Robustness Estimation and Optimisation for Semantic Web Service Composition with Stochastic Service Failures

Chen Wang, Hui Ma, Gang Chen, Sven Hartmann and Jüergen Branke

Abstract—Service-oriented architecture (SOA) is a widely adopted software engineering paradigm that encourages modular and reusable applications. One popular application of SOA is web service composition, which aims to loosely couple web services to accommodate complex goals not achievable through any individual existing web service. Many approaches have been proposed to construct composite services with optimized quality, Quality of Service (QoS) and/or Quality of Semantic Matchmaking (QoSM), assuming that QoS of web services seldom or never changes. However, the composition services generated by these approaches may not perform well when QoSs of their component services change, and even may not be executable at execution time due to the failure of its component services. Therefore, it is important to build Web service compositions that are robust to stochastic service failures. The challenges of building robust service compositions are to efficiently generate service composition with near-optimal quality in a large search space of available services, and to efficiently and accurately measure the robustness of composite service considering all possible failure scenarios. In this paper, we propose a two-stage GA approach to robust web service composition, which can generate robust baseline composite services at the design phase, and efficiently and effectively handle stochastic service failures at execution phase to maintain the good quality of the composite services. In particular, we propose an archive-based adaptive evolutionary control over two sequential evolutionary stages to efficiently and effectively produce baseline solutions. Further, we develop an efficient robustness measurement based on fewer carefully selected scenarios that serve as an important lower bound of the robustness over all the possible scenarios. We have conducted experiments with benchmark datasets to evaluate the performance of our proposed approach. Our experiments show that our method can produce high-robustness composite services, achieving outstanding performance (i.e. high QoS and QoMS) consistently in the event of stochastic service failures, on service repositories with varying sizes.

Index Terms—Web service composition, Robust optimisation, Combinatorial optimisation, Genetic Algorithm.

1 Introduction

Web services are modular, self-describing, self-contained applications that are available on the Internet, serving as reusable software components in the domain of service-oriented computing [1]. Since a single web service often cannot satisfy users' complicated requirements, web service composition aims to provide added values by constructing a composite service via loosely coupling many web services over the Internet. To ensure the validity of service compositions, all the inputs of the coupled web services must be fulfilled. In service composition, *fully automated service composition* has attracted much attention, and it simultaneously constructs service workflows and selects services, without strictly obeying any predefined workflows [2]. Many researchers in this field have been working on fully automated service composition with the aim to optimise the overall quality of composite services. This quality often refers to the functional and non-functional requirements, i.e., *Quality of Semantic Matchmaking* (QoSM) and *Quality of service* (QoS). Many relevant studies [3, 4, 5, 6, 7, 8, 9] focus on *static web service composition*, assuming that the QoS of the elementary web services remains stable. Such a rigid assumption makes these works less practical for handling service composition problems in the context of a dynamic service environment, where QoS is changing from time to time. [10]. Therefore, *dynamic web service composition* is introduced to emphasise on the challenges caused by the dynamic and unpredictable nature of QoS in web service composition at the run-time.

Services available for composition can experience QoS changes at any time. On the one hand, new services are published, and old ones are modified or removed due to the changes in users' demands [11]. In fact, newly published services might be more suitable because they are faster, cheaper and aggregate multiple functionalities required by a composite service. Those changes either render existing composite services invalid or present new opportunities for building more desirable composite services. Therefore, delivering composite services with reliable QoS is a critical and significant challenge in the dynamic environment. In practice, QoS changes can be related to many different QoS criteria [12], such as response time, throughput, failure probability, availability, price and popularity. Among these QoS criteria, the stochastic failure of web services is the most critical uncertainty [13]. This is because the composite service discovered at the design phase can become completely useless at the time of its execution if any component service fails.

To consider stochastic service failures in service composition, many works [14, 15, 16] repair the composite services while re-optimising QoS of composite services at the execution phase in the event of service failures, without abandoning ongoing composite services completely. However, these

- *Chen Wang, Hui Ma, and Gang Chen are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand.*
  E-mail: {chen.wang, hui.ma, aaron.chen}@ecs.vuw.ac.nz
- *Sven Hartmann is with the Department of Informatics, Clausthal University of Technology, Germany.*
  E-mail: sven.hartmann@tu-clausthal.de
- *Jüergen Branke is with the Warwick Business School, University of Warwick, UK.*
  E-mail: Juergen.Branke@wbs.ac.uk

approaches ignore the importance of building robust composite services at the design phase. These composite services can handle stochastic service failures in a robust manner at the execution phase. Therefore, our recent work [17] studied robust service composition problem for effectively handling stochastic service failures. This work constructed composite service with the aim to optimise the robustness in terms of expected QoS and QoSM at the design phase. Such a robust solution serves as the blueprint/baseline and is expected to continue to work reliably or be easily re-optimised with negligible impact on quality at the execution phase. In this paper, we will continue to investigate the same problem that focuses on constructing robust composite services in the event of stochastic service failures.

it is hard to measure the robustness of candidate solutions in terms of expected QoS and QoSM, because it is very time-consuming to calculate the true robustness of an individual by enumerating all the failure scenarios. In the literature, one conventional approach for measuring the robustness is to simulate the scenarios exhaustively. However, such a simulation method often requires large computing resources. To tackle this issue, robustness estimation is usually employed to approximate the robustness of candidate solutions. For example, Wang et al. [17] estimated the robustness of candidate composite services through Monte Carlo sampling [18], i.e., an unweighted averaged fitness value over a set of randomly sampled scenarios. However, Monte Carlo sampling often requires a large sample size to be determined manually for a good trade-off between algorithm performance and sample cost. Moreover, this robust estimation method is only tested on a small service composition benchmark, i.e., OWLS-TC [19]. OWLS-TC contains multiple composition tasks over a small-size service repository with 946 web services, which also indicates small dimensions of decision variables.

When dealing with service composition over a large service repository (i.e., a service repository that consists of a large number of services), the complexity of the fitness landscape will increase dramatically [20]. This complexity makes the robustness of composite services much harder to be estimated. This is known as the "curse of dimensionality", which may lead to the deterioration of the performance in robustness estimation. For example, Monte Carlo sampling can be computationally costly for evaluating a single solution. This is because it requires a much larger sampling size to ensure the estimated robustness is sufficiently accurate. When the dimension of decision variables grows, it becomes actually infeasible to estimate the robustness.

In general, to tackle such a high-dimensional robust optimisation problem, fitness approximation methods are usually employed in evolutionary computation (EC). This approximation-assisted approach aims to find a good trade-off between the accuracy of estimation and the efficiency. Two critical technical challenges in designing an approximation-assisted EC method are *what fitness approximation method to be used for the fitness estimation*, and *how to integrate the approximate method into the optimisation process*. We will discuss these two challenges for solving our robust web service composition problem one by one.

Regarding the first challenge, three types of approximation methods, i.e., problem approximation, data-driven functional approximation, and fitness inheritance, are usually employed in the literature [21, 22]. The first type of approximation methods often uses an approximate, easier-to-solve problem to replace the original problem. For example, Monte Carlo sampling, instead of complete sampling, is used to sample dynamic scenarios in our recent work [17]. The second type of approximation methods trains explicit models (also called meta-models or surrogates) based on historical solutions using various meta-modelling techniques, such as polynomials or gaussian processes. However, adopting such a technique needs sufficient historical data that maps between the design parameters and the quality of the design. The third type of approximate methods estimates the fitness of one individual by the fitness of other similar individuals. However, in our problem, it is hard to define a similarity or distance measure between any two composite services. This is because composite services that serve the same functionality can differ in terms of both the component services and workflow structures that integrate the component services together. Moreover, when the dimension of decision variables increases, distance measures can become less useful. In a nutshell, problem approximation can help because it can simplify the simulations over service failures. However, a more accurate estimation with fewer samples is one of the major challenges in this paper.

The second challenge is how to integrate the approximate models into the optimisation process. Evolutionary control is used to decide whether an actual or approximation method should be utilised for fitness evaluations [21]. Jin et al. [21] grouped existing works into two categories: individual-based and generation-based evolutionary control. The first category allows some of the individuals to be evaluated by the actual model while the others are evaluated based on the estimation model. The second category restricts all the individuals in a certain generation to be either evaluated by the actual model or the estimation model. Although there are no clear advantages of one category over the other, generation-based evolutionary control is more suitable to be implemented in parallel and can achieve good performance with fewer control interventions based on generations, rather than individuals. Moreover, an adaptive frequency over the generations should be considered because the fidelity of the approximate model may vary significantly during the optimisation [21].

To address these two challenges above, we propose an EC-based approach to robust web service composition with fitness approximation that effectively handles stochastic service failures. Genetic Algorithm (GA) is a popular EC technique that has enabled the tackling of several challenging service composition problems [5]. Therefore, GA is utilised as an EC technique to generate high-robustness baseline services in this paper. This approach can achieve outstanding performance in both effectiveness and efficiency when the size of the service repository increases dramatically. These outstanding performances are observed by comparing it with some state-of-the-art works for finding robust composite services using large benchmarks. The contributions of this paper are listed below.

1) To perform an efficient and accurate robustness estimation for handling stochastic service failures, we introduce a new fitness approximation method via fewer selected scenarios for calculating the robustness of service composition at the design phase. Different from using randomly sampled scenarios for robustness estimation in [17], this method carefully selects scenarios with the aim to reduce the variance of the robustness estimation, especially when a high-dimensional robust

optimization problem is considered. Moreover, to leverage the influence of the various selected scenarios, the weights of these selected scenarios are considered to measure the importance of the selected scenarios in the robustness estimation.

2) To further reduce the computation time of GA with the fitness approximation and maintain its effectiveness in finding robust composite services, a two-stage GA, denoted GA-2Stage, is proposed with an adaptive evolution control mechanism. Particularly, in stage one, efficient evaluations (i.e., comprehensive quality for measuring the overall quality of composite service under static service composition environment) are employed to find good solutions that are likely to be robust in the event of stochastic service failures. In stage two, these solutions can be further evolved to improve their robustness using our proposed robustness estimation method. Moreover, an archive-based evolutionary control is proposed to determine at which generation stage two should start. Meanwhile, the first generation in stage two is initialised with the archive, which stores good solutions found in stage one. Note that the adaptive evolutionary control mechanism is not introduced in our previous work [17]
3) To demonstrate the effectiveness and efficiency of our GA-2Stage algorithm, we conduct experiments to compare it against three GA-based approaches: one method only employs GA with our proposed robustness estimation method over all the generations (henceforth referred to as GA-RE); the other two existing state-of-the-art methods: one is Monte Carlo sampling-based method that has recently been developed in [17] (henceforth referred to as GA-MC); the other is Fixed Length GA [5] (henceforth referred to as FL) that achieves outstanding performance in finding high-quality solutions in static composition environment. Our experiment results show that our method can produce high-robustness composite services, achieving outstanding performance (i.e. high QoS and QoMS) consistently in the case of stochastic service failures, regardless of the size of the service repository.

2 Related Work

In this section, we review some state-of-the-art EC-based approaches for web service composition with the objectives of optimizing QoS or QoSM of composite services. Afterwards, we discuss some state-of-the-art works in developing robustness measures and a few EC works with fitness estimation methods.

2.1 Literature on web service composition problem

EC techniques have been used to automatically generate composite services with optimized QoS and/or QoSM [3, 5, 4, 23, 7, 8, 9]. These works can be divided into two groups based on the assumptions on QoS: QoS of web service are either static or dynamic.

The first group usually assumes that the QoS of web services seldom changes or does not change at all. In this group, QoS often refers to the mean values of the historical QoS, which is accessible to service users. This field has been widely studied in the past few years, focusing on developing effective and efficient EC techniques to find composite services with optimised QoS. To achieve such a goal, researchers have been working on developing new and effective representations of composite services for EC techniques, such as DAG-based [24, 17, 25, 26], tree-based [4, 7, 27, 28, 29], tree-like based [7], and permutation-based representations [5, 23, 8]. The majority of these works also propose different representation-dependent genetic operators to explore large searching spaces, while some of them [8, 17] propose new sampling techniques for breeding new promising solutions from the learned distributions (i.e., Node or Edge Histogram Matrix) of historical solutions. However, all these works may not achieve desirable runtime performance as a result of using component services with changing QoS.

The second group focuses on handling dynamic QoS. Dynamic QoS values vary in bounded-interval values or can be estimated based on the past QoS distributions. For example, some works [14, 15, 16] rely on the bounded-interval QoS values to simulate periodically changed QoS while re-optimising the QoS of composite services periodically. However, the QoS changes that are assumed to happen after every fixed time are the victim of idealisation. Several concurrent works, such as [30], propose a fuzzy-based QoS model based on the bounded-interval QoS values to measure the uncertainties in QoS. A few works [31, 32, 33, 34] assume that the changes of QoS follow some historical patterns and can be predicted in the future. For example, [31, 34] assume that the QoS follows a known probability distribution, and can be estimated based on the past QoS values. However, these approaches do not consider the impact of time on QoS. To address this limitation, [35] consider time-varying QoS by proposing a time-series prediction model while optimising the predictive QoS of composite services. Such a problem is formulated as predictive-trend-aware service composition by the same authors in [36]. They further conduct extensive case studies with diverse randomly-generated composition workflows. Although these works are capable of handling QoS in the future in a predictive manner, they often require sufficient historical data that are not always available for newly registered web services. Therefore, their prediction model may become less reliable to predict the QoS in the future.

In this paper, we study the robust web service composition problem at the design phase. A robust composite service is expected to continue to work reliably or be easily repaired with negligible impact on quality through a fast local search technique. Our preliminary study of this problem has been reported in [17]. Some interesting ideas have been explored in some works that handle service failures through distributed service deployment [37, 38]. For example, a sufficient number of duplicated services can be jointly deployed to prevent un-predicted service failures. These works are related but clearly targeted to address a different problem than our paper. Moreover, some dynamical reconfiguration methods [13, 39] have been investigated by searching for a replacement of the failed component services and/or some of their neighbouring services at the execution phase. Similar ideas are also explored to simultaneously consider forward and backward recovery, and service cancellability at execution phase in [40, 41]. However, those methods do not focus on simultaneously addressing multiple concerns in dynamic web service composition, which include: (1) handling fully-automated service composition problem, where a service composition workflow is unknown, (2)

taking into account the importance of building robust composite services at the design phase, (3) optimizing QoS in the event of service failures.

2.2 Literature on fitness approximation and EC with fitness estimation

Fitness approximation has been widely used to solve computationally expensive single-objective and multi-objective problems [21, 22]. Existing fitness approximation techniques can be classified into three types: problem approximation, data-driven functional approximation, and fitness inheritance in the literature [21, 22]. As we discussed in Sect.1, problem approximation can better serve our needs to estimate the robustness of composite services. In the literature, fitness approximation has been utilised to find solutions with optimised robustness. This optimisation problem is called a robust optimisation problem. In fact, decision-makers concern not only the performance of the solution but also the sensitivity of performance with respect to small changes in the environment. In robust optimisation problems, each solution evaluation can be computed based on the simulations, which can be highly time-consuming. For example, EC techniques usually employ the expected fitness over disturbances via either explicit averaging or implicit averaging techniques [21] for finding high-robustness solutions. Such techniques based on the averaged fitness values are not always practical due to the required computational resources [42]. Therefore, fitness approximation is used for reducing the number of evaluations.

Fitness approximation is utilised in the evolutionary optimisation of expensive problems for the purpose of reducing computational time. In EC with fitness approximation, evolutionary control is a technique to manage fitness approximation for the evaluation of individuals. Evolutionary control aims to achieve a good trade-off between less accurate fitness evaluations and computational cost by replacing costly fitness functions with the fitness approximation. Techniques in evolutionary control can be grouped into individual-based and generation-based [21, 22]. In the individual-based, some of the individuals are evaluated using the fitness approximation while the others are evaluated using the real fitness function. For example, [43] proposed an evolutionary control that ensures individuals with good estimated fitness values will be re-evaluated on the real fitness function. These good individuals are selected from individuals in each individual cluster of each population. In contrast, in the generation-based evolutionary control, all individuals in one generation are either evaluated using the fitness approximation or the real fitness function. Recently, much attention has been paid to adaptive adjustment of the frequency of using the fitness approximation in either individual-based or generation-based evolutionary control. As we discussed in Sec. 1, a fixed evolutionary control frequency can be unpractical as the fidelity of the approximate model may vary significantly during the optimisation [21]. For example, an adaptive generation-based evolutionary control is proposed based on the error of the fitness approximation [44].

In our work, like many real-world robust optimisation problems, "real fitness function" actually does exist, but it is not feasible to compute the robustness of the composite services over all possible events of service failures. Therefore, a fitness approximation method will be proposed and play the role of "real fitness function". Besides that, an efficient comprehensive quality evaluation method is suggested in some generations to further reduce the overall execution time of our EC-based approach. This comprehensive quality evaluation will be utilised to efficiently find good solutions that are likely to have high robustness. Consequently, these two evaluation methods are adaptively employed based on our proposed generation-based evolutionary control in this paper.

3 Preliminaries

3.1 Web Service Composition Problem

Table 1 shows a list of abbreviations and acronyms used in the formulations of web service composition problem.

TABLE 1: List of Abbreviations and Acronyms.

Abbreviation	Description
C	Composite service
$type_{link}$	matchmaking type of a robust causal link
MT	matchmaking type of a composite service
sim_{link}	semantic similarity
SIM	semantic similarity of a composite service
pr_S	Service failure probability
$QoSM$	Quality of semantic matchmaking
QoS	Quality of service
t_S, c_S, r_S, a_S	Response time, cost, reliability, and availability of S
pr_S	Failure probability of S
r_C, a_C, ct_C, t_C	Aggregated response time, cost, reliability, and availability of C
f_{cq}	Overall quality of composite services
r_{LB}	Lower bound robustness estimation
r_{MC}	Monte Carlo robustness estimation
S	Semantic web service
$\mathcal{SR}$	Service repository

A *semantic web service* (*service*, for short) is a tuple $S = (I_S, O_S, QoS_S)$ where I_S is a set of service inputs that are consumed by S, O_S is a set of service outputs that are produced by S, and $QoS_S = \{t_S, ct_S, r_S, a_S, pr_S\}$ is a set of non-functional attributes of S. The inputs I_S and outputs O_S are parameters modelled through concepts in a domain-specific ontology $\mathcal{O}$. The attributes t_S, ct_S, r_S, a_S and pr_S refer to the response time, cost, reliability, availability, and *service failure probability* respectively [12]. The first four attributes are commonly used QoS attributes [45]. However, in practice, the execution of a composite service is usually confronted with stochastic service failures [12]. A *service failure probability* pr_S can be approximated by dividing the number of failed invocations by the total number of invocations conducted in the past on service S [10]. Also, pr_S of newly published web services can be estimated as the pr_S of web services hosted by the same service providers in the same location. For any service S hosted by different service providers, its failure probability is assumed to be independent.

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

A *composition task* (also called *service request*) over a given $\mathcal{SR}$ is a tuple $T = (I_T, O_T)$ where I_T is a set of task inputs, and O_T is a set of task outputs. I_T and O_T are parameters that are semantically described by concepts in the ontology $\mathcal{O}$. Two special atomic services $Start = (\emptyset, I_T, \emptyset)$ and $End = (O_T, \emptyset, \emptyset)$ are always included in $\mathcal{SR}$ to account for the input and output of a given composition task T.

We use *matchmaking types* to describe the level of a match between outputs and inputs [46]. For concepts a, b in $\mathcal{O}$ the *matchmaking* returns $exact$ if a and b are equivalent ($a \equiv b$), $plugin$ if a is a sub-concept of b ($a \sqsubseteq b$), $subsume$ if a is a super-concept of b ($a \sqsupseteq b$), and $fail$ if none of previous matchmaking types applies. In this paper we

TABLE 2: QoS calculation for a composite service C

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

are only interested in *exact* and *plugin* matches for robust compositions, see [47]. As argued in [47] *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* [48] 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 $\mathcal{O}$, the *semantic similarity* $sim(a, b)$ is calculated based on the edge counting method in a taxonomy like WorldNet [49]. Advantages of this method are simple calculation and good semantic measurement [49]. As discussed in [48], we use *matchmaking type* ($type_{link}$) and *semantic similarity* (sim_{link}) to denote robust causal link, which is defined as follows:

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

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

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

The *QoSM* of a composite service measured by matchmaking type (MT) and semantic similarity (SIM) is obtained by aggregating all robust causal links as follows:

$$MT = \prod_{j=1}^{m} type_{link_j} \tag{3}$$

$$SIM = \frac{1}{m} \sum_{j=1}^{m} sim_{link_j} \tag{4}$$

Formal expressions as in [50] are used to represent service compositions. The constructors $\bullet$, $\parallel$, $+$ and $*$ are used to denote sequential composition, parallel composition, choice, and iteration, respectively. The set of *composite service expressions* is the smallest collection $\mathcal{SC}$ that contains all atomic services and that is closed under these constructors. That is, whenever $C_0, C_1, \ldots, C_d$ are in $\mathcal{SC}$ then $\bullet(C_1, \ldots, C_d)$, $\parallel (C_1, \ldots, C_d)$, $+(C_1, \ldots, C_d)$, and $*C_0$ are in $\mathcal{SC}$, too. Let C be a composite service expression. If C denotes an atomic service S then its QoS is given by QoS_S. Otherwise the QoS of C can be obtained inductively as summarized in Table 2. Herein, $p_1, \ldots, p_d$ with $\sum_{k=1}^{d} p_k = 1$ denote the probabilities of the different options of the choice $+$, while ℓ denotes the average number of iterations. Therefore, QoS of a service composition solution, i.e., availability (A), reliability (R), execution time (T), and cost (CT) can be obtained by aggregating a_C, r_C, t_C and ct_C as in Table 2.

In the presentation of this paper, we mainly focus on two constructors, sequence $\bullet$ and parallel $\parallel$, similar to most automated service composition works [8, 25, 28, 27, 51], where a *composite service* is often represented in the form of a directed acyclic graph (DAG, denoted as $\mathcal{G}$). In a DAG, nodes represent web services (also called *component services*) and edges represent robust causal links. A *composite service* can also be indirectly represented as a permutation $\Pi = (\pi_0, \pi_1, \ldots, \pi_{n-1})$, elements of which are $\{0, 1, \ldots, n-1\}$ such that $\pi_i \neq \pi_j$ for all $i \neq j$. Each element in Π represents a unique index of a web service in the service repository. According to [8], a permutation Π needs to be interpreted, and can be further decoded into a $\mathcal{G}$ (denoted as $\Pi \Rightarrow \mathcal{G}$). Such a decoding process can ensure the validity of composite services if $\Pi \Rightarrow \mathcal{G}$ holds, see details in Sect. 4.3.

In a static composition environment, QoS of composite services seldom change. To involve multiple quality criteria in decision making in such a static composition environment, 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 [52]:

$$f_{cq}(\Pi) = \begin{cases} w_1 \hat{MT} + w_2 \hat{SIM} + w_3 \hat{A} + w_4 \hat{R} + \\ \quad w_5(1 - \hat{T}) + w_6(1 - \hat{CT}) & \text{if } \Pi \Rightarrow \mathcal{G} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

with $\sum_{k=1}^{6} w_k = 1$. This objective function is defined as a *comprehensive quality*, denoted as f_{cq}, for service composition. We can adjust the weights according to the user's preferences. $\hat{MT}$, $\hat{SIM}$, $\hat{A}$, $\hat{R}$, $\hat{T}$, and $\hat{CT}$ are normalized to 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 minimum value 0 and maximum value 1. The minimum and maximum value of Q_3, Q_4, Q_5, and Q_6 are calculated across all the relevant services. Note that relevant services are discovered based on the composition task over the service repository $\mathcal{SR}$ using greedy search, see details in [5, 27, 28].

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

The goal of the static web service composition is to find a composite service $C^\star$ that maximises the objective function in Eq. (5) for a given composition task T.

3.2 Robust web service composition

In this paper, we consider ***R**obust **W**eb **S**ervice **C**omposition* for handling stochastic **S**ervice **F**ailures (henceforth referred

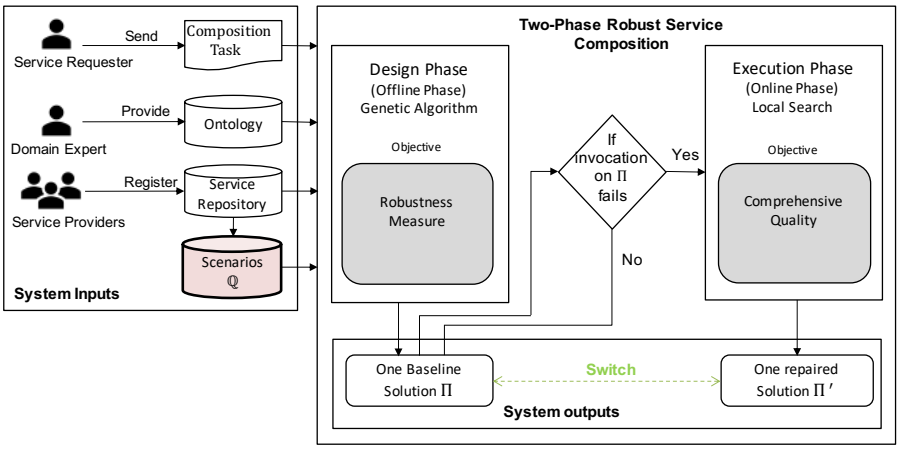

Fig. 1: two-phase robust web service composition and execution process

to as **RWSC-SF**). We recently modelled this problem as a two-phase web service composition problem, consisting of the design phase and the execution phase [17]. In the design phase, it aims to construct baseline composite services (i.e., services to be deployed over the Internet) with optimised robustness by explicitly considering stochastic service failures. The baseline solution is further tested in the execution phase. Note that "tested" in this paper refers to simulated evaluations of a composite service for its execution. Such a solution is expected to continue working reliably or be easily repaired during testing.

To explicitly consider robustness in the design phase, scenario-based simulation methods are often used to evaluate the robustness through the use of a continuous or discrete scenarios set [53]. Our recent work [17] defined the *robustness* of a composite service in the presence of stochastic service failures that create a discrete set of scenarios $\mathbb{Q}$. A *scenario* $Q \in \mathbb{Q}$ corresponds to a set of services $\{S_j\}$ that remain accessible during the execution of a composite service, where $\sum_{Q \in \mathbb{Q}} Pr(Q) = 1$. Let $\mathscr{L}(\Pi, Q)$ be a local search operator (i.e., an efficient re-optimization technique) that produces a new feasible composite solution Π' for Q through applying local changes to Π. The robustness is defined as the expected quality of a composite service across all possible scenarios and can be directly estimated through Monte Carlo sampling [18] as follows:

$$r(\Pi) = \sum_{Q \in \mathbb{Q}} f_{cq}(\mathscr{L}(\Pi, Q)) Pr(Q) \approx \frac{1}{N} \sum_{i=1}^{N} f_{cq}(\mathscr{L}(\Pi, Q_i)) \quad (7)$$

where N is the sample size. Particularly, in Eq. (7), Π is evaluated N times based on N sampled Q_i. $f_{cq}(\Pi)$ measures the comprehensive quality of a composite service defined in Eq. (5).

Our two-phase robust web service composition system is illustrated in Fig. 1. This composition system requires three inputs: a composition task initialised by the service requester, a service repository provided by the service providers, and an ontology defined by the domain experts. At the design phase, a global searching technique, such as an EC method can be utilised to efficiently search for a baseline solution Π with optimised robustness using a fitness function based on any proposed robustness measure, such as Eq. (5). This robust composite service Π serves as an output of a robust service composition system. At the execution phase, the baseline solution will be executed if none of its component services fails. Otherwise, this baseline solution will be repaired through a local search technique to resume its feasibility, and its execution continues thereafter. This repairing process does not guarantee that the solution Π can always be repaired because no composition services $\mathcal{G}$ could be decoded from Π for some scenarios, i.e., $\Pi \Rightarrow \mathcal{G}$ does not hold. In contrast, for other scenarios, a repaired solution Π' is returned and does not depend on any failed services at the time of the execution. Note that the same local search operator is used in both the design phase and execution phase.

4 A GA-2Stage Approach to RWSC-SF

In this section, we first present our fitness approximation method in Sect. 4.1. Subsequently, we will outline the main steps of our two-stage GA-Based approach to robust web service composition in Sect. 4.2. We will also discuss some essential steps in detail, such as representation of composition solutions, evolutionary control, genetic operators, and simulation-based evaluations.

Fitness approximation has been applied with some recent success to estimate the robustness of composite solutions for handling stochastic service failures using Monte Carlo sampling, performing a cheap but not necessarily accurate fitness estimation in the frame of evolutionary computation. The success, however, strongly depends on the accuracy (i.e., the size of sampling) of the approximation that must be investigated in advance to ensure a reliable survival strategy in an evolutionary optimisation process. Apart from the accuracy, the cost of sampling must be determined to ensure a good trade-off. However, when the size of the service repository increases, Monte Carlo sampling may become computationally very expensive for reliably approximating the fitness. This is because a very large sampling size is required to ensure the approximated fitness to be used to distinguish individuals in the evolutionary process. Therefore, a new scenario-based fitness estimation method with fewer sampled scenarios, rather than randomly sampled scenarios, should be developed to achieve an ideal trade-off among accuracy and sampling cost. Particularly, we define an important lower bound of expected fitness as an approximated fitness to be maximised. We expect that improvement in lower bound leads

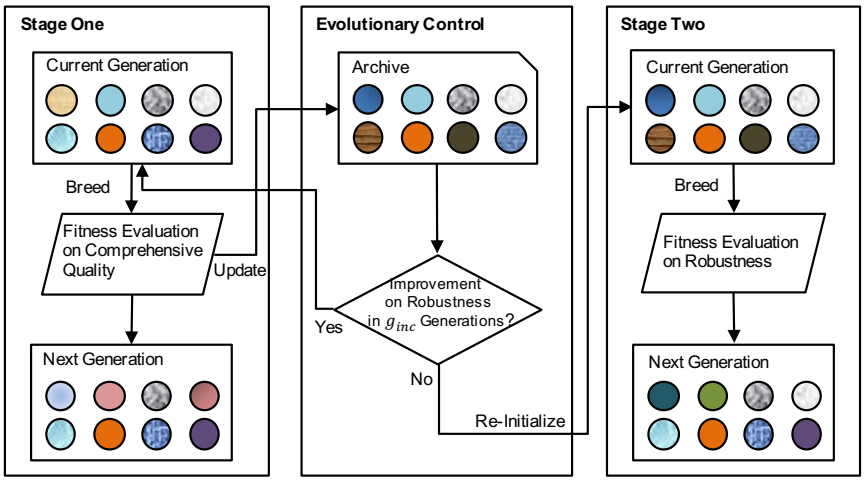

Fig. 2: Two-stage GA with fitness approximation for solving RWSC-SF

to improvement in true robustness with a high probability. This is achieved by carefully selecting different scenarios, each of which only considers one service failure that is more likely to happen, see details in Sect. 4.1. Consequently, the robustness is estimated based on these selected scenarios with weights that measure their importance.

Moreover, to further reduce the computation time without decreasing algorithm effectiveness, a two-stage GA with fitness approximation will be introduced along with the proposed fitness approximation method above. The robustness of evolved composite services in the first stage is more roughly estimated based on individuals' comprehensive quality when no services will become unavailable, while solutions in the second stage are more accurately estimated based on our newly proposed fitness approximation method. More specifically, stage one tries to efficiently find good individuals that are more likely to have high robustness. This is because composite solutions found in stage one can be evolved into solutions with a small number of component services using evaluations on their comprehensive quality. Such solutions found in stage one are unlikely to be affected in the event of service failures due to their relatively small number of component services. Consequently, these good solutions can contribute to better robustness. Therefore, they are stored in an archive and utilised to initialise the first population in stage two. Such initialisation is more beneficial in both effectiveness and efficiency, compared to GA-RE that only employs our proposed robust approximation method through all the generations.

The generation updates and evolutionary control over the two sequential stages of our method are illustrated in Fig. 2. In stage one, an archive is utilised to measure solution improvement in terms of their average robustness. Once the archive does not have any improvement on the robustness in g_{inc} consecutive generations, stage two will start by re-initialising the current generation with solutions in the archive.

4.1 Fitness approximation

As shown in Eq. (7), the robustness is defined as the expected quality of a composite service across all possible scenarios. As discussed previously, we define a lower bound of expected fitness as an approximated fitness to be maximized below:

$$\begin{aligned} r(\Pi) &= \sum_{Q \in \mathbb{Q}} f_{cq}(\mathscr{L}(\Pi, Q)) Pr(Q) \\ &\geqslant \sum_{Q^\star \in \mathbb{Q}^\star} f_{cq}(\mathscr{L}(\Pi, Q^\star)) Pr(Q^\star) \\ &= \sum_{Q^\star \in \mathbb{Q}^\star} f_{cq}(\mathscr{L}(\Pi, Q^\star)) pr_{S_i} \prod_{j \neq i} (1 - pr_{S_j}) \end{aligned} \tag{8}$$

where $\mathbb{Q}^\star \subseteq \mathbb{Q}$ are selected scenarios that only have one service failure, and any $Q^\star \in \mathbb{Q}^\star$ are not identical to each other. Therefore, the total number of scenarios in $\mathbb{Q}^\star$ equals to $|\mathcal{SR}|$. Let S_i be the failed service in every scenario, $Pr(Q^\star)$ can be calculated based on a joint probability of services in $\mathcal{SR}$. Note that, when service repository is very big, this joint probability might result in an arithmetic numerical overflow. To avoid this issue, we can calculate this joint probability in logarithmic space. Thus it becomes a sum.

4.2 Outline of GA-2Stage

Our proposed method is outlined in ALGORITHM 1. GA-2Stage takes five inputs: service composition task $T = (I_T, O_T)$, an ontology $\mathcal{O}$ that describes all the parameters of the web services, and the number of neighbours n_{nb} to be exploited for repairing each solution in the scenarios. In ALGORITHM 1, we start with initializing an empty archive $\mathcal{A}$ that plays the role of evolutionary control, and population $\mathcal{P}^0$ with m randomly generated permutations Π_k^g (where $k = 1, \ldots, m$), see details in Sect. 4.3. In Step 3, we evaluate each permutation in the initialized population by decoding it into an interpreted DAG. The decoding method is actually an application of a forward graph-building algorithm in [5, 8], see details in Sect. 4.3. The DAG enables an easy calculation of the comprehensive quality defined in Eq. (5). The purpose of utilizing this evaluation method has already been discussed at the beginning of this section. In Step 4, we adaptively set up the generation number $g^\star$, at which stage two begins based on an archive-based evolutionary control, see details in Sect. 4.4. The iterative steps (Steps 5 to 15) will be repeated until the maximum number of generations has been reached. During each iteration, $g^\star$ can be adaptively updated to control the lengths of stage one (i.e. $g < g^\star$) and stage two (i.e. $g \geq g^\star$). In stage one, we produce m new

ALGORITHM 1. GA-2Stage for RWSC-SF.

Input : composition task T, Ontology $\mathcal{O}$, service repository $\mathcal{SR}$, and the number of neighbors n_{nb}
Output: a baseline solution

Set generation counter $g \leftarrow 0$;
Initialize an empty archive $\mathcal{A}$ and a $\mathcal{P}^g$ with m random permutations, each represented as a Π_k^g (where $k = 1, \ldots, m$);
Evaluate each permutation in $\mathcal{P}^g$ using Eq. (5) based on its decoded DAG, $\mathcal{G}_k^g$;
Set $g^\star$ based on a updated $\mathcal{A}$ from $\mathcal{P}^g$;
while $g < g_{max}$ **do**
  if $g < g^\star$ **then** `// stage one`
    Populate $\mathcal{P}^{g+1}$ with m permutations from $\mathcal{P}^g$ through the use of genetic operators;
    Evaluate each permutation in $\mathcal{P}^{g+1}$ using Eq. (5);
  if $g = g^\star$ **then** `// stage two starts`
    Populate $\mathcal{P}^{g+1}$ with m permutations from $\mathcal{A}$;
    Evaluate each permutation in $\mathcal{P}^{g+1}$ using Eq. (8);
  if $g > g^\star$ **then** `// stage two`
    Populate $\mathcal{P}^{g+1}$ with m permutations from $\mathcal{P}^g$ through the use of genetic operators;
    Evaluate each permutation in $\mathcal{P}^{g+1}$ using Eq. (8);
  Set $g \leftarrow g + 1$;
Select the best solution Π^{opt} in $\mathcal{P}^g$ as a baseline;

permutations to form the next generation $\mathcal{P}^{g+1}$ by genetic operators, see details in Sect. 4.5. All the permutations in the newly formed population are then evaluated using Eq. (5). The initial population in stage two is constructed from the archive, and all the permutations in $\mathcal{P}^{g+1}$ are then evaluated using the fitness approximation given in Eq. (8). Finally, the best solution with the highest fitness is returned as a baseline solution at the end of the evolutionary process.

4.3 Permutation-based representation

Service permutations have been successfully utilized as indirect representations in the domain of fully automated service composition [5, 8]. Such a permutation, however, needs to be interpreted. For that, a decoding algorithm is used to map a permutation to a DAG. The decoded DAG presents users with a complete workflow of service execution and also allows easy calculation of fitness in Eq. (5).

Example 4.1. Let us consider a composition task $T = (\{a, b\}, \{e, f\})$ and a service repository $\mathcal{SR}$ consisting of six atomic services. $S_0 = (\{e, f\}, \{g\}, QoS_{S_0})$, $S_1 = (\{b\}, \{c, d\}, QoS_{S_1})$, $S_2 = (\{c\}, \{e\}, QoS_{S_2})$, $S_3 = (\{d\}, \{f\}, QoS_{S_3})$, $S_4 = (\{a\}, \{h\}, QoS_{S_4})$ and $S_5 = (\{c\}, \{e, f\}, QoS_{S_5})$. The two special services $Start = (\emptyset, \{a, b\}, \emptyset)$ and $End = (\{e, f\}, \emptyset, \emptyset)$ are defined by a given composition task T. Fig. 5 illustrates an example of producing a DAG from a given permutation $[4, 1, 0, 2, 3, 5]$.

In Fig. 5, we check the satisfaction on the inputs of services in the permutation from left to right. If any services can be immediately satisfied by the provided inputs of composition task I_T, we remove it from the permutation and add it to the DAG with a connection to $Start$. Afterwards,

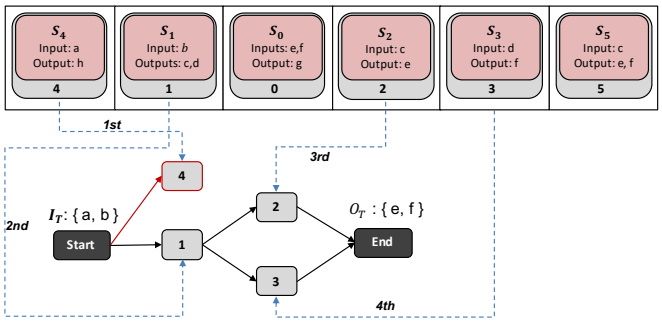

Fig. 3: Decoding a permutation into a DAG

we continue checking on the inputs of services by using the I_T and outputs of the services and add satisfied services to the DAG. We continue this process until we can add End to the graph. In the last phase of the decoding process, some redundant services whose outputs contribute nothing to End will be removed.

4.4 Archive-based adaptive evolutionary control

The archive-based evolutionary control proposed in our GA-2Stage algorithm is used to adaptively update the generation number $g^\star$, at which stage two should begin. Specifically, $g^\star$ should be increased by g_{inc} generations based on the robustness changes in the archive, starting with a predefined generation number. Such an updating mechanism for $g^\star$ allows evolved solutions at the updated generation $g^\star$ achieve the highest possible robustness with the least computation resources due to the cheap evaluation method assigned to stage one.

ALGORITHM 2. Generating $g^\star$ based on an adaptive archive-based evolutionary control.

Input : archive $\mathcal{A}$ with maximal size m, population $\mathcal{P}^g$, initial generation number $g^\star$ and generation increment step g_{inc}
Output: generation number $g^\star$

Set generation counter $g \leftarrow 0$;
if $g < g^\star$ **then**
  Update $\mathcal{A}$ with $\mathcal{P}^g$;
if $g = g^\star$ **then**
  Evaluate each permutation in $\mathcal{A}$ using Eq. (8);
  Calculate average robustness $\overline{R}$ for permutations in $\mathcal{A}$;
  Update $\mathcal{A}$ with $\mathcal{P}^g$;
  Evaluate each permutation in $\mathcal{A}$ using Eq. (8);
  Calculate average robustness $\overline{R}'$ for permutations in $\mathcal{A}$;
  if $\overline{R}' > \overline{R}$ **then**
    $g^\star \leftarrow g^\star + g_{inc}$;
return $g^\star$;

This evolutionary control is outlined in ALGORITHM 2. This algorithm takes three inputs: an archive of size m that stores good individuals, the current population $\mathcal{P}^g$, the generation $g^\star$ at which stage two should start, and generation increment step g_{inc} for $g^\star$. When $g < g^\star$, this algorithm updates the archive by storing all distinct composite services from $\mathcal{P}^g$ based on the comprehensive quality in a descending order. Subsequently, when $g = g^\star$, we evaluate the robustness of each permutation in the archive using Eq. (8)

and calculate the average robustness of all the permutations as $\overline{R}$. After calculating the average robustness $\overline{R}$, we calculate the average robustness again as $\overline{R}'$ after updating the archive. The archive is updated in the same way as we discussed above. Consequently, we increase $g^\star$ by g_{inc} if $\overline{R}' > \overline{R}$. Otherwise, $g^\star$ remains unchanged.

4.5 Genetic Operators

We utilize order crossover [54] and one-point swap mutation to drive the evolution of robust composite services. Fig. 4 illustrates an example of crossover and mutation for the selected parent solutions. Particularly, in a crossover, two children are produced from two parents, and each child preserves a part of one parent while its remaining elements are filled by another parent. For example, Child 1 preserves positions 3 and 4 of Parent 1 while the other parts are filled from left to right with 1, 5, 2 and 6 that are obtained from Parent 2 from its left to right. In a mutation, one child is produced by swapping two elements of the parent. For example, Child 3 is produced by swapping 2 and 4 in Parent 1.

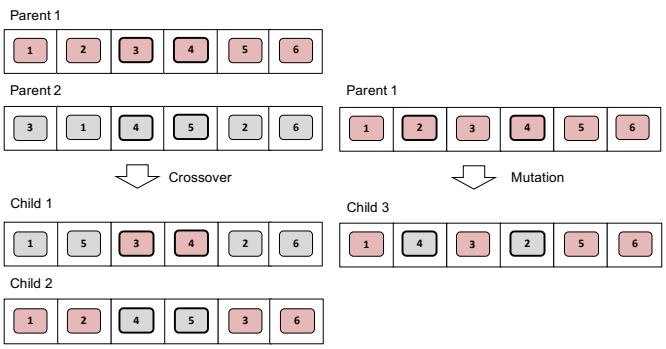

Fig. 4: Crossover and mutation

4.6 Fitness approximation based on selected scenarios

In ALGORITHM 3, we outline the calculation process of the robustness of each permutation Π in a population $\mathcal{P}^g$ using Eq. (8). We firstly produce $|\mathcal{SR}|$ scenarios, each of which only considers one service failure, and is different from each other. That is followed by identifying the best-repaired solution (i.e., a neighbouring solution associated with the highest comprehensive quality obtained through n_{nb} explored neighbours from Step 4 to Step 6). In step 7, we calculate $Pr(Q)$ as the weight of scenario Q. After an iteration over all the scenarios, comprehensive quality of all the best-repaired solutions over all the scenarios with different weights $Pr(Q)$ are summed up according to Eq. (8).

Example 4.2. Let us consider a composition task $T = (\{a, b\}, \{e, f\})$ and a service repository $\mathcal{SR}$ consisting of six atomic services. $S_0 = (\{e, f\}, \{g\}, QoS_{S_0})$, $S_1 = (\{b\}, \{c, d\}, QoS_{S_1})$, $S_2 = (\{c\}, \{e\}, QoS_{S_2})$, $S_3 = (\{d\}, \{f\}, QoS_{S_3})$, $S_4 = (\{a\}, \{h\}, QoS_{S_4})$ and $S_5 = (\{c\}, \{e, f\}, QoS_{S_5})$. The two special services $Start = (\emptyset, \{a, b\}, \emptyset)$ and $End = (\{e, f\}, \emptyset, \emptyset)$ are defined by a given composition task T. Fig. 5 illustrates an example of 6 selected scenarios that only consider one service failure at one time, and each service failure is different from each other. In this example, we use Scenario 6 to demonstrate

ALGORITHM 3. Simulation-based evaluation with local search (Step 11 and 14 in ALGORITHM 1).

Input : population $\mathcal{P}^g$, the number of neighbor n_{nb} and service repository $\mathcal{SR}$
Output: evaluated $\mathcal{P}^g$

1 **for each** Π *in* $\mathcal{P}^g$ **do**
2 Sample $|\mathcal{SR}|$ scenarios based on the number of S in $\mathcal{SR}$;
3 **foreach** *scenario* Q *in the* $|\mathcal{SR}|$ *sampled scenarios* **do**
4 Produce another permutation $\Pi^\star$ that encodes Π based on Q;
5 Generate a size n_{nb} of neighbors from $\Pi^\star$ by local search operator;
6 Identify the best neighbor Π' with the highest fitness based on Eq. (5);
7 Calculate $Pr(Q)$ as the weight for scenario Q;
8 Set the robustness of Π as a weighted-sum fitness value of $|\mathcal{SR}|$ Π' based on Eq. (8);
9 **return** evaluated $\mathcal{P}^g$;

how a new permutation (i.e., $\Pi^\star$) is produced as a starting solution point of our local search.

Based on the size of the given service repository $\mathcal{SR}$, we can produce 6 scenarios. Let $\{S_1, S_2, S_3, S_4, S_5\}$ be a produced scenario with S_0 becoming inaccessible. Therefore, $\{1, 2, 3, 4, 5\}$ is a set of service indexes corresponding to one sampled scenario in Fig. 5. In a similar way, $\{0, 1, 2, 3, 4\}$ is a set of service indexes corresponding to one sampled scenario representing the failure of S_5. To demonstrate the repairing process, we also take an arbitrary permutation $\Pi = [4, 1, 0, 2, 3, 5]$ as an example solution to the service composition problem. To encode scenarios for a given permutation Π, we produce another permutation $\Pi^\star$ for each scenario by only removing the service indexes of failed services from the permutation, keeping the order of other elements in the permutation. By doing this, the newly produced permutation $\Pi^\star$ can keep some promising component services from the candidate Π. For example, two permutations with two decoded DAGs are created for encoding two scenarios Q_1 and Q_6 respectively. In the two decoded DAGs, we can see that the promising service index 1 of the decoded $\mathcal{G}^\star$ from $\Pi^\star$ is inherited from that of the $\mathcal{G}$ decoded from Π in Fig. 5. On the other hand, the decoded DAG $\mathcal{G}^\star$ in Q_1 remains the same as the original $\mathcal{G}$ as the failure of S_0 has no impact on the execution of the original $\mathcal{G}$. In such a case, a repairing process is not involved. On the other hand, the decoded DAG $\mathcal{G}^\star$ in Q_1 is not identical to the original $\mathcal{G}$, as the failure of S_5 prevents the successful execution of the original $\mathcal{G}$. Therefore, a repairing process will be involved by preparing a starting point for local search — a tidy-up permutation.

The tidy-up permutation is crucial to the repairing process and used to set a good starting point for the local search. In Fig. 5, we tidy up permutation $[4, 1, 0, 2, 3]$ into $[1, 2, 3, |4, 0]$ ($|$ is just displayed for the courtesy of the reader, but not part of the representation) as an input of local search. We produce this permutation by combining two parts, one part $[1, 2, 3]$ is service indexes of component service in $\mathcal{G}^\star$, sorted based on the longest distance from $Start$ to every component services of $\mathcal{G}^\star$ while the second part $[4, 0]$ is indexes of remaining services that are available but not utilised by $\mathcal{G}^\star$.

Let $\Pi^\star = (\pi_0, \ldots, \pi_t, |\pi_{t+1}, \ldots, \pi_{n-1})$ be the produced

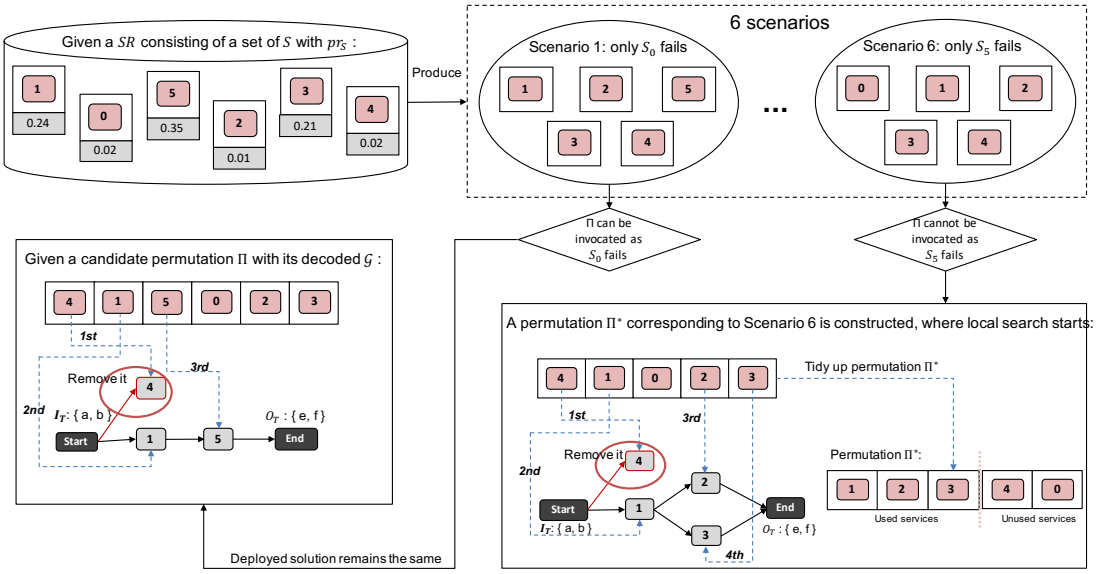

Fig. 5: An example of 6 selected scenarios involved in the lower bound robustness estimation and a new permutation $\Pi^\star$ produced from Scenario 6 as a starting solution point for local search

permutation in Step 4, elements of the permutation are $\{0, \ldots, t, t+1, \ldots, n-1\}$ such that $\pi_i \neq \pi_j$ for all $i \neq j$. Particularly, $\{0, \ldots, t\}$ are service indexes (i.e., id number) of the component services in the corresponding $\mathcal{G}$, and is sorted based on the longest distance from $Start$ to every component services of $\mathcal{G}$, while $\{t+1, \ldots, n-1\}$ are indexes of remaining services in $\mathcal{SR}$ not utilised by $\mathcal{G}$. Subsequently, we apply a stochastic local search operator (layer-based constrained one-point swap, see details in [55]) to $\Pi^\star$. To perform this local search, the layer information (i.e., different layers include different web services as layer members) must be utilised. Generally speaking, layer information indicates the order of a service being considered for the inclusion into a DAG of a composite service, starting from the input of a composition task I_T. For example, the first layer L_1 includes services that can be immediately executed based on the input of the composition task I_T. The second layer L_2 contains those services that can be executed by using I_T and outputs produced by services in L_1. Other layers can be discovered in a similar way, see the layer discovery technique in [5, 55]. Therefore, a neighbouring permutation is produced by swapping two selected service indexes Π_a and Π_b in the permutation. Particularly, one service index Π_a, where $0 \leq a \leq t$, is selected, and one layer L_k, where L_k $s.t.$ $\Pi_a \in L_k$, is identified. Afterwards, another service index Π_b is randomly selected from the index set $L_k \cap \{\Pi_{t+1}, \ldots, \Pi_{n-1}\}$.

Example 4.3. Let us consider a layer-based constrained one-point swap, starting from the produced permutation $[1, 2, 3, |4, 0]$ in *Example* 4.2. Fig. 6 illustrates a process of producing a neighboring permutation from the given permutation.

For the permutation $[1, 2, 3, |4, 0]$, one service index (e.g., 1) is firstly randomly selected before the $|$ in the permutation (i.e., 1, 2 or 3). Then we get the layer information of service index 1 (e.g., layer L_1 contains 1). Afterwards, another service index (e.g., 4) is randomly selected from the intersection set of service indexes in L_1 and the service indexes after the $|$. Consequently, 1 and 4 are swapped to generate a new

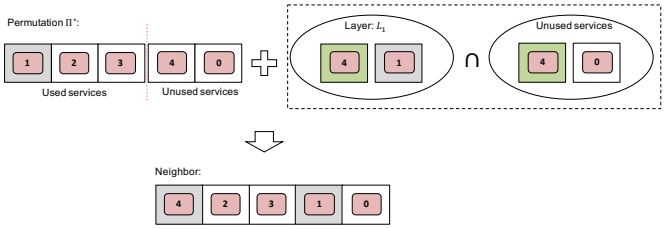

Fig. 6: A local search operator is performed on the permutation $\Pi^\star$ to produce a neighboring permutation, for the purpose of searching a high-quality solution under Scenario 6 in Fig. 5

permutation.

5 Experimental Evaluation

We conduct experiments to evaluate the performances of GA-2Stage, GA-RE, GA-MC and FL. GA-2Stage, GA-RE and GA-MC aim to generate composite services with near-optimal robustness using Eq. (8) and/or Eq. (7). On the other hand, FL only focus on generating composite services with near-optimal comprehensive quality using Eq. (5). We use three service composition benchmarks (i.e., OWL-S TC [19], WSC-08 [56] and WSC-09 [57]) to test the performance of the methods. OWL-S TC has five composition tasks (i.e., OWL-S TC1 to OWL-S TC5) with services that are collected from the real-world. One service repository with 946 web services is used by the five composition tasks. Unlike OWL-S TC, WSC-08 and WSC-09 includes eight and five composition tasks respectively with web services that are simulated for Web Service Challenges competition. Particularly, WSC08 contains 8 composition tasks with increasing size of service repository, i.e., 158, 558, 608, 1041, 1090, 2198, 4113, and 8119, and WSC09 contains 5 composition tasks with increasing size of service repository, i.e., 572, 4129, 8138, 8301, and 15211, respectively. Moreover, each service in OWL-S TC, WSC-08 and WSC-09 is extended with real-world QoS attributes obtained from QWS dataset [58]. Apart from that, each service is also associated with a separate service failure

TABLE 3: Mean fitness values tested based on the baseline solutions for GA-2Stage in comparison to GA-RE, FL and GA-MC (Note: the higher the fitness the better)

Task	GA-2Stage	GA-RE	FL	GA-MC
OWLS-TC 1	**0.922788 ± 0.000179**	**0.922862 ± 0.00018**	**0.922791 ± 0.000311**	0.910623 ± 0.033122
OWLS-TC 2	**0.931586 ± 0.002107**	**0.932095 ± 0.000277**	0.929618 ± 0.005009	0.915558 ± 0.025327
OWLS-TC 3	**0.863518 ± 0.003623**	**0.863061 ± 0.00725**	0.854218 ± 0.00779	**0.862459 ± 0.003403**
OWLS-TC 4	**0.789396 ± 0.00857**	**0.791174 ± 0.004451**	0.779121 ± 0.012348	**0.789101 ± 0.005596**
OWLS-TC 5	**0.826731 ± 0.005054**	**0.826509 ± 0.005275**	0.812852 ± 0.012388	**0.824296 ± 0.013078**
WSC08-1	**0.397971 ± 0.002975**	**0.398896 ± 0.003538**	0.395194 ± 0.002782	0.383523 ± 0.005433
WSC08-2	**0.576417 ± 0.00378**	**0.576716 ± 0.00289**	0.568166 ± 0.005362	0.572707 ± 0.006905
WSC08-3	**0.025494 ± 0.00295**	**0.02501 ± 0.001415**	**0.025118 ± 0.001333**	**0.02509 ± 0.001424**
WSC08-4	**0.274788 ± 0.00295**	**0.275631 ± 0.002872**	0.271112 ± 0.004133	0.268906 ± 0.003492
WSC08-5	**0.275513 ± 0.002107**	**0.275814 ± 0.003147**	**0.275259 ± 0.002625**	0.272394 ± 0.003412
WSC08-6	**0.072178 ± 0.002047**	**0.072036 ± 0.001605**	**0.072017 ± 0.002114**	**0.071496 ± 0.002028**
WSC08-7	**0.216909 ± 0.003111**	**0.217957 ± 0.00322**	**0.216321 ± 0.003162**	0.214177 ± 0.00333
WSC08-8	**0.053596 ± 0.002047**	**0.054259 ± 0.001909**	**0.0536 ± 0.002007**	**0.053406 ± 0.001992**
WSC09-1	**0.514032 ± 0.005805**	**0.514389 ± 0.006263**	0.501957 ± 0.005435	0.509053 ± 0.007493
WSC09-2	**0.272445 ± 0.00309**	**0.272217 ± 0.002653**	**0.272905 ± 0.002886**	0.269513 ± 0.003516
WSC09-3	**0.387672 ± 0.003014**	**0.386712 ± 0.002356**	0.384879 ± 0.002562	0.378036 ± 0.005318
WSC09-4	**0.068139 ± 0.002223**	**0.067328 ± 0.001968**	**0.067515 ± 0.002214**	-
WSC09-5	**0.108845 ± 0.002602**	**0.109347 ± 0.002619**	**0.107809 ± 0.002011**	**0.108056 ± 0.002067**

rate. The failure rate of a service is generated from the normal distribution $\mathcal{N}(\mu, \sigma^2)$ truncated to the interval $[0, 1]$ with mean μ and variance σ^2. According to the failure rates reported in [10] and by using 15 000 failure probabilities observed by 150 users on 100 web services collected from publicly available real-world WSDL-based web services on the Internet, μ and σ are set to 0.0405 and 0.1732. The benchmark data sets and the source code of GA-2Stage algorithm have been made publicly available online under MIT License [1]

To perform fair comparisons among GA-2Stage, GA-RE, GA-MC and FL, we follow the parameter setting reported in the literature [5]: population size m is set to 30, tournament size is set to 2 and elitism is set to 2. The crossover and mutation rates are inspired by Koza's operator settings [59] and are set to 0.95 and 0.05, respectively. In Eq. (8), for each solution, the number of scenarios equals the size of the service repository, and the number of scenarios that trigger local search equals to the number of component service utilized by the solution. Therefore, to ensure a fair comparison with GA-MC, the number of scenarios N in Eq. (7) for GA-MC does not follow the reported size 50 in [17]. Instead, we ensure a doubled number of scenarios that employ local search in GA-MC for a more reliable estimation of robustness. To do that, we keep sampling scenarios randomly until twice the number of scenarios that have triggered the local search. Moreover, according to [17], we set the maximum generation to 100, and the number of local search steps (i.e., n_{nb}) to 10 for empirically producing a good compromise between computation cost and service quality. For the archive-based evolutionary control, the initial generation number $g^\star$ and increased generation number g_{inc} are set to 60 and 6 respectively. For Eq. (5), all weights are set to balance quality criteria in both QoSM and QoS, i.e., w_1 and w_2 are set to 0.25, and w_3, w_4, w_5 and w_6 to 0.125 [8] according to the settings in [8]. We have also conducted tests with other weight settings and generally observed the same behaviour.

We run GA-2Stage, GA-RE, GA-MC and FL 30 times with 30 different random seeds. We then test each baseline composite service obtained by every run of every algorithm over 200 different simulated scenarios according to [17].

1. Benchmark data sets, and source code of GA-2Stage are available from https://github.com/chenwangnida/GAPlusGARobustSurrogate

TABLE 4: Summary of statistical significance tests for mean fitness values, where each column shows the win/draw/loss score of one method against a competing one for all tasks of OWLS-TC, WSC08 and WSC09.

Dataset	Method	GA-2Stage	GA-RE	FL	GA-MC
OWLS-TC (5 tasks)	GA-2Stage	-	0/5/0	0/1/4	0/3/2
	GA-RE	0/5/0	-	0/1/4	0/3/2
	FL	4/1/0	4/1/0	-	2/3/0
	GA-MC	2/3/0	2/3/0	0/3/2	-
WSC08 (8 tasks)	GA-2Stage	-	0/8/0	0/5/3	0/3/5
	GA-RE	0/8/0	-	0/5/3	0/3/5
	FL	3/5/0	3/5/0	-	0/3/5
	GA-MC	5/3/0	5/3/0	5/3/0	-
WSC09 (5 tasks)	GA-2Stage	-	0/5/0	0/3/2	0/1/4
	GA-RE	0/5/0	-	0/2/3	0/1/4
	FL	2/3/0	3/2/0	-	1/1/3
	GA-MC	4/1/0	4/1/0	3/1/1	-

For example, let us consider a service repository of 6 web services $\{S_0, S_1, S_2, S_3, S_4, S_5\}$ that is discussed in **Example** 4.2. Every single web service may experience service failure, so there are 2^6 different failure scenarios that could be tested for the execution phase. In other words, when testing we also consider those scenarios where multiple web services fail at the execution phase. Note that, scenarios considered at the design phase are different from those 200 simulated scenarios sampled for testing, such differences are important because we want to accurately measure the robustness of any composite service during the execution phase. Subsequently, we use two-sample t-test with a significance level of 5% to verify the observed difference in the mean fitness values and the execution times tested on the baselines found by GA-2Stage, GA-RE, GA-MC and FL. Lastly, we further study the benefits of the robust estimation models, i.e., the lower bound robust estimation in Eq. 8 and the Monte Carlo estimation in Eq. 7, and demonstrate their accuracy in ranking candidate solutions using Kendall's Tau test with a significance level of 5%.

5.1 Comparison of the effectiveness

To study the effectiveness of GA-2Stage, GA-RE, FL and GA-MC in finding robust baseline solutions, Table. 3 shows the mean fitness values and standard deviations obtained from testing on baseline solutions for GA2-Stage, GA-RE, FL and GA-MC over 30 runs, and each run is tested over 200 random scenarios of service failures at the execution phase. We verify the significant differences in the fitness values

using two-sample t test, and the winner is highlighted in the table. In particular, we use pairwise comparisons among GA-2Stage, GA-RE, GA-MC and FL using independent-sample T-test with a significance level of 5% to verify the observed differences in performance concerning fitness values. Afterwards, the top performances are identified, and its related value is highlighted in green color in Table 3. The pairwise comparison results for fitness are summarized in Table 4, where *win/draw/loss* shows the scores of one method compared to all the others, and displays the frequency that this method outperforms, equals or is outperformed by the competing method. This testing and comparison methods are also used in Sect 5.2. Note that all the P-values are lower than 0.001, and any "$-$" in the tables means results cannot be collected since the related testing instances has been running for more than 16 days for the design phase.

Compared to FL and GA-MC, GA-2Stage and GA-RE outperform these two methods as evidenced by the performance, summarised in Table 4. Particularly, baseline solutions produced by GA-2Stage and GA-RE both achieve consistently good performance for all the tasks in OWLS-TC, WSC-08, and WSC-09 as top performers regardless of the size of the service repository. Therefore, composite services produced by GA-2Stage and GA-RE are more likely to maintain a good quality despite stochastic service failures. This finding matches well with our expectation that our fitness approximation in GA-2Stage and GA-RE is very effective in dealing with service composition tasks with both small and large service repositories. Moreover, the effectiveness of GA-2Stage and GA-RE are very comparable to each other. This observation agrees with our expectation that GA-2Stage can maintain the effectiveness of GA-RE in finding robust composite services by our cheap two-stage evaluations, without performing fitness approximation throughout all the generations.

Moreover, for the two baseline methods FL and GA-MC, GA-MC outperforms FL in OWLS-TC benchmark while the same performance cannot be observed from WSC08 and WSC-09 benchmarks. This is because the robustness measure using Eq. (7) in GA-MC presents low variances for the OWLS-TC benchmarks with small service repository size. This robustness measure can be reliable to distinguish good or bad solutions during the evolutionary process, compared to that in the WSC-08 and WSC-09 benchmarks with large service repository size. In other words, for large benchmark datasets, such as WSC-08 and WSC-09, the robustness measure in GA-MC presents high variances and cannot be reliably used as fitness values in any EC technique.

TABLE 5: Mean execution time (in s) observed for GA-2Stage in comparison to GA-RE, FL and GA-MC at the design phase (Note: the shorter the time the better)

Task	GA-2Stage	GA-RE	FL	GA-MC
OWLS-TC 1	101.009067 ± 23.07637	221.854233 ± 63.968435	**2.279767 ± 0.594116**	319.139533 ± 76.385136
OWLS-TC 2	35.9972 ± 31.302637	51.851 ± 34.814491	**1.502733 ± 0.163235**	115.441333 ± 71.247137
OWLS-TC 3	13.0314 ± 4.849419	27.075967 ± 14.63108	**1.4005 ± 0.132212**	55.393333 ± 24.899348
OWLS-TC 4	173.224267 ± 116.190079	468.054967 ± 342.97007	**13.785767 ± 21.966587**	786.6216 ± 365.605208
OWLS-TC 5	336.6059 ± 206.179729	901.813933 ± 598.884817	**19.577733 ± 71.642104**	1263.4348 ± 807.615199
WSC08-1	362.312133 ± 77.091671	1153.114633 ± 157.16768	**14.593633 ± 7.036069**	2401.1046 ± 290.709963
WSC08-2	136.833233 ± 32.808761	346.3632 ± 88.461812	**9.5453 ± 5.685242**	630.930267 ± 98.000929
WSC08-3	5445.803933 ± 2356.34388	20646.882333 ± 2831.701793	**297.080867 ± 50.772516**	8243.711267 ± 827.243178
WSC08-4	363.368233 ± 46.265638	946.272833 ± 116.302594	**22.1297 ± 5.09158**	1223.7892 ± 119.670871
WSC08-5	5411.626833 ± 1754.906719	21636.348867 ± 2504.446328	**287.389433 ± 69.087335**	33964.1487 ± 5147.121731
WSC08-6	58857.504067 ± 42330.514059	309276.646467 ± 36848.385724	**3056.054767 ± 744.411865**	304471.121333 ± 37767.070623
WSC08-7	8399.0462 ± 2735.343313	26800.4088 ± 4888.82274	**477.589233 ± 166.107954**	36787.8031 ± 5155.913198
WSC08-8	10500.235667 ± 4032.270104	39111.0008 ± 5217.311172	**572.005833 ± 96.59887**	22512.887867 ± 2243.886478
WSC09-1	327.1701 ± 80.081116	943.368733 ± 269.944223	**15.832667 ± 12.600583**	1633.791 ± 284.342794
WSC09-2	11490.332567 ± 3504.752592	40003.2077 ± 5467.541146	**499.271067 ± 183.33795**	61763.3132 ± 7293.294917
WSC09-3	6117.213533 ± 1423.425485	17718.882467 ± 3384.984318	**401.276233 ± 238.865704**	38380.1644 ± 7324.715594
WSC09-4	223757.395885 ± 192691.648686	1420123.714684 ± 292166.614721	**11110.008733 ± 2371.111691**	-
WSC09-5	50052.6394 ± 17629.244248	174415.461533 ± 34932.061346	**1972.0222 ± 713.868674**	160876.807733 ± 16575.127178

TABLE 6: Mean execution time (in ms) per scenario by local search based on the baseline solutions found by GA-2Stage in comparison to GA-RE, FL and GA-MC (Note: the shorter the time the better)

Task	GA-2Stage	GA-RE	FL	GA-MC
OWLS-TC 1	0.185822 ± 0.075933	0.155067 ± 0.06195	0.194944 ± 0.095481	2.339717 ± 6.703382
OWLS-TC 2	0.95495 ± 1.129079	0.456811 ± 0.323291	1.173133 ± 1.618681	4.135078 ± 7.997827
OWLS-TC 3	1.160528 ± 0.836618	0.788439 ± 0.574859	1.363739 ± 0.892455	0.926944 ± 0.666366
OWLS-TC 4	8.860739 ± 5.273149	9.315556 ± 7.508798	10.824494 ± 5.943972	10.265489 ± 6.760639
OWLS-TC 5	15.748111 ± 12.911706	12.694856 ± 10.350321	22.812806 ± 21.672252	16.579461 ± 22.341204
WSC08-1	21.327167 ± 2.756077	21.454789 ± 2.873897	20.090467 ± 2.6058	25.159133 ± 3.281322
WSC08-2	12.451461 ± 3.029406	12.090028 ± 3.017539	12.761439 ± 2.55482	11.707733 ± 3.262212
WSC08-3	32.899689 ± 3.254111	33.112011 ± 4.172865	32.191772 ± 3.75622	33.462789 ± 4.481216
WSC08-4	16.180183 ± 2.256799	14.979589 ± 1.932228	15.879867 ± 2.083646	16.741572 ± 2.032044
WSC08-5	179.797372 ± 17.259394	175.223117 ± 20.767383	184.219939 ± 17.570621	184.892367 ± 24.642377
WSC08-6	911.157344 ± 121.979441	891.088928 ± 75.58605	929.152878 ± 85.605171	907.872372 ± 84.003491
WSC08-7	214.58365 ± 25.636885	207.045233 ± 21.754379	205.707067 ± 23.674507	221.702394 ± 32.026932
WSC08-8	88.189956 ± 10.84816	88.570883 ± 7.407168	88.2868 ± 8.049778	91.657628 ± 8.401898
WSC09-1	25.138389 ± 6.636436	25.804 ± 6.851249	24.943722 ± 5.962107	29.2981 ± 7.077208
WSC09-2	297.721911 ± 33.783551	315.269294 ± 31.808168	297.211728 ± 21.107784	312.352106 ± 38.903679
WSC09-3	302.659394 ± 57.691578	306.980983 ± 35.572703	297.8739 ± 42.378935	315.209356 ± 48.449898
WSC09-4	3411.716628 ± 447.572982	3560.59905 ± 449.65906	3528.859594 ± 620.60136	-
WSC09-5	603.66355 ± 117.838659	606.873822 ± 67.904197	581.632528 ± 64.545121	608.908839 ± 50.244834

5.2 Comparison of the efficiency

To study the efficiency of GA-2Stage, GA-RE, FL and GA-MC at both design phase and execution phase, Tables 5 and 6 show execution times observed for design phase and execution phase, respectively, over 30 runs. We keep employing pairwise comparisons with two-sample T test to detect any noticeable differences in the experiment results in the efficiency, see Tables 7 and 8.

TABLE 7: Summary of statistical significance tests for mean execution time, where each column shows the win/draw/loss score of one method against a competing one for all tasks of OWLS-TC, WSC08 and WSC09.

Dataset	Method	GA-2Stage	GA-RE	FL	GA-MC
OWLS-TC (5 tasks)	GA-2Stage	-	0/0/5	5/0/0	0/0/5
	GA-RE	5/0/0	-	5/0/0	0/0/5
	FL	0/0/5	0/0/5	-	0/0/5
	GA-MC	5/0/0	5/0/0	5/0/0	-
WSC08 (8 tasks)	GA-2Stage	-	0/0/8	8/0/0	0/0/8
	GA-RE	8/0/0	-	8/0/0	0/0/8
	FL	0/0/8	0/0/8	-	0/0/8
	GA-MC	8/0/0	8/0/0	8/0/0	-
WSC09 (5 tasks)	GA-2Stage	-		5/0/0	0/0/5
	GA-RE	5/0/0	-	5/0/0	0/0/5
	FL	0/0/5	0/0/5	-	0/0/5
	GA-MC	5/0/0	5/0/0	5/0/0	-

TABLE 8: Summary of statistical significance tests for mean execution time per scenario, where each column shows the win/draw/loss score of one method against a competing one for all tasks of OWLS-TC, WSC08 and WSC09.

Dataset	Method	GA-2Stage	GA-RE	FL	GA-MC
OWLS-TC (5 tasks)	GA-2Stage	-	0/5/0	0/5/0	0/5/0
	GA-RE	0/5/0	-	0/5/0	0/5/0
	FL	0/5/0	0/5/0	-	0/5/0
	GA-MC	0/5/0	0/5/0	0/5/0	-
WSC08 (8 tasks)	GA-2Stage	-	0/8/0	0/8/0	0/8/0
	GA-RE	0/8/0	-	0/8/0	0/8/0
	FL	0/8/0	0/8/0	-	0/8/0
	GA-MC	0/8/0	0/8/0	0/8/0	-
WSC09 (5 tasks)	GA-2Stage	-	0/5/0	0/5/0	0/5/0
	GA-RE	0/5/0	-	0/5/0	0/5/0
	FL	0/5/0	0/5/0	-	0/5/0
	GA-MC	0/5/0	0/5/0	0/5/0	-

At the design phase, see Tables 5 and Tables 7, we can observe that FL consistently takes significantly less execution time (in seconds) comparing to all the other methods. This is because the fitness evaluation in FL through Eq. (5) is far more efficient than GA-Stage, GA-RE and GA-MC. On the other hand, GA-MC consistently requires the most execution time in the design phase, while GA-RE requires the second most execution time in the design phase. This is because a single evaluation of one candidate solution involves N calculations of comprehensive quality using Eq. (7). This N is much larger than the number of scenarios in GA-RE using Eq. (8). As we discussed in Sect. 5.1, GA-MC become less reliable when it is tested on a large benchmark. Although we increase N in Eq. (7) to allow more accurate robustness approximation, GA-MC does not outperform GA-2Stage for finding high-robustness solutions.

GA-RE requires less significant execution time at the design phase, compared to GA-MC. This is because the efficiency of evolving robust composite services is further improved with the help of fitness approximation used in GA-RE. On the other hand, although GA-RE consumes much longer execution time at the design phase, GA-RE gains much higher quality against the stochastic service failures at the execution phase, see the previous discussion in Sect. 5.1.

In addition, GA-2Stage further improves the efficiency of GA-RE by introducing a two-stage optimisation process with adaptive evolutionary control. This is because the majority of the generations in GA-2Stage employ a single evaluation via comprehensive quality using Eq. (5), while the rest of the generations employ the fitness approximation using Eq. (8). Meanwhile, GA-2Stage can still maintain high effectiveness in finding high-robustness baseline solutions.

At the execution phase, see Table 6, no results are highlighted in the tables since we do not observe any significant difference among any two competing methods. This indicates that baseline solutions produced by all the methods for every task need a similar amount of time to be repaired in the event of service failures. This amount of repair time is independent of the design phase. Moreover, this amount of repair time is far less than the execution required at the training stage, and the frequency of producing baseline solutions by all the methods is also far less frequent than that of repairing the baseline solutions by local search.

5.3 Comparison of the accuracy in robustness estimation

To study the accuracy of the lower bound robust estimation (called r_{LB}) in Eq. (8) and the Monte Carlo estimation (called r_{MC}) in Eq. (7), we compare each of them with the testing results, which serve as the "ground truth" of the robustness of any composite service in this study. Particularly, we are interested in how accurate different estimation methods are capable of ranking multiple composite services. For this purpose, we firstly record the fitness values of 30 composite services measured by the lower bound robust estimation, Monte Carlo estimation, and the "ground truth", for WSC09-1. Afterwards, we calculate the rank correlation between the r_{LB} and the "ground truth", and between the r_{MC} and the "ground truth", using Pearson, Kendall's tau and Spearman's rho.

Table 9 shows the correlation coefficient values and P-values of Pearson, Kendall's tau and Spearman's rho over two pairs of ranks, i.e., r_{LB} and the "ground truth", and r_{MC} and the "ground truth". We can see that both two correlation tests reject the null hypothesis that the two ranks are uncorrelated because all the P-values are less than 0.05. In addition, we can observe that the correlation coefficient between r_{LB} and the "ground truth" is consistently higher than that between r_{MC} and the "ground truth". This indicates that lower bound robust estimation is more accurate than the Monte Carlo robust estimation.

TABLE 9: Results of three statistical correlation tests using Pearson, Kendall's tau, and Spearman's rho.

Method		r_{LB}	r_{MC}
Pearson	Correlation coefficient	**0.611078**	0.426510
	P-value	0.000334	0.018756
Kendall's tau	Correlation coefficient	**0.452137**	0.310345
	P-value	0.000468	0.016017
Spearman's rho	Correlation coefficient	**0.615333**	0.493215
	P-value	0.000296	0.005615

6 Conclusions

In this paper, we proposed a fitness approximation method for measuring the robustness of composite services, and two-stage GA with fitness estimation, namely GA-2Stage, for handling stochastic service failures. In particular, we proposed a robustness measure using a lower bound of the expected fitness function. GA-2Stage is proposed with

an archive-based adaptive evolutionary control over two sequential stages to efficiently and effectively produce baseline solutions with high robustness regardless of the size of the service repository. Such baseline solutions can handle situations that some of the component services are not available at the execution phase via an efficient local search to find feasible and high-quality solutions at the execution phase. Our experimental evaluation shows that GA-2Stage can efficiently produce high-robustness baseline solutions consistently compared to a state-of-the-art FL method that merely focuses on searching high-quality solutions and a recently proposed GA method based on Monte Carlo sampling. Besides that, with the help of our proposed evolutionary control, GA-2Stage can also achieve much higher efficiency at the design phase with negligible impact on finding high-robustness solutions. In the future, we can study the applications of the proposed robustness estimation and the adaptive evolutionary control mechanism in other evolutionary algorithms.

References

[1] F. Curbera, W. Nagy, and S. Weerawarana, "Web services: Why and how," in *Workshop on Object-Oriented Web Services-OOPSLA*, vol. 2001, 2001.

[2] J. Rao and X. Su, "A survey of automated web service composition methods," in *International Workshop on Semantic Web Services and Web Process Composition*. Springer, 2004, pp. 43–54.

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

[4] P. Rodriguez-Mier, M. Mucientes, M. Lama, and M. I. Couto, "Composition of web services through genetic programming," *Evolutionary Intelligence*, vol. 3, no. 3-4, pp. 171–186, 2010.

[5] A. S. da Silva, Y. Mei, H. Ma, and M. Zhang, "Evolutionary computation for automatic web service composition: an indirect representation approach," *J. Heuristics*, pp. 425–456, 2018.

[6] S. Chattopadhyay, A. Banerjee, and N. Banerjee, "A fast and scalable mechanism for web service composition," *ACM Transactions on the Web (TWEB)*, vol. 11, no. 4, p. 26, 2017.

[7] C. Wang, H. Ma, G. Chen, and S. Hartmann, "GP-based approach to comprehensive quality-aware automated semantic web service composition," in *Asia-Pacific Conf. on Simulated Evolution and Learning*. Springer, 2017, pp. 170–183.

[8] ——, "Knowledge-driven automated web service composition — an EDA-based approach," in *Int. Conf. on Web Information Systems Engineering*. Springer, 2018, pp. 135–150.

[9] H. Yin, C. Zhang, B. Zhang, Y. Guo, and T. Liu, "A hybrid multiobjective discrete particle swarm optimization algorithm for a SLA-aware service composition problem," *Mathematical Problems in Engineering*, 2014.

[10] Z. Zheng, Y. Zhang, and M. R. Lyu, "Investigating QoS of real-world web services," *IEEE Trans. Services Computing*, vol. 7, no. 1, pp. 32–39, 2014.

[11] D. Kuropka, P. Tröger, S. Staab, and M. Weske, *Semantic service provisioning*. Springer, 2008.

[12] A. M. Daniel and T. Menasc, "QoS issues in web services," *IEEE Internet Computing*, vol. 6, no. 6, pp. 72–75, 2002.

[13] F. Boudries, S. Sadouki, and A. Tari, "A bio-inspired algorithm for dynamic reconfiguration with end-to-end constraints in web services composition," *Service Oriented Computing and Applications*, vol. 13, no. 3, pp. 251–260, 2019.

[14] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Software Engineering*, vol. 33, no. 6, pp. 369–384, 2007.

[15] A. Mostafa and M. Zhang, "Multi-objective service composition in uncertain environments," *IEEE Trans. Services Computing*, 2015.

[16] L. Wang, J. Shen, and J. Luo, "Impacts of pheromone modification strategies in ant colony for data-intensive service provision," in *IEEE Int. Conf. on Web Services*. IEEE, 2014, pp. 177–184.

[17] C. Wang, H. Ma, A. Chen, and S. Hartmann, "Towards robust web service composition with stochastic service failures based on a genetic algorithm," in *Australasian Joint Conf. on Artificial Intelligence*. Springer, 2019, pp. 445–459.

[18] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo method*. Wiley, 2016, vol. 10.

[19] U. Küster, B. König-Ries, and A. Krug, "Opossum-an online portal to collect and share SWS descriptions," in *Semantic Computing, 2008 IEEE International Conference on*. IEEE, 2008, pp. 480–481.

[20] W. Du, W. Zhong, Y. Tang, W. Du, and Y. Jin, "High-dimensional robust multi-objective optimization for order scheduling: A decision variable classification approach," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 293–304, 2018.

[21] Y. Jin, J. Branke *et al.*, "Evolutionary optimization in uncertain environments-a survey," *IEEE Transactions on evolutionary computation*, vol. 9, no. 3, pp. 303–317, 2005.

[22] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[23] C. Wang, H. Ma, A. Chen, and S. Hartmann, "Comprehensive quality-aware automated semantic web service composition," in *Australasian Joint Conf. on Artificial Intell.* Springer, 2017, pp. 195–207.

[24] N. Arunachalam and A. Amuthan, "Integrated probability multi-search and solution acceptance rule-based artificial bee colony optimization scheme for web service composition," *Natural Computing*, pp. 1–16, 2019.

[25] A. S. da Silva, H. Ma, and M. Zhang, "Graphevol: a graph evolution technique for web service composition," in *Database and Expert Systems Applications*. Springer, 2015, pp. 134–142.

[26] S. Chattopadhyay and A. Banerjee, "Qos-aware automatic web service composition with multiple objectives," *ACM Transactions on the Web (TWEB)*, vol. 14, no. 3, pp. 1–38, 2020.

[27] A. S. da Silva, H. Ma, and M. Zhang, "Genetic programming for QoS-aware web service composition and selection," *Soft Computing*, pp. 1–17, 2016.

[28] H. Ma, A. Wang, and M. Zhang, "A hybrid approach using genetic programming and greedy search for QoS-aware web service composition," *Trans. Large-Scale Data Knowledge-Centered Syst.*, vol. 18, pp. 180–205, 2015.

[29] Y. Yu, H. Ma, and M. Zhang, "An adaptive genetic programming approach to QoS-aware web services composition," in *2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 1740–1747.

[30] X. Jian, Q. Zhu, and Y. Xia, "An interval-based fuzzy ranking approach for QoS uncertainty-aware service composition," *Optik-International Journal for Light and Electron Optics*, vol. 127, no. 4, pp. 2102–2110, 2016.

[31] S.-Y. Hwang, C.-C. Hsu, and C.-H. Lee, "Service selection for web services with probabilistic QoS," *IEEE transactions on services computing*, vol. 8, no. 3, pp. 467–480, 2014.

[32] A. Amin, A. Colman, and L. Grunske, "An approach to forecasting QoS attributes of web services based on arima and garch models," in *IEEE Int. Conf. on Web Services*. IEEE, 2012, pp. 74–81.

[33] M. Li, Z. Hua, J. Zhao, Y. Zou, and B. Xie, "Arima model-based web services trustworthiness evaluation and prediction," in *Int. Conf. on Service-Oriented Computing*. Springer, 2012, pp. 648–655.

[34] S. Chattopadhyay and A. Banerjee, "QSCAS: QoS aware web service composition algorithms with stochastic parameters," in *2016 IEEE International Conference on Web Services (ICWS)*. IEEE, 2016, pp. 388–395.

[35] X. Sun, J. Chen, Y. Xia, Q. He, Y. Wang, X. Luo, R. Zhang, W. Han, and Q. Wu, "A fluctuation-aware approach for predictive web service composition," in *2018 IEEE International Conference on Services Computing (SCC)*. IEEE, 2018, pp. 121–128.

[36] X. Sun, S. Wang, Y. Xia, and W. Zheng, "Predictive-trend-aware composition of web services with time-varying quality-of-service," *IEEE Access*, vol. 8, pp. 1910–1921, 2020.

[37] S. C. Geyik, B. K. Szymanski, and P. Zerfos, "Robust dynamic service composition in sensor networks," *IEEE Trans. Services Computing*, pp. 560–572, 2013.

[38] F. Wagner, F. Ishikawa, and S. Honiden, "Robust service compositions with functional and location diversity," *IEEE Trans. Services Computing*, vol. 9, no. 2, pp. 277–290, 2016.

[39] K.-J. Lin, J. Zhang, Y. Zhai, and B. Xu, "The design and implementation of service process reconfiguration with end-to-end qos constraints in soa," *Service Oriented Computing and Applications*, vol. 4, no. 3, pp. 157–168, 2010.

[40] K. Rajaram, C. Babu, and A. Adiththan, "Dynamic transaction aware web service selection," *International Journal of Cooperative Information Systems*, vol. 23, no. 03, p. 1450004, 2014.

[41] ——, "Tx-faith: A transactional framework for failure tolerant execution of hierarchical long-running transactions in business applications," *International Journal of Web Services Research (IJWSR)*, vol. 11, no. 3, pp. 1–26, 2014.

[42] I. Paenke, J. Branke, and Y. Jin, "Efficient search for robust solutions by means of evolutionary algorithms and fitness approxima-

tion," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 4, pp. 405–420, 2006.

[43] L. Gräning, Y. Jin, and B. Sendhoff, "Individual-based management of meta-models for evolutionary optimization with application to three-dimensional blade optimization," in *Evolutionary computation in dynamic and uncertain environments*. Springer, 2007, pp. 225–250.

[44] Y. Jin, M. Hüsken, and B. Sendhoff, "Quality measures for approximate models in evolutionary computation," in *GECCO*, 2003, pp. 170–173.

[45] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality driven web services composition," in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 411–421.

[46] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *International Semantic Web Conference*. Springer, 2002, pp. 333–347.

[47] F. Lécué, "Optimizing QoS-aware semantic web service composition," in *International Semantic Web Conference*. Springer, 2009, pp. 375–391.

[48] F. Lécué, A. Delteil, and A. Léger, "Optimizing causal link based web service composition." in *ECAI*, 2008, pp. 45–49.

[49] K. Shet, U. D. Acharya *et al.*, "A new similarity measure for taxonomy based on edge counting," *arXiv preprint arXiv:1211.4709*, 2012.

[50] H. Ma, K.-D. Schewe, B. Thalheim, and Q. Wang, "A formal model for the interoperability of service clouds," *Service Oriented Computing and Applications*, vol. 6, no. 3, pp. 189–205, 2012.

[51] A. S. da Silva, Y. Mei, H. Ma, and M. Zhang, "Particle swarm optimisation with sequence-like indirect representation for web service composition," in *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer, 2016, pp. 202–218.

[52] C.-L. Hwang and K. Yoon, "Lecture notes in economics and mathematical systems," *Multiple Objective Decision Making, Methods and Applications: A State-of-the-Art Survey*, 1981.

[53] S. C. Leung* and Y. Wu, "A robust optimization model for stochastic aggregate production planning," *Production planning & control*, vol. 15, no. 5, pp. 502–514, 2004.

[54] L. Davis, "Applying adaptive algorithms to epistatic domains." in *IJCAI*, vol. 85, 1985, pp. 162–164.

[55] C. Wang, H. Ma, G. Chen, and S. Hartmann, "Memetic EDA-based approaches to comprehensive quality-aware automated semantic web service composition," *arXiv preprint arXiv:1906.07900*, 2019.

[56] A. Bansal, M. B. Blake, S. Kona, S. Bleul, T. Weise, and M. C. Jaeger, "WSC-08: continuing the web services challenge," in *E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 2008 10th IEEE Conference on*. IEEE, 2008, pp. 351–354.

[57] S. Kona, A. Bansal, M. B. Blake, S. Bleul, and T. Weise, "WSC-2009: a quality of service-oriented web services challenge," in *2009 IEEE Conference on Commerce and Enterprise Computing*. IEEE, 2009, pp. 487–490.

[58] E. Al-Masri and Q. H. Mahmoud, "Qos-based discovery and ranking of web services," in *Int. Conf. on Computer Comm. Networks*. IEEE, 2007, pp. 529–534.

[59] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.