Manuscript version: Author’s Accepted Manuscript
The version presented in WRAP is the author’s accepted manuscript and may differ from the published version or Version of Record.

Persistent WRAP URL:
http://wrap.warwick.ac.uk/143427

How to cite:
Please refer to published version for the most recent bibliographic citation information. If a published version is known of, the repository item page linked to above, will contain details on accessing it.

Copyright and reuse:
The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions.

Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

Publisher’s statement:
Please refer to the repository item page, publisher’s statement section, for further information.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk.
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
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
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 run-time 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 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>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Composite service</td>
</tr>
<tr>
<td>type</td>
<td>matchmaking type of a robust causal link</td>
</tr>
<tr>
<td>MT</td>
<td>matchmaking type of a composite service</td>
</tr>
<tr>
<td>Sim</td>
<td>semantic similarity</td>
</tr>
<tr>
<td>SIS, SSM</td>
<td>semantic similarity of a composite service</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of service</td>
</tr>
<tr>
<td>QoSM</td>
<td>Quality of semantic matchmaking</td>
</tr>
<tr>
<td>tS, cS, rS, aS</td>
<td>Response time, cost, reliability, and availability of a composite service</td>
</tr>
<tr>
<td>prS</td>
<td>Failure probability of S</td>
</tr>
<tr>
<td>rE</td>
<td>Overall quality of composite services</td>
</tr>
<tr>
<td>lrN</td>
<td>Lower bound robustness estimation</td>
</tr>
<tr>
<td>svc</td>
<td>Monte Carlo robustness estimation</td>
</tr>
<tr>
<td>SRC</td>
<td>Semantic web service</td>
</tr>
<tr>
<td>S</td>
<td>Service repository</td>
</tr>
</tbody>
</table>

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 $QoS_S = \{tS, cS, rS, aS, prS\}$ 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 $O$. The attributes $tS, cS, rS, aS$, and $prS$ 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 $prS$ 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, $prS$ of newly published web services can be estimated as the $prS$ 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 $SRC$ is a finite collection of services supported by a common ontology $O$.

A composition task (also called service request) over a given $SRC$ 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 $O$. Two special atomic services $Start = (\emptyset, I_T, \emptyset)$ and $End = (O_T, \emptyset, \emptyset)$ are always included in $SRC$ 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 $O$ the matchmaking returns exact if $a$ and $b$ are equivalent ($a = b$), plugin if $a$ is a sub-concept of $b (a \subseteq b)$, subsume if $a$ is a super-concept of $b (a \supseteq b)$, and fail if none of previous matching types applies. In this paper we
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 \( \text{sim}(a,b) \) is calculated based on the edge counting method in a taxonomy like WordNet [49]. Advantages of this method are simple calculation and good semantic measurement [49]. As discussed in [48], we use matchmaking type (\( \text{type}_{\text{link}} \)) and semantic similarity (\( \text{sim}_{\text{link}} \)) to denote robust causal links, which is defined as follows:

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

\[
\text{sim}_{\text{link}} = \text{sim}(a,b) = \frac{2N_a}{N_a + N_b}
\]

with a suitable parameter \( p \), 0 < \( p \) < 1, and with \( N_a \), \( N_b \) and \( N_c \) which measure the distances from concept \( a \), concept \( b \), and the closest common ancestor \( c \) of \( a \) and \( b \) to the top concept of the ontology \( \mathcal{O} \), respectively. If more than one pair of matched output and input exist from service \( S \) to service \( S' \), \( \text{type}_{\text{c}} \) and \( \text{sim}_{\text{c}} \) 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:

\[
\text{MT} = \prod_{j=1}^{m} \text{type}_{\text{link}_j}
\]

\[
\text{SIM} = \frac{1}{m} \sum_{j=1}^{m} \text{sim}_{\text{link}_j}
\]

Formal expressions as in [50] are used to represent service compositions. The constructors \( \bullet, \|, + \) and \( * \) are used to denote sequential composition, parallel composition, choice, and iteration, respectively. The set of 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 \( \bullet(C_1, \ldots, C_d), \| (C_1, \ldots, C_d), + (C_1, \ldots, C_d), \) 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.

<table>
<thead>
<tr>
<th>( C )</th>
<th>( r_C )</th>
<th>( a_C )</th>
<th>( ct_C )</th>
<th>( t_C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (C_1, \ldots, C_d) )</td>
<td>( \prod_{k=1}^{d} r_{C_k} )</td>
<td>( \prod_{k=1}^{d} a_{C_k} )</td>
<td>( \sum_{k=1}^{d} ct_{C_k} )</td>
<td>( \sum_{k=1}^{d} t_{C_k} )</td>
</tr>
<tr>
<td>( (C_1, \ldots, C_d) )</td>
<td>( \prod_{k=1}^{d} r_{C_k} )</td>
<td>( \prod_{k=1}^{d} a_{C_k} )</td>
<td>( \sum_{k=1}^{d} ct_{C_k} )</td>
<td>( \text{MAX}(t_{C_k}</td>
</tr>
<tr>
<td>( + (C_1, \ldots, C_d) )</td>
<td>( \prod_{k=1}^{d} p_k \cdot r_{C_k} )</td>
<td>( \prod_{k=1}^{d} p_k \cdot a_{C_k} )</td>
<td>( \sum_{k=1}^{d} p_k \cdot ct_{C_k} )</td>
<td>( \sum_{k=1}^{d} p_k \cdot t_{C_k} )</td>
</tr>
</tbody>
</table>

| \( *C_0 \) | \( r_{C_0} \) | \( a_{C_0} \) | \( ct_{C_0} \) | \( t_{C_0} \) |

In the presentation of this paper, we mainly focus on two constructors, sequence \( \bullet \) and 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 in \( \Pi \) represents a unique index of a web service in the service repository. According to [8], a permutation \( \Pi \) needs to be interpreted, and can be further decoded into a \( \mathcal{G} \) (denoted as \( \Pi \triangleright \mathcal{G} \)). Such a decoding process can ensure the validity of composite services if \( \Pi \triangleright \mathcal{G} \) 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]:

\[
f_{eq}(\Pi) = \begin{cases} w_1 \text{MT} + w_2 \text{SIM} + w_3 \hat{A} + w_4 \hat{R} + w_5 (1 - \hat{T}) + w_6 (1 - \hat{CT}) & \text{if } \Pi \triangleright \mathcal{G} \\ 0 & \text{otherwise} \end{cases}
\]

with \( \sum_{k=1}^{6} w_k = 1 \). This objective function is defined as a comprehensive quality, denoted as \( f_{eq} \), for service composition. We can adjust the weights according to the user’s preferences. \( \text{MT} , \text{SIM} , \hat{A} , \hat{R} , \hat{T} \), and \( \hat{CT} \) are normalized to the range from 0 to 1 using Eq. (6). To simplify the presentation we also use the notation \( (Q_1, Q_2, Q_3, Q_4, Q_5, Q_6) = (\text{MT}, \text{SIM}, \hat{A}, \hat{R}, \hat{T}, \hat{CT}) \). \( Q_1 \) 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].

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

The goal of the static web service composition is to find a composite service \( C^* \) 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 Robust Web Service Composition for handling stochastic Service Failures (henceforth referred
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 $\mathcal{Q}$. A scenario $Q \in \mathcal{Q}$ corresponds to a set of services $\{S_j\}$ that remain accessible during the execution of a composite service, where $\sum_{Q \in \mathcal{Q}} Pr(Q) = 1$. Let $\mathcal{Z}(\Pi, Q)$ be a local search operator (i.e., an efficient re-optimization technique) that produces a new feasible composite solution $\Pi'$ for $Q$ through applying local changes to $\Pi$. 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:

$$r(\Pi) = \sum_{Q \in \mathcal{Q}} f_{eq}(\mathcal{Z}(\Pi, Q))Pr(Q) \approx \frac{1}{N} \sum_{i=1}^{N} f_{eq}(\mathcal{Z}(\Pi_i, Q_i))$$  \hspace{1cm} (7)$$

where $N$ is the sample size. Particularly, in Eq. (7), $\Pi$ is evaluated $N$ times based on $N$ sampled $Q_i$. $f_{eq}(\Pi)$ 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 requestor, 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 $\Pi$ with optimised robustness using a fitness function based on any proposed robustness measure, such as Eq. (5). This robust composite service $\Pi$ 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 $\Pi$ can always be repaired because no composition services $\mathcal{G}$ could be decoded from $\Pi$ for some scenarios, i.e., $\Pi \Rightarrow \mathcal{G}$ does not hold. In contrast, for other scenarios, a repaired solution $\Pi'$ 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
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:

$$r(\Pi) = \sum_{Q \in \mathcal{Q}} f_{cq}(\mathcal{L}(\Pi, Q))Pr(Q)$$

$$\geq \sum_{Q^* \in \mathcal{Q}^*} f_{cq}(\mathcal{L}(\Pi, Q^*))Pr(Q^*)$$

$$= \sum_{Q^* \in \mathcal{Q}^*} f_{cq}(\mathcal{L}(\Pi, Q^*))pr_{S_i} \prod_{j \neq i}(1 - pr_{S_j}) \quad (8)$$

where $Q^* \subseteq \mathcal{Q}$ are selected scenarios that only have one service failure, and any $Q^* \in \mathcal{Q}^*$ are not identical to each other. Therefore, the total number of scenarios in $\mathcal{Q}^*$ equals to $|\mathcal{SR}|$. Let $S_i$ be the failed service in every scenario, $Pr(Q^*)$ 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 $T = (T_F, O_T)$, 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 $A$ that plays the role of evolutionary control, and population $\mathcal{P}^0$ with $m$ randomly generated permutations $\Pi_k$ (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 $g^*$, 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, $g^*$ can be adaptively updated to control the lengths of stage one (i.e. $g < g^*$) and stage two (i.e. $g \geq g^*$). In stage one, we produce $m$ new
permutations to form the next generation \( P^{g+1} \) 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 \( P^{g+1} \) 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 \( SR \) consisting of six atomic services. \( S_1 = \{\{a, f\}, \{g\}, QoS_{S_1}\}, S_2 = \{\{b, e\}, \{c, d\}, QoS_{S_2}\}, S_3 = \{\{d, f\}, QoS_{S_3}\}, S_4 = \{\{a, h\}, QoS_{S_4}\}, S_5 = \{\{e, f\}, QoS_{S_5}\}. \) The two special services Start = \{\emptyset, \{a, b\}, \emptyset\} and End = \{\{e, f\}, \emptyset, \emptyset\} are defined by a given composition task \( T \). Fig. 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, 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 \( g^* \), at which stage two should begin. Specifically, \( g^* \) 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 \( g^* \) allows evolved solutions at the updated generation \( g^* \) achieve the highest possible robustness with the least computation resources due to the cheap evaluation method assigned to stage one.

**Algorithm 1. GA-2Stage for RWSC-SF.**

**Input:** composition task \( T \), Ontology \( O \), service repository \( SR \), and the number of neighbors \( n_{nb} \)

**Output:** a baseline solution

1. Set generation counter \( g \leftarrow 0 \);
2. Initialize an empty archive \( A \) and a \( P^g \) with \( m \) random permutations, each represented as a \( \Pi_k \) (where \( k = 1, \ldots, m \));
3. Evaluate each permutation in \( P^g \) using Eq. (5) based on its decoded DAG, \( G_k^g \);
4. Select \( g^* \) based on a updated \( A \) from \( P^g \);
5. while \( g < g_{max} \) do
6. if \( g < g^* \) then // stage one
7. \( P^{g+1} \) with \( m \) permutations from \( P^g \) through the use of genetic operators;
8. Evaluate each permutation in \( P^{g+1} \) using Eq. (5);
9. if \( g = g^* \) then // stage two starts
10. \( P^{g+1} \) with \( m \) permutations from \( A \);
11. Evaluate each permutation in \( P^{g+1} \) using Eq. (8);
12. if \( g > g^* \) then // stage two
13. \( P^{g+1} \) with \( m \) permutations from \( P^g \) through the use of genetic operators;
14. Evaluate each permutation in \( P^{g+1} \) using Eq. (8);
15. Set \( g \leftarrow g + 1 \);
16. Select the best solution \( \Pi_{g^{opt}} \) in \( P^g \) as a baseline;

**Algorithm 2. Generating \( g^* \) based on an adaptive archive-based evolutionary control.**

**Input:** archive \( A \) with maximal size \( m \), population \( P^g \), initial generation number \( g^* \) and generation increment step \( g_{inc} \)

**Output:** generation number \( g^* \)

1. Set generation counter \( g \leftarrow 0 \);
2. if \( g < g^* \) then
3. Update \( A \) with \( P^g \);
4. if \( g = g^* \) then
5. Evaluate each permutation in \( A \) using Eq. (8);
6. Calculate average robustness \( \bar{R} \) for permutations in \( A \);
7. Update \( A \) with \( P^g \);
8. Evaluate each permutation in \( A \) using Eq. (8);
9. Calculate average robustness \( \bar{R} \) for permutations in \( A \);
10. if \( \bar{R} > \bar{R} \) then
11. \( g^* \leftarrow g^* + g_{inc} \);
12. return \( g^* \);

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 \( P^g \), the generation \( g^* \) at which stage two should start, and generation increment step \( g_{inc} \) for \( g^* \). When \( g < g^* \), this algorithm updates the archive by storing all distinct composite services from \( P^g \) based on the comprehensive quality in a descending order. Subsequently, when \( g = g^* \), we evaluate the robustness of each permutation in the archive using Eq. (8).
and calculate the average robustness of all the permutations as $\mathcal{R}$. After calculating the average robustness $\mathcal{R}$, we calculate the average robustness again as $\mathcal{R}'$ after updating the archive. The archive is updated in the same way as we discussed above. Consequently, we increase $g^*$ by $g_{uc}$ if $\mathcal{R}' > \mathcal{R}$. Otherwise, $g^*$ 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.

![Crossover and mutation](image)

**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 $\Pi$ in a population $\mathcal{P}^*$ 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 $T = \{(a, b), \{c, f\}\}$ and a service repository $\mathcal{SR}$ consisting of six atomic services. $S_1 = \{\{e, f\}, \{g\}, QoS_{S_0}\}$, $S_2 = \{\{b\}, \{c, d\}, QoS_{S_2}\}$, $S_3 = \{\{e\}, \{g\}, QoS_{S_3}\}$, $S_4 = \{\{a\}, \{h\}, QoS_{S_4}\}$ and $S_5 = \{\{c\}, \{e, f\}, QoS_{S_5}\}$. The two special services $\text{Start} = \{\emptyset, \{a, b\}, \emptyset\}$ and $\text{End} = \{\{e, f\}, \emptyset, \emptyset\}$ 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 how a new permutation (i.e., $\Pi^*$) 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 $\Pi = [4, 1, 0, 2, 3, 4]$ as an example solution to the service composition problem. To encode scenarios for a given permutation $\Pi$, we produce another permutation $\Pi^*$ 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 $\Pi^*$ can keep some promising component services from the candidate $\Pi$. For example, two permutations with two decoded DAGs are created for encoding two scenarios $Q_1$ and $Q_5$ respectively. In the two decoded DAGs, we can see that the promising service index 1 of the decoded $\mathcal{G}^*$ from $\Pi^*$ is inherited from that of the $\mathcal{G}$ decoded from $\Pi$ in Fig. 5. On the other hand, the decoded DAG $\mathcal{G}^*$ in $Q_5$ 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 $\mathcal{G}^*$ 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 $\mathcal{G}^*$, sorted based on the longest distance from $\text{Start}$ to every component services of $\mathcal{G}^*$ while the second part $[4, 0]$ is indexes of remaining services that are available but not utilised by $\mathcal{G}^*$.

Let $\Pi^* = (\pi_0, \ldots, \pi_t, |\pi_{t+1}, \ldots, \pi_{n-1})$ be the produced
Given a candidate permutation \( \Pi \) with its decoded \( \mathcal{G} \):

- 1st: Remove it \( \ell_{0}(\{e, f\}) \)
- 2nd: \( \ell_{1}(a, b) \)
- 3rd: \( \ell_{t}(\{a, b\}) \)
- 4th: \( \ell_{t}(\{e, f\}) \)

Permutation \( \Pi^* \):
- Start
- 1 2 3 4 0
- Unused services: Used services: 

Deployed solution remains the same.

**Fig. 5:** An example of 6 selected scenarios involved in the lower bound robustness estimation and a new permutation \( \Pi^* \) 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 \( \text{Start} \) to every component service of \( \mathcal{G} \), while \( \{t + 1, \ldots, n - 1\} \) are indexes of remaining services in \( \mathcal{S} \mathcal{R} \) not utilised by \( \mathcal{G} \). Subsequently, we apply a stochastic local search operator (layer-based constrained one-point swap, see details in [55]) to \( \Pi^* \). 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 \( \Pi_a \) and \( \Pi_b \) in the permutation. Particularly, one service index \( \Pi_a \), 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 \( \Pi_b \) 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 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)

<table>
<thead>
<tr>
<th>Task</th>
<th>GA-2Stage</th>
<th>GA-RE</th>
<th>FL</th>
<th>GA-MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWLS-TC 1</td>
<td>0.922788 ± 0.000179</td>
<td>0.922862 ± 0.00018</td>
<td>0.922791 ± 0.000111</td>
<td>0.916621 ± 0.002412</td>
</tr>
<tr>
<td>OWLS-TC 2</td>
<td>0.931586 ± 0.002107</td>
<td>0.932095 ± 0.000277</td>
<td>0.92618 ± 0.005009</td>
<td>0.915558 ± 0.025327</td>
</tr>
<tr>
<td>OWLS-TC 3</td>
<td>0.863518 ± 0.003623</td>
<td>0.863061 ± 0.00725</td>
<td>0.854218 ± 0.00779</td>
<td>0.862459 ± 0.003403</td>
</tr>
<tr>
<td>OWLS-TC 4</td>
<td>0.789396 ± 0.00857</td>
<td>0.791174 ± 0.004451</td>
<td>0.779121 ± 0.012348</td>
<td>0.789101 ± 0.005596</td>
</tr>
<tr>
<td>OWLS-TC 5</td>
<td>0.828751 ± 0.005054</td>
<td>0.825059 ± 0.00275</td>
<td>0.812652 ± 0.012398</td>
<td>0.824206 ± 0.013075</td>
</tr>
<tr>
<td>WSC08-1</td>
<td>0.399791 ± 0.002975</td>
<td>0.398686 ± 0.003538</td>
<td>0.395194 ± 0.002752</td>
<td>0.383542 ± 0.005418</td>
</tr>
<tr>
<td>WSC08-2</td>
<td>0.576417 ± 0.00378</td>
<td>0.576716 ± 0.00289</td>
<td>0.568166 ± 0.005362</td>
<td>0.572707 ± 0.006905</td>
</tr>
<tr>
<td>WSC08-3</td>
<td>0.025494 ± 0.00295</td>
<td>0.02501 ± 0.001415</td>
<td>0.025118 ± 0.001333</td>
<td>0.02509 ± 0.001424</td>
</tr>
<tr>
<td>WSC08-4</td>
<td>0.274788 ± 0.00295</td>
<td>0.274563 ± 0.002672</td>
<td>0.271112 ± 0.004139</td>
<td>0.268906 ± 0.003492</td>
</tr>
<tr>
<td>WSC08-5</td>
<td>0.275513 ± 0.002107</td>
<td>0.275814 ± 0.003147</td>
<td>0.275259 ± 0.002625</td>
<td>0.272394 ± 0.003412</td>
</tr>
<tr>
<td>WSC08-6</td>
<td>0.072178 ± 0.002047</td>
<td>0.072036 ± 0.001605</td>
<td>0.072017 ± 0.002114</td>
<td>0.071496 ± 0.002028</td>
</tr>
<tr>
<td>WSC08-7</td>
<td>0.216090 ± 0.003111</td>
<td>0.217957 ± 0.003922</td>
<td>0.216321 ± 0.003332</td>
<td>0.214177 ± 0.003333</td>
</tr>
<tr>
<td>WSC08-8</td>
<td>0.053596 ± 0.002047</td>
<td>0.054259 ± 0.001909</td>
<td>0.0536 ± 0.002007</td>
<td>0.053540 ± 0.001992</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>GA-2Stage</th>
<th>GA-RE</th>
<th>FL</th>
<th>GA-MC</th>
<th>OWLS-TC (5 tasks)</th>
<th>WSC08 (8 tasks)</th>
<th>WSC09 (5 tasks)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GA-2Stage</td>
<td>GA-RE</td>
<td>FL</td>
<td>GA-MC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OWLS-TC</td>
<td>GA-2Stage</td>
<td>0/5/0</td>
<td>0/1</td>
<td>-</td>
<td>0/1</td>
<td>0/1/4</td>
<td>0/1/4</td>
<td>0/1/4</td>
</tr>
<tr>
<td></td>
<td>GA-RE</td>
<td>4/1/0</td>
<td>4/1/0</td>
<td>-</td>
<td>2/3/0</td>
<td>0/1/4</td>
<td>0/1/4</td>
<td>0/1/4</td>
</tr>
<tr>
<td></td>
<td>FL</td>
<td>2/3/0</td>
<td>2/3/0</td>
<td>-</td>
<td>2/3/0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WSC08</td>
<td>GA-2Stage</td>
<td>0/8/0</td>
<td>0/8/0</td>
<td>-</td>
<td>0/5/3</td>
<td>0/5/3</td>
<td>0/5/3</td>
<td>0/5/3</td>
</tr>
<tr>
<td></td>
<td>GA-RE</td>
<td>3/5/0</td>
<td>3/5/0</td>
<td>-</td>
<td>0/5/3</td>
<td>0/5/3</td>
<td>0/5/3</td>
<td>0/5/3</td>
</tr>
<tr>
<td></td>
<td>FL</td>
<td>3/5/0</td>
<td>3/5/0</td>
<td>-</td>
<td>0/5/3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WSC09</td>
<td>GA-2Stage</td>
<td>0/5/0</td>
<td>0/5/0</td>
<td>-</td>
<td>0/3/3</td>
<td>0/3/3</td>
<td>0/3/3</td>
<td>0/3/3</td>
</tr>
<tr>
<td></td>
<td>GA-RE</td>
<td>2/3/0</td>
<td>2/3/0</td>
<td>-</td>
<td>1/3/1</td>
<td>0/3/3</td>
<td>0/3/3</td>
<td>0/3/3</td>
</tr>
<tr>
<td></td>
<td>FL</td>
<td>4/1/0</td>
<td>4/1/0</td>
<td>-</td>
<td>1/3/1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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 GA-2Stage, 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 produced by the runs.
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 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

<table>
<thead>
<tr>
<th>Task</th>
<th>GA-2Stage</th>
<th>GA-RE</th>
<th>FL</th>
<th>GA-MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWLS-TC2</td>
<td>1.956923± 3.783479</td>
<td>2.546923± 8.369423</td>
<td>3.295923± 5.369423</td>
<td>4.128923± 6.359423</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Task</th>
<th>GA-2Stage</th>
<th>GA-RE</th>
<th>FL</th>
<th>GA-MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWLS-TC2</td>
<td>1.956923± 3.783479</td>
<td>2.546923± 8.369423</td>
<td>3.295923± 5.369423</td>
<td>4.128923± 6.359423</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Method</th>
<th>GA-2Stage</th>
<th>GA-RE</th>
<th>FL</th>
<th>GA-MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWLS-TC (5 tasks)</td>
<td>GA-2Stage</td>
<td>5/0/0</td>
<td>0/0/5</td>
<td>0/5/0</td>
<td>0/0/5</td>
</tr>
<tr>
<td></td>
<td>GA-RE</td>
<td>5/0/0</td>
<td>0/0/5</td>
<td>0/5/0</td>
<td>0/0/5</td>
</tr>
<tr>
<td></td>
<td>FL</td>
<td>5/0/0</td>
<td>0/0/5</td>
<td>-</td>
<td>0/0/5</td>
</tr>
<tr>
<td></td>
<td>GA-MC</td>
<td>5/0/0</td>
<td>0/0/5</td>
<td>0/5/0</td>
<td>-</td>
</tr>
<tr>
<td>WSC08 (8 tasks)</td>
<td>GA-2Stage</td>
<td>8/0/0</td>
<td>0/0/8</td>
<td>0/8/0</td>
<td>0/0/8</td>
</tr>
<tr>
<td></td>
<td>GA-RE</td>
<td>8/0/0</td>
<td>0/0/8</td>
<td>0/8/0</td>
<td>0/0/8</td>
</tr>
<tr>
<td></td>
<td>FL</td>
<td>8/0/0</td>
<td>0/0/8</td>
<td>-</td>
<td>0/0/8</td>
</tr>
<tr>
<td></td>
<td>GA-MC</td>
<td>8/0/0</td>
<td>0/0/8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WSC09 (5 tasks)</td>
<td>GA-2Stage</td>
<td>5/0/0</td>
<td>0/0/5</td>
<td>5/0/0</td>
<td>0/0/5</td>
</tr>
<tr>
<td></td>
<td>GA-RE</td>
<td>5/0/0</td>
<td>0/0/5</td>
<td>0/5/0</td>
<td>0/0/5</td>
</tr>
<tr>
<td></td>
<td>FL</td>
<td>5/0/0</td>
<td>0/0/5</td>
<td>-</td>
<td>0/0/5</td>
</tr>
<tr>
<td></td>
<td>GA-MC</td>
<td>5/0/0</td>
<td>0/0/5</td>
<td>0/5/0</td>
<td>-</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>( r_{LB} )</th>
<th>( r_{MC} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pearson</td>
<td>0.611078</td>
<td>0.428310</td>
</tr>
<tr>
<td>Kendall’s tau</td>
<td>0.452137</td>
<td>0.310345</td>
</tr>
<tr>
<td>Spearman’s rho</td>
<td>0.615333</td>
<td>0.493215</td>
</tr>
</tbody>
</table>

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

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


