences such as EK, DEXA, and APCCM.

Gang Chen obtained his B.Eng degree from Beijing Institute of Technology in China (2000) and his PhD degree from Nanyang Technological University (NTU) in Singapore (2006) respectively. From 2000 to 2001, he worked as Software Engineer at Founder Electronics Pte. Ltd, Beijing China. From 2005 to 2006, he worked as Software Engineer at Crimsonlogic Pte. Ltd, Singapore. From 2006 to 2007, Chen worked as Research Fellow at the Information Communication Institute of Singapore (ICIS), School of Electrical and Electronic Engineering (EEE) at NTU.

From 2007 to 2010, I worked as teaching fellow and visiting assistant professor at school of EEE, NTU. From 2010 to 2012, he worked as lecturer and postgraduate programme leader in the Department of Computing at Unitec Institute of Technology. Since 2012, he became a senior lecturer in the School of Engineering and Computer Science at Victoria University of Wellington.

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). He is working as a senior researcher 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.