J Intell Manuf
DOI 10.1007/s10845-016-1215-0

A hybrid approach using genetic and fruit fly optimization algorithms for QoS-aware cloud service composition

Fateh Seghir[1] · **Abdellah Khababa**[1]

Received: 25 December 2015 / Accepted: 30 March 2016

Abstract This paper addresses the QoS-aware cloud service composition problem, which is known as a NP-hard problem, and proposes a hybrid genetic algorithm (HGA) to solve it. The proposed algorithm combines two phases to perform the evolutionary process search, including genetic algorithm phase and fruit fly optimization phase. In genetic algorithm phase, a novel roulette wheel selection operator is proposed to enhance the efficiency and the exploration search. To reduce the computation time and to maintain a balance between the exploration and exploitation abilities of the proposed HGA, the fruit fly optimization phase is incorporated as a local search strategy. In order to speed-up the convergence of the proposed algorithm, the initial population of HGA is created on the basis of a heuristic local selection method, and the elitism strategy is applied in each generation to prevent the loss of the best solutions during the evolutionary process. The parameter settings of our HGA were tuned and calibrated using the taguchi method of design of experiment, and we suggested the optimal values of these parameters. The experimental results show that the proposed algorithm outperforms the simple genetic algorithm, simple fruit fly optimization algorithm, and another recently proposed algorithm (DGABC) in terms of optimality, computation time, convergence speed and feasibility rate.

Keywords Service composition · Cloud computing · Quality of service (QoS) · Genetic algorithm · Fruit fly optimization algorithm

✉ Fateh Seghir
seghir.fateh@gmail.com

Abdellah Khababa
khababa_abdelh@yahoo.fr

[1] Department of Computer Science, University of Ferhat Abbas-Setif 1, 19000 Sétif, Algeria

Introduction

With the proliferation of the cloud computing, more and more cloud services providing similar functionalities but offering different quality of service (QoS), such as execution time, price and availability ... etc., will be offered on the web. Therefore, selecting the optimal simple cloud services to build an optimal composite service that meets the global end-to-end QoS constraints, is one of the most important problems in service composition, which is called QoS-aware cloud service composition problem (Jula et al. 2014; Tao et al. 2008).

The QoS-aware cloud service composition (QoS-CSC) is a combinatorial optimization problem, which is also a strong NP-hard optimization one (Tao et al. 2008, 2013). Many studies including exact and meta-heuristic methods have been proposed to resolve the QoS-CSC. Due to the large scale of the QoS-CSC problem, the application of exact approaches such as linear programming (Zeng et al. 2004; Alrifai and Risse 2009) and mixed integer programming (Ardagna and Pernici 2007; Alrifai et al. 2010) are restricted. Other kind of approaches, optimization methods based on bio-inspired and meta-heuristic algorithms have shown better performance in solving the QoS-CSC problem. These include genetic algorithm (GA) (Canfora et al. 2005; Yue and Chengwen 2008; Zhi-peng et al. 2009; Maolin and feng 2010; Wu et al. 2014; Jin et al. 2015; Ding et al. 2015), particle swarm optimization (PSO) (Tao et al. 2008; Liao et al. 2014; Wang et al. 2013), ant colony optimization (AC) (Wu and Zhu 2013), artificial bee colony (ABC) (Huo et al. 2015; Xue et al. 2016) and so on. Optimization approaches can obtain near-optimal solutions within a short period of time; thus, it has attracted the attention of academia in the last years.

Very recently, a novel global evolutionary optimization: fruit fly optimization algorithm (FOA) is proposed by Pan

Published online: 08 April 2016

(2012), who is inspired by the food finding behavior of the fruit fly. Compared with other evolutionary algorithms, such as GA, PSO, and AC . . . etc.; it has few parameters to adjust, easy to implement and it has proven to be more effective to solve different kinds of discrete and continuous optimization problems such as multidimensional knapsack problems (Wang et al. 2013; Zhang et al. 2015), financial distress model (Pan 2012), semiconductor final testing scheduling problem (Zheng et al. 2014), homogeneous fuzzy series-parallel redundancy allocation problem (Mousavi et al. 2015a), a location allocation-inventory problem in a two-echelon supply chain network (Mousavi et al. 2015b), steelmaking casting problem (Li et al. 2014) and stand-alone hybrid photovoltaic (PV)-wind-diesel-battery system (Jingyi and Xiaofang 2015). To the best of our knowledge, there are few works about FOA for solving the QoS-CSC. In Zhang et al. (2015), propose an improved optimization algorithm based on FOA for service composition (WS_FOA), in which each parameter of FOA is described according to the service composition model; the experimental results show that WS_FOA has higher operational efficiency than PSO. However, the traditional FOA algorithm lacks the ability to explore the solution search, which is easy to fall into local optima (Wang et al. 2015). In view of these, and in order to benefit from the higher exploitation ability of FOA (Zhang et al. 2015) and the exploration ability of the GA, in this paper, the FOA process is integrated in the GA as a local search strategy.

This paper proposes an optimal optimization algorithm for QoS-CSC problem (HGA) based on GA and FOA algorithms. First, we present a heuristic local selection method for generating an improved initial population to speed-up the convergence of our proposed algorithm. Next, the genetic phase, which contains a novel roulette wheel selection operator, a probabilistic one/two point crossover operator and mutation operator. The latters are employed in the HGA to enhance the exploration search process. After that, and in order to improve the exploitation based search of HGA, the FOA phase is used as local search process. Moreover, the elitism strategy is applied in each generation of HGA to prevent the loss of the best solutions during the evolutionary process. Finally, a design of experiment (DOE) via Taguchi method is used to optimize the HGA's parameter settings; the experimental results show that the proposed algorithm is more effective than the simple genetic algorithm used in Canfora et al. (2005), simple fruit fly optimization algorithm (Pan 2012) and the one stated in Huo et al. (2015), which can obtain an accurate and optimal global solution in a reasonable computation time.

The remainder of this paper is organized as follows: Sect. "Related work" discusses the related literature. In Sect. "GA, FOA and problem description" the QoS-CSC problem is described after introducing the canonical algorithms GA and FOA briefly. In Sect. "The proposed approach", the proposed algorithm is presented in details, including the encoding solution schema and its fitness evaluation, population initialization process, the applied genetic operators, and the FOA phase; we end this very section with presenting the framework of the proposed HGA. In Sect. "Experiments", the parameter settings of HGA are investigated and our experimental results are shown. Finally, we conclude this paper with a conclusion and the future work in Sect. "Conclusion and future work".

Related work

Many approaches have been proposed for QoS-CSC problem in the web service and cloud computing contexts, which can be classified into two categories: (a) exact algorithms such as linear programming and mixed integer linear programming (Zeng et al. 2004; Ardagna and Pernici 2007; Alrifai and Risse 2009; Alrifai et al. 2010). (b) Optimization methods based on bio-inspired and meta-heuristic algorithms (Canfora et al. 2005; Yue and Chengwen 2008; Zhi-peng et al. 2009; Maolin and feng 2010; Wu et al. 2014; Jin et al. 2015; Liao et al. 2014; Wang et al. 2013; Wu and Zhu 2013; El Haddad et al. 2010; Ding et al. 2015; Huo et al. 2015; Xue et al. 2016).

Exact algorithms

In Zeng et al. (2004), formulated the QoS-driven web service selection problem as a linear integer programming and they proposed two approaches to tackle it: local selection method and global optimization method. The former is more efficient than the latter method in computation time, because the local selection approach's execution time is polynomial, and it cannot guarantee the global QoS or satisfy the global QoS constraints. The global method overcomes these shortcomings, but with an expansive computation cost. In Ardagna and Pernici (2007), the web service selection is formalized as a mixed integer linear programming (MILP), which is solved using the linear CPLEX solver. Negotiations techniques for QoS parameters are performed unless there are feasible solutions to the problem. In Alrifai and Risse (2009), combined the local and global selection methods to benefit from the advantages of those techniques, and then proposed a hybrid approach to solve the QoS web service selection problem. The integer linear programming model is used to decompose the global QoS constraints into local ones; the distributed local selection finds the near-optimal solution that meet these local constraints unlike the optimal solution, which is with a high computation cost. To deal with the scalability issues of linear integer programming techniques, in Alrifai et al. (2010), an offline skyline operator is used to reduce the number of candidate services (i.e. eliminate the dominated

services) from the research space prior the application of the MILP selection algorithm. The linear programming methods are very efficient and can obtain an optimal solution. However, the poor scalability of these methods due to their computation complexity time, and moreover, their objective functions and global constraints must be linear, which limits these methods to some extent.

Optimization methods

In recent decades some evolutionary and meta-heuristic algorithms have been employed to solve the QoS-CSC problem. These methods are faster than MIP for large-scale problems, and allow the consideration of non-linear composition rules (Ardagna and Pernici 2007; Alrifai and Risse 2009), and thus, they are suitable to tackle the QoS-CSC problem. In Canfora et al. (2005), has proposed a method based on a simple genetic algorithm, where the genome is encoded as an integer array to represent the composite service, and its penalized fitness function is used to drive the evolution process towards a constraint satisfaction (i.e. two types of penalties: static and dynamic are proposed for chromosomes that violate constraints). In Yue and Chengwen (2008), an improved genetic algorithm named (CoDiGA) is proposed for web service composition; a relation matrix coding scheme is introduced to express all paths of a composite service at the same time, an enhanced initial population and evolution policy are proposed based on population diversity in order to quick the convergence of CoDiGA. In Zhi-peng et al. (2009), a simulated-annealing-based genetic algorithm (QQDSGA) is provided to tackle the QoS-aware web services selection problem, the presented simulations show that QQDSGA is better than GA and simulated annealing (SA) in terms of efficiency. Differently from the above GA's approaches, a hybrid genetic algorithm (HGA) is proposed by Maolin and feng (2010), where both, QoS proprieties and dependency and conflict constraints among web services are taken in consideration to tackle the constrained web service composition problem. The research by Wu et al. (2014) uses the fitness sharing method to more exploit areas of the search space in the proposed genetic algorithm approach, which takes account business correlations among candidate services. In Jin et al. (2015), a genetic algorithm approach is proposed using ranking method to calculate the fitness of individuals preventing premature convergence, where the correlations among cloud services are also considered. In Tao et al. (2008), PSO algorithm with an array integer encoding scheme is applied to find out a good solution to multi-objective MGrid resource service composition and optimal-selection (MO-MRSCOS) problem, the experimental results show that the proposed methods are sound on both efficiency and effectiveness for solving MO-MRSCOS. The PSO algorithm for service composition problem is also improved by some other authors (Liao et al. 2014; Wang et al. 2013).

In Wu and Zhu (2013), model the problem of QoS-aware dynamic service composition as directed acyclic graph, where both non-functional properties (QoS) and transactional behavior among web services are considered, and they proposed an ant-colony algorithm to solve this problem. Their experimental results show that the proposed AC approach outperforms the local Transactional-QoS optimization method proposed by El Haddad et al. (2010). The transactional and QoS-aware web service selection problem is also investigated by Ding et al. (2015), where they proposed an approach based on GA. The ABC algortihm is used by many authors to tackle the QoS-CSC problem. In Huo et al. (2015), a discrete gbest-guided artificial bee colony (DGABC) is proposed as it simulates the search for the optimal service composition solution via the exploration of bees for food. In Xue et al. (2016), proposed an improved ABC algorithm, which is brought into the framework of genetic algorithm, to construct the complete service composition method.

Meta-heuristic approaches can obtain near-optimal solutions within a short period of time; thus, it has attracted the attention of academia in last years. However, this kind of methods presents some drawbacks, for instance premature or local optimum stagnation and the slow convergence to find near-optimal solutions. Moreover, the large space search of the QoS-CSC is a critical problem related to the accuracy of the solution, and thus, our motivation in this very study will focus on designing an effective algorithm to solve the problem.

GA, FOA and problem description

Genetic algorithm (GA)

GA is a probabilistic search for an algorithm that mimics the process of biological evolution in a naturel environment Goldberg (1989), which has been widely applied to solve several optimization problems, such as combinatorial optimization, production scheduling, and other fields. The basic GA flowchart is shown in Fig. 1. At the beginning of the algorithm, an initial population of individuals (chromosomes) is randomly generated, after that, each chromosome which encodes a single possible solution to the problem in the form of encoded string is evaluated according to its defined fitness function. Next, a new population (generation) is produced by using three genetic operators: selection, crossover and mutation. The generated solutions (offspring) are taken to replace the old population (parents) for the next iteration (evolution) depending on their fitness. This process of evolution is repeated for each generation until the stopping criteria of the algorithm is reached.

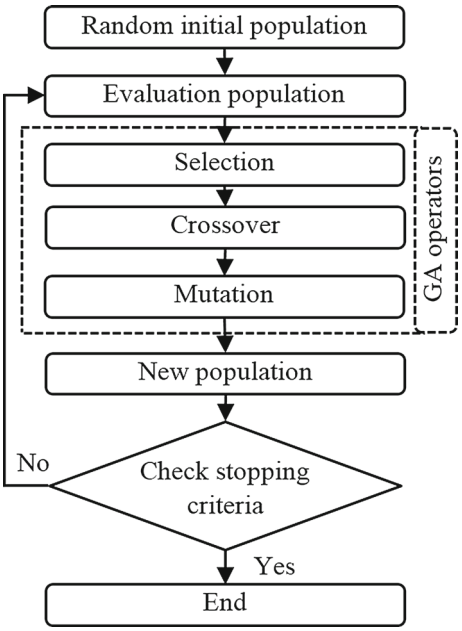

Fig. 1 Flowchart of simple genetic algorithm

Fruit fly optimization algorithm (FOA)

Very recently, a new global optimization algorithm, called Fruit fly Optimization Algorithm (FOA), has been presented by Pan (2012), which was inspired by the food finding behavior of the fruit fly. The fruit fly itself is superior to other species in sensing and perceiving, especially in osphresis and vision (Pan 2012). FOA uses two main foraging processes to achieve an optimal global optimization. The first foraging process is to locate the food source through smelling by using osphresis organ and fly towards the corresponding location (i.e. smell-based search process), the second foraging process is to use the foraging sensitive vision to find the best food source location and fly towards it (i.e. vision-based search process). The following steps summarize the behavior of FOA (Pan 2012; Zheng et al. 2014; Li et al. 2014).

Step 1. Initialization of parameters.

1. Initialize the fruit fly group location: generate randomly the location of the fruit fly group in the search space.
2. Initialize the maximum number of generations (MAX_GEN) and population size (SN).

Step 2. Smell-based search process.

1. Foraging smell: generate randomly a population of SN fruit flies around the current fruit fly group location.
2. Evaluate the population: evaluate the fitness value (smell concentration) for each fruit fly of the generated population.

Step 3. Vision-based search process: find the best fruit fly with the optimal fitness, and then let the fruit fly group fly towards the best one (the current fruit fly group location is updated with the position of the best one).

Step 4. Check the stopping criterion: if the maximum number of generations MAX_GEN is reached stop the algorithm; otherwise, go back to step 2.

Notation and problem description

Let consider the following descriptions:

- $ACSC = \{T_1, T_2, \ldots, T_n\}$, denotes an abstract cloud service composition, in which $T_{i\,(i=1,2,\ldots n)}$ are atomic tasks that comprise it and their total number is (n). Each task T_i is related to the candidate cloud service set $S_i = (CS_i^1, CS_i^2, \ldots, CS_i^m)$, where CS_i^j is the jth candidate cloud service, and m is the number of candidate cloud services of each candidate set S_i.
- The set $QoS = \{q_1, q_2, \ldots q_r\}$ of criteria for each cloud service CS can be classified into two classes: positive and negative QoS attributes (denoted as QoS^+ and QoS^-, respectively), where r is the total number of concerned QoS parameters, and for each positive criterion, larger values indicate higher performance (e.g. reliability and availability) while for negative criterion, smaller values indicate higher performance (e.g. price and response time).
- The set $GCst = \{Cst_1, Cst_2, \ldots, Cst_k\}$ indicates the global constraints specified by the user, where Cst_t with $t \leq k$ and $k \leq r$ is the global QoS constraint over the QoS attribute q_t.
- $W = \{w_1, w_2, \ldots, w_r\}$, denotes the user's preferences for each QoS attribute $q_t \in QoS$, and $w_t \in [0, 1]$ is the weight of the tth QoS attribute with $\sum_{t=1}^{r} w_t = 1$.

The conceptual overview of cloud service composition is shown in Fig. 2. Two main processes are invoked using the functional descriptions (i.e. inputs and outputs descriptions of $ACSC$) and non-functional values (i.e. global QoS constraints and weights of user's preferences of QoS attributes) in order to resolve the user's request. These processes are the discovery process, and QoS-aware cloud service composition process. In the discovery process, sets of candidate cloud services $S_{i\,(i=1\ldots n)}$ that can perform tasks $T_{i\,(i=1\ldots n)}$ are discovered through functional descriptions. The *QoS-aware cloud service composition* process aims to find a composite of concrete cloud services $CCS = (CS_1^{j_1}, CS_2^{j_2}, \ldots, CS_n^{j_n})$ by binding each task T_i to a selected cloud service $CS_i^{j_i}\,(j_i \in [1, m])$ from the related set of candidate cloud services S_i, where these selected services satisfy the non-functional conditions of the global QoS constraints $GCst$

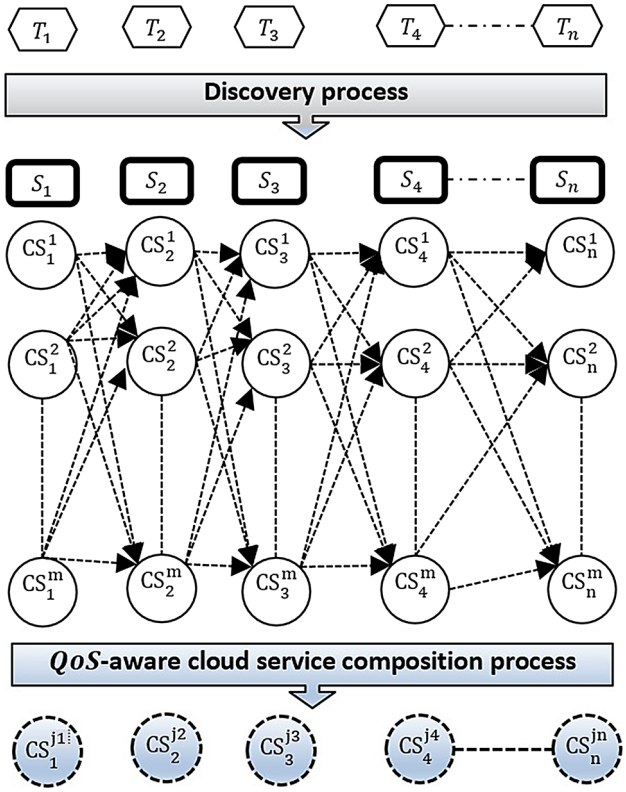

Fig. 2 Conceptual overview of cloud service composition

and have the optimal value of global QoS on the basis of user's preferences W.

In order to calculate the score value of the global QoS of each CCS, we evaluate this according to the QoS values of its component cloud services, and composite models (i.e. sequential, parallel, conditional and loop), which define the interconnection structures among these component services. The simple additive weighting (SAW) method is applied (Zeleny 1982) to map the aggregated QoS values of CCS into a single real value. SAW method contains two steps: scaling and weighting. The former normalizes the various measurement metrics of QoS attributes to the same scale into real values between 0 and 1. (Eqs. 1 and 2 show the normalization step of each aggregate attribute q_t for CCS). The later multiplies each normalized value with a preference weight w_t and then aggregates them into score value of this CCS, (Eq. 3 show the score value of each CCS).

– normalization of aggregate negative attributes (i.e. $q_t \in QoS^-$):

$$Q(CCS)_t = \begin{cases} \frac{agg(q_t^{max}) - agg(q_t)}{agg(q_t^{max}) - agg(q_t^{min})} & \text{if } agg(q_t^{max}) \neq agg(q_t^{min}) \\ 1 & \text{if } agg(q_t^{max}) = agg(q_t^{min}) \end{cases} \quad (1)$$

– normalalization of aggregate positive attributes(i.e. $q_t \in QoS^+$:):

$$Q(CCS)_t = \begin{cases} \frac{agg(q_t) - agg(q_t^{min})}{agg(q_t^{max}) - agg(q_t^{min})} & \text{if } agg(q_t^{max}) \neq agg(q_t^{min}) \\ 1 & \text{if } agg(q_t^{max}) = agg(q_t^{min}) \end{cases} \quad (2)$$

In Eqs. 1 and 2, $agg(q_t^{max})$ and $agg(q_t^{min})$ denote the maximum and minimum possible aggregated values of the tth QoS criterion for the composite CCS, respectively, and $agg(q_t)$ denotes the aggregated value of the tth QoS criterion of CCS.

$$Score(CCS) = \sum_{t=1}^{r} Q(CCS)_t * w_t \quad (3)$$

The aggregation operation (i.e. $agg = \{\sum, \prod, avg$, min and max$\}$) for each QoS attribute can be applied according to the composite model and the characteristic of the corresponding QoS parameter. For example, the sum operation (i.e. $\sum$) is the aggregation operation of the QoS response time for the sequential composite model. The calculation score formulas of the most used QoS attributes in literature for the different composition models (Tao et al. 2008; Zeng et al. 2004) are shown in Table 1. In this study, only the sequential composite model is considered, while the other models can be simplified or converted using the methods mentioned in Ardagna and Pernici (2007).Using the aforementioned descriptions and the above evaluation method to calculate the global QoS score of CCS, the QoS-CSC is planned to find the optimal CCS with the maximum score value and meet the global QoS constraints $GCst$, which can be formulated as follows:

$$\max : Score(CCS) \quad (4)$$

$$subject\ to$$

$$\forall t = 1 \ldots k \begin{cases} agg(q_t) \leq Cst_t \text{ if } q_t \in QoS^- \\ agg(q_t) \geq Cst_t \text{ if } q_t \in QoS^+, \end{cases} \quad (5)$$

where $Cst_t \in GCst$, and $agg(q_t)$ denotes the aggregated value of the tth QoS criterion of the composite concrete cloud services CCS.

The proposed approach

In this section, we present the design of our HGA for QoS-CSC problem, which includes the solution's encoding scheme, improved population initialization, fitness evaluation of solution, the genetic algorithm phase in which a novel selection operator is proposed to enhance the efficiency and exploration ability, and in order to reduce the computation time and to maintain a balance between exploration and exploitation abilities of HGA, the fruit fly optimization phase is performed after every genetic evolution, and the elitism

Table 1 QoS aggregation functions for the different composition models.

Attribute (q_t)	Composition model			
	Sequential: (n sequential cloud services)	Parallel: (m parallel cloud services)	Conditional: (call CS_i^j with pr_i probability)	Loop: (call CS_i^j h times)
price ($t = price$)	$\sum_{i=1}^{n}(q_{price,i}^{j})$	$\sum_{i=1}^{m}(q_{price,i}^{j})$	$\sum_{i=1}^{m}(q_{price,i}^{j} * pr_i)$	$h * (q_{price,i}^{j})$
time ($t = time$)	$\sum_{i=1}^{n}(q_{time,i}^{j})$	$\max_{i=1}^{m}(q_{time,i}^{j})$	$\sum_{i=1}^{m}(q_{time,i}^{j} * pr_i)$	$h * (q_{time,i}^{j})$
availability ($t = ava$)	$\prod_{i=1}^{n}(q_{ava,i}^{j})$	$\prod_{i=1}^{m}(q_{ava,i}^{j})$	$\sum_{i=1}^{m}(q_{ava,i}^{j} * pr_i)$	$(q_{ava,i}^{j})^h$
reliability ($t = rel$)	$\prod_{i=1}^{n}(q_{rel,i}^{j})$	$\prod_{i=1}^{m}(q_{rel,i}^{j})$	$\sum_{i=1}^{m}(q_{rel,i}^{j} * pr_i)$	$(q_{rel,i}^{j})^h$
reputation ($t = rpt$)	$avg_{i=1}^{n}(q_{rpt,i}^{j})$	$avg_{i=1}^{m}(q_{rpt,i}^{j})$	$\sum_{i=1}^{m}(q_{rpt,i}^{j} * pr_i)$	$q_{rpt,i}^{j}$
throughput ($t = thr$)	$\min_{i=1}^{n}(q_{thr,i}^{j})$	$\min_{i=1}^{m}(q_{thr,i}^{j})$	$\sum_{i=1}^{m}(q_{thr,i}^{j} * pr_i)$	$q_{thr,i}^{j}$

j is the index value of the selected concrete cloud service from the related candidate set S_i. In the case of the conditional compoistion model: $\sum_{i=1}^{m} pr_i = 1$ and m its number of choices

operator is applied after the termination of the fruit fly optimization phase to prevent the loss of the best solutions during the evolutionary process. In the following subsections, we present the components of the proposed algorithm and its framework.

Encoding scheme

In the proposed HGA, each individual is a solution of the QoS-CSC problem, which is represented by an array of integers. Each integer in the array corresponds to a chosen concrete cloud service from the corresponding candidate set of concrete cloud services. In Fig. 3 the shown individual represents composite concrete cloud service consisting of five abstract cloud services, in which each entry in the array encodes the selected concrete cloud service from its related candidate set. For example, the second entry in the array represents the first concrete cloud service of the second abstract cloud service (i.e. the concrete cloud service: CS_2^1).

Population initialization

In general, the initial population of an interactive evolutionary algorithm such as GA or FOA is generated randomly. But a population with a high level of solution quality and diversity is very crucial in order to get a near-optimal solution with short convergence time. In this study we propose a heuristic local optimization selection method to generate an improved solution and then we present the procedure of generating the initial population of our HGA.

Local optimization selection method

To create an individual (ind) with high level of solution quality, we use a local optimization selection method (Zeng et al. 2004), which is given by the two following steps:

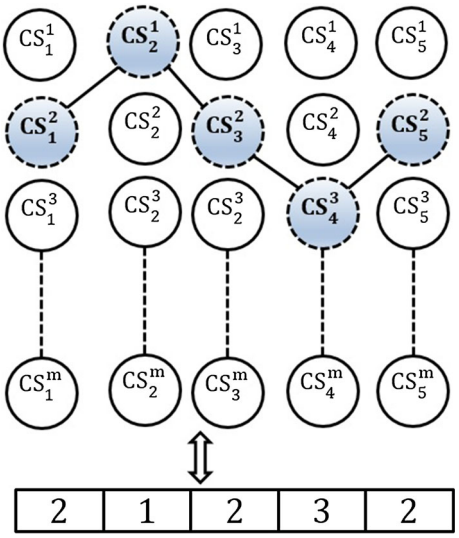

Fig. 3 Array integer encoding of solution

Step 1. Calculate the local score of each concrete cloud service CS_i^j, using the SAW method as follows:

$$Score_l(CS_i^j) = \sum_{q_t \in QoS^-} \frac{q_{t,i}^{max} - q_{t,i}^{j}}{q_{t,i}^{max} - q_{t,i}^{min}} * w_t + \sum_{q_t \in QoS^+} \frac{q_{t,i}^{j} - q_{t,i}^{min}}{q_{t,i}^{max} - q_{t,i}^{min}} * w_t, \quad (6)$$

where $q_{t,i}^{j}$ denotes the tth criterion value of the jth cloud service CS_i^j from the ith candidate set S_i, $q_{t,i}^{max}$ and $q_{t,i}^{min}$ denote the maximum and minimum values of the tth attribute in the set S_i, and w_t represents the user's weight of the tth QoS attribute.

Step 2. For each set of candidate cloud services S_i corresponds to the position i of what an individual ind does, do

1. Use a binary tournament selection Goldberg (1989) to select two different concrete cloud services CS_i^j and CS_i^k at random from the set S_i.
2. If $Score_l(CS_i^j) > Score_l(CS_i^k)$,Affect the ith position of ind with the index value j of the cloud service CS_i^j
3. Else Affect the ith position of ind with the index value k of the cloud service CS_i^k
4. End if

End For

Improved initial population generation

Using the above method of generating a solution, we initialize the population as follows:

Step 1. Generate an improved solution using the *local optimization selection method*.
Step 2. Let iteration $itr = 1$, perform the following steps until $itr = PopSize$ ($PopSize$: is the population size.)
Step 3. Generate a new improved solution using the *local optimization selection method*. If the newly-generated solution is not the same as any solution in the current population, insert it into the population and let $itr = itr + 1$; otherwise, discard it.
Step 4. Go back to step 3.

An illustrative example will help us understand and demonstrate the effectiveness of the aforesaid initial generating population process. Let's consider, for instance, a sequential abstract cloud service composition $ACSC$ of three atomic tasks, where each task $T_{i(i=1...3)}$ is related to the candidate cloud service S_i of three concrete cloud services $CS_i^{j\,(j=1...3)}$. Each concrete cloud service CS_i^j has two QoS-criteria : cost ($) and response time (ms), assigned by the QoS-weights $w_{price} = 0.5$ and $w_{time} = 0.5$, respectively. QoS values of theses concrete cloud services are shown in Table 2. Using the formula given in Eq. 6, we can calculate the local score of each CS_i^j, for example, $Score_l(CS_2^1) = \frac{8-2}{8-2} * 0.5 + \frac{200-180}{200-150} * 0.5 = 0.7$. Table 3 presents the evaluated local score values of theses cloud services. By using our proposed local optimization selection method, It is obvious that, for the candidate set S_3, the concrete service CS_3^2 with the highest local score is more likely to be selected in the generated individual when the selection's probability of the concrete service CS_3^3 with the lowest local score, is zero(i.e. CS_3^3 has never been selected in individual's generation by applying the tournament selection method in the candidate set S_3).

In the presented example, the total number of possible combinations to generate individuals is 3^3, implying that we have 27 different solutions. To show the effectiveness

Table 2 An example of nine cloud services assessed on two criteria

QoS attribue values : $CS_i^j(cost, time)$		
S_1	S_2	S_3
$CS_1^1(2, 220)$	$CS_2^1(2, 180)$	$CS_3^1(4, 140)$
$CS_1^2(3, 190)$	$CS_2^2(2, 200)$	$CS_3^2(1, 150)$
$CS_1^3(5, 180)$	$CS_2^3(8, 150)$	$CS_3^3(3, 170)$

Table 3 Local score values of the nine cloud services

Local score values: $CS_i^j\{Score_l(CS_i^j)\}$		
S_1	S_2	S_3
$CS_1^1\{0.50\}$	$CS_2^1\{0.70\}$	$CS_3^1\{0.50\}$
$CS_1^2\{0.71\}$	$CS_2^2\{0.50\}$	$CS_3^2\{0.84\}$
$CS_1^3\{0.50\}$	$CS_2^3\{0.50\}$	$CS_3^3\{0.17\}$

of the proposed improved initial population generation, two populations of 7 solutions are generated using our heuristic generation method and a random approach (denoted as Imp_{pop} and Rnd_{pop}, respectively). A population with a high level of solution quality (Qua_{pop}) and diversity (Div_{pop}) is more important to get a near-optimal global solution. The following pseudo-code is used to determine the Qua_{pop} and Div_{pop} values of each generated population, in which the function $GeneratePop()$ simulates our proposed generation approach to create Imp_{pop} or a random method to generate Rnd_{pop}.

1. **For** i=1 to 1000 % experiment repeats 1000 times.
2. $Pop = GeneratePop()$ % generates a population (Pop).
3. $S(Pop) = calculateScore(Pop)$ % calculates the score values of the individuals in (Pop) using Eq. 3.
4. $AS_i = average(S(Pop))$ % calculates the scores' average value of the individuals in Pop and store it in AS_i.
5. $D(Pop) = calculateDiv(Pop)$ % calculates the diversity values of the individuals in Pop as we will illustrate in Sect. "Selection operator".
6. $AD_i = average(D(Pop))$ % calculates the diversities' average value of the individuals in Pop and store it in AD_i.
7. **End For**
8. $Qua_{pop} = \frac{\sum_{i=1}^{1000}(AS_i)}{1000}$
9. $Div_{pop} = \frac{\sum_{i=1}^{1000}(AD_i)}{1000}$

Table 4 lists the result of the above code applied in the current example (Exp_{data}) and another example (Rnd_{data}). In Rnd_{data}: we have a sequential $ACSC$ with 15 abstract cloud services, and for each abstract service, 500 concrete

Table 4 Diversity and quality solution results of the generated populations for the two examples: Exp_{data} and Rnd_{data}

	Exp_{data}		Rnd_{data}	
	Qua_{pop}	Div_{pop}	Qua_{pop}	Div_{pop}
Imp_{pop}	0.6403	0.2599	0.6171	0.6649
Rnd_{pop}	0.5561	0.2854	0.4994	0.6654

services are created, where the cost and the time QoS values of theses cloud services were randomly generated in the intervals [2, 15]($), [20, 1500] (ms), respectively. The number of generated solutions for populations Imp_{pop} and Rnd_{pop} in Rnd_{data} example is set to 10. The results shown in Table. 4 reflect that our approach in generating a powerful population is significantly better than the random generation method (i.e. the quality solution of Imp_{pop} is better than Rnd_{pop}'s quality solution), and this process of population generation has a great impact on the coverage and the diversity of solutions when the optimization problem has a high number of objectives in a huge search space (i.e. in Rnd_{data} example, Imp_{pop} and Div_{pop} have the same diversity values).

Fitness evaluation

A fitness value is assigned for each individual in the generated population of the proposed HGA, which measures the individual's performance in the research space. As discussed in the problem description section, the QoS-CSC aims to find a solution with a maximum score value of the global QoS and meets the global QoS constraints (i.e. Highest feasible solution where all the QoS constraints are satisfied), but some of the generated individuals may be infeasible solutions (i.e. individuals with some or all global QoS constraints are unverified), to distinguish between each infeasible and feasible solution, a penalty is given to the fitness of an infeasible solution. The employed penalty function in our study is shown in Eq. 7 and the fitness function for each individual is given in Eq. 9.

$$Pn(ind) = \begin{cases} 0 & \text{satisfied Csts} \\ \sum_{t=1}^{k} CstV_t(ind)^{\alpha} * p_t & \text{unsatisfied Csts,} \end{cases} \quad (7)$$

where α is the severity of the penalty (here,$\alpha = 2$), $p_{t(t=1\ldots k)} \in [0, 1]$ is the penalty factor with $\sum_{t=1}^{k} p_t = 1$ and $CstV_t(ind)$ is the constraint violation value for the attribute q_t which is defined in Eq. 8.

$$CstV_t(ind) = \begin{cases} \frac{\max(0, Cst_t - agg(q_t))}{Cst_t} & \text{if } q_t \in QoS^{+} \\ \frac{\max(0, agg(q_t) - Cst_t)}{Cst_t} & \text{if } q_t \in QoS^{-}, \end{cases} \quad (8)$$

where $Cst_t \in GCst$, $agg(q_t)$ denotes the aggregated value of the tth QoS criterion of the individual ind, and Cst_t and $agg(q_t)$ refer to negative or positive constraints formulas $agg(q_t) \leq Cst_t$,$agg(q_t) \geq Cst_t$ given in Eq. 5, respectively.

$$fit(ind) = \begin{cases} 0.5 + Score(ind) * 0.5 & \text{if } ind \text{ is feasible} \\ Score(ind) * 0.5 - Pn(ind) & \text{if } ind \text{ is infeasible,} \end{cases} \quad (9)$$

where $Score(ind)$, is the score value of the individual ind given in Eq. 3 and $Pn(ind)$ is the penalty value of this individual.

It can be seen from Eq. 9 that fitness values of infeasible solutions are between 0 and 0.5, while those of feasible ones are between 0.5 and 1. Thus, it can be assured that a feasible solution has more fitness value than any infeasible solution.

Genetic phase (global exploration)

Selection operator

Roulette wheel selection is an often selection operator used in several proposed genetic algorithms tackling the QoS-CSC (Canfora et al. 2005; Yue and Chengwen 2008; Maolin and feng 2010; Wang et al. 2015), the basic strategy of its selection is: the better fitted an individual is, and the larger the probability of its survival and mating is Goldberg (1989). By this strategy of selection, stronger (more fitted) individuals dominate weaker individuals in the population, which causes the loss of population diversity; as a result, it will engender the premature convergence phenomena. To overcome this problem, we propose a novel roulette wheel selection based on the selection's probability related to the fitness of the chromosomes and its diversities at the same time, where fitted and diversified individuals are typically more likely to be selected for reproduction. In order to calculate the new selection's scores for individuals, we evaluate the diversities between each individual and the rest of the individuals within the population. In this study we use the Hamming distance to define the distance between two individuals ind_j and ind_k, which is given as follow:

$$dist(ind_j, ind_k) = \sum_{i=1}^{n} y_i, \quad (10)$$

where $y_i = \begin{cases} 1 & \text{if } ind_j(i) \neq ind_k(i) \\ 0 & \text{if } ind_j(i) = ind_k(i) \end{cases}$, and $ind_j(i)$ is the value in the ith position of the individual ind_j. The diversity $div(ind_j)$ between each individual ind_j and the rest of the individuals of the population is given in Eq. 11, and its new selection's score $Scr_{sel}(ind_j)$ is given in Eq. 12, which is calculated using the above mentioned simple additive weighting technique SAW (i.e. SAW is used in order to map the fitness and the diversity values of each individual into a single real value).

$$div(ind_j) = \sum_{h=1}^{PopSize} dist(ind_j, ind_h), \tag{11}$$

where $j \neq h$ and $PopSize$ is the population size

$$Scr_{sel}(ind_j) = N_f(ind_j) * w_f + N_d(ind_j) * w_d \tag{12}$$

where

$$N_f(ind_j) = \begin{cases} \frac{fit(ind_j) - \min(fit_{Pop})}{\max(fit_{Pop}) - \min(fit_{Pop})} & \text{if } \max(fit_{Pop}) \neq \min(fit_{Pop}) \\ 1 & \text{if } \max(fit_{Pop}) = \min(fit_{Pop}) \end{cases} \tag{13}$$

$$N_d(ind_j) = \begin{cases} \frac{div(ind_j) - \min(div_{Pop})}{\max(div_{Pop}) - \min(div_{Pop})} & \text{if } \max(div_{Pop}) \neq \min(div_{Pop}) \\ 1 & \text{if } \max(div_{Pop}) = \min(div_{Pop}) \end{cases} \tag{14}$$

in Eqs. 12, 13 and 14 :
$fit(ind_j)$, is the fitness value of the individual ind_j shown in Eq. 9.
$div(ind_j)$, is the diversity value of the individual ind_j.
$\max(fit_{Pop})$ and $\min(fit_{Pop})$, denote the maximum and minimum fitness values, respectively, in the current population Pop.
$\max(div_{Pop})$ and $\min(div_{Pop})$, denote the maximum and minimum diversity values, respectively in the current population Pop.
w_f and w_d, are the weighting factors for controlling the weights of fitness and diversity values, respectively, in the current population, which are adaptively calculated using the following equations:

$$w_f = 0.5 + \frac{counter}{2 * max_{itr}} \tag{15}$$

$$w_d = 0.5 - \frac{counter}{2 * max_{itr}}, \tag{16}$$

where $counter$ and max_{itr} are the current iteration value and the maximum number of iterations of the proposed algorithm, respectively. To select an individual for a new population (Pop_{new}), we use the following steps:

Step 1. Generate a random real number $(r) \in [0, 1]$.
Step 2. If $r \leq p_1$, select the first individual ind_1, otherwise; select the ith individual $ind_{i(i=1...PopSize)}$, such that $p_{i-1} < r \leq p_i$, where p_i is the cumulative probability for each individual ind_i in the current population (Pop) which is calculated as defined in Eq. 17.

$$\begin{cases} p_1 = \frac{Scr_{sel}(ind_1)}{\sum_{i=1}^{PopSize} Scr_{sel}(ind_i)} \\ p_i = p_{i-1} + \frac{Scr_{sel}(ind_i)}{\sum_{j=1}^{PopSize} Scr_{sel}(ind_j)} \quad \text{if}, 2 \leq i \leq PopSize \end{cases} \tag{17}$$

Crossover operator

After selecting two individuals (parents) in the current population using the above selection operator, a crossover operator produces two new individuals (children) by combining the genes of the two selected parents. The most using crossover operators for QoS-CSC are a one-point crossover (Wang et al. 2015), or a two-point crossover (Canfora et al. 2005; Wu et al. 2014; Jin et al. 2015). In the one-point crossover, a cutting point is randomly selected in two individuals. After that, the genes from the cutting point to the end of the individuals are swapped between the two chromosomes (see Fig. 4a). In the two-point crossover, two crossover locations are randomly selected in two individuals. After that, the subsequences of genes, between the two locations of each individual are exchanged from a chromosome to another one (see Fig. 4b). In this study, a random real number $r \in [0, 1]$ is generated, if $r \leq 0.5$, the one-point crossover is used for a reproduction; otherwise, the standard two-point crossover is employed in our proposed algorithm for evolution.

Mutation operator

In order to prevent a potential stagnation of the population in a local optimum, we apply the mutation operator to maintain the diversity of the population, and to achieve a better exploration in the research space by making a slow and small genetic frequency change in the generated chromosomes. The mutation operator used in this study selects a position (i.e. an abstract cloud service) in the generated individual (i.e. individual obtained by crossover) and randomly replaces it with a new chosen concrete cloud service in the corresponding candidate set of cloud services (see Fig. 4c)

FOA phase (local exploitation)

The genetic phase described above make more attention to the global exploration, to reduce the computation time and to maintain a balance between the exploration and the exploitation abilities of the proposed HGA. As inspired by the onlooker bee phase of the artificial bee colony algorithm (ABC) (Karaboga and Basturk 2007), the FOA scheme is performed on the better solutions generated after the genetic evolution. In the smell-based foraging process, SN neighbors are generated around each selected solution; to be specific, the new neighbor is generated by applying the mutation operator discussed in Sect. "Mutation operator" for L posi-

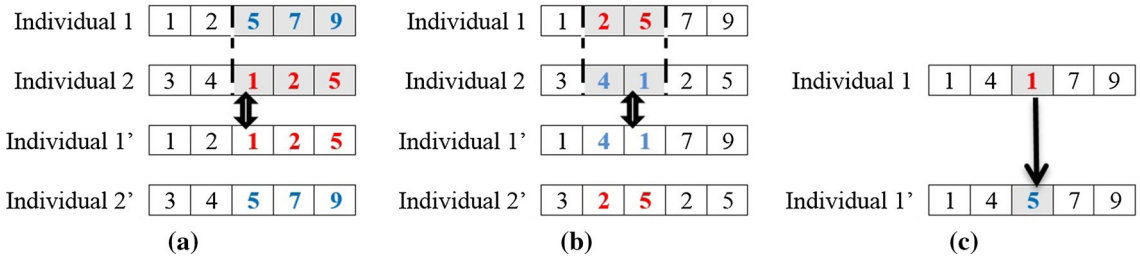

Fig. 4 Crossover and mutation operators in genetic phase. **a** One-point crossover, **b** two-point crossover, **c** mutation operation

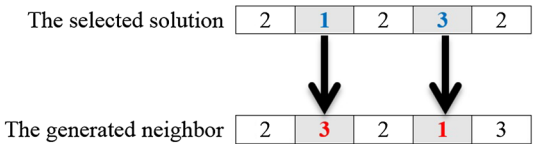

Fig. 5 Neighborhood strategy in the smell-based search process

tions that are randomly selected from the chosen solution. Consider the solution in Fig. 3, for example, the generated neighbor is illustrated in Fig. 5, where $l = 2$. It can be seen that the second and the forth positions (CS_2^1, CS_4^3) are replaced by the concrete cloud services (CS_2^3, CS_4^1). In the vision-based search processes, the SN neighbors of the selected individual (ind) are evaluated and the best one $(BestInd)$ of the neighbors is returned; if $BestInd$ is better than ind, then the individual ind is replaced with $BestInd$. Otherwise, it is maintained. The detailed steps for the FOA process applied on each generated solution by the genetic phase are given as follows:

Step 1. For each individual ind_i generated after the genetic phase, where i is the population size $(PopSize)$, perform the following steps.
Step 2. Calculate the selecting probability ps of ind_i by using Eq. 18 (Karaboga and Basturk 2007).

$$ps = \frac{fit(ind_i)}{\sum_{j=1}^{PopSize} fit(ind_j)}, \tag{18}$$

where $fit(ind_i)$ is the fitness value of the individual ind_i given in Eq. 9.

Step 3. Generate a random number $r \in [0, 1]$
Step 4. If r is smaller than ps, SN neighbors are generated around ind_i to construct a sub-population $Subp_i$ .Then ind_i is replaced with the best individual $BestInd$ in the corresponding sub-population $Subp_i$ (i.e. if $BestInd$ has bigger fitness than ind_i); otherwise, ind_i remains unchanged.

The elitism operator

In order to replace the old population (Pop) with the new population (Pop_{new}), the elitism strategy is used as a replacement one. In our proposed algorithm, we replace the worst solution of population Pop_{new} with the best solution of population Pop .This strategy of preserving and replacing, speed up the convergence of the algorithm and prevent the loss of the best individuals during the evolutionary process.

The stopping criterion

There are several stopping criteria to terminate a metaheuristic algorithm such as:

1. Reaching a specific CPU time;
2. Fixing a large number $MaxItr$ as maximum number of iterations;
3. Converging to a specific value of fitness function;
4. and letting max_gen_imp is a fixed number of generations. If the solution Sol_{Best} with the highest fitness value does not improve over the max_gen_imp generations, the algorithm stops and returns the near-optimal solution Sol_{Best}.

In our experiments we use the second or/and the fourth criteria to stop the proposed HGA.

The framework of the proposed HGA

The following is the detailed steps of the proposed HGA:

Step 1. Initialization phase.

1. Set the system parameters: Population size $(PopSize)$, Probability of Crossover(P_c), Probability of Mutation (P_m) , neighbors' size (SN) and the number of the selected positions (L) in the smell-based search process, and the maximum number of iteration $(MaxItr)$.
2. Generate the initial population (Pop) as discussed in Sect. "Improved initial population generation".

3. Calculate the fitness value of each solution in the initial population (Pop).

Step 2. If the stopping criterion is satisfied, stop the HGA and return the best solution; otherwise, perform steps 3 to 6.

Step 3. Genetic phase. Construct the new population (Pop_{new}) using the population(Pop) as follows:

1. In order to select two new individuals (pair of parents), apply the selection operator as discussed in Sect. "Selection operator".
2. Apply the crossover operator to the selected pairs of parents with crossover probability of (P_c) as discussed in Sect. "Crossover operator".
3. Apply the mutation operator to each individual obtained by crossover with mutation probability of (P_m) as discussed in Sect. "Mutation operator".

Step 4. FOA phase. Perform the local exploitation in the new population (Pop_{new}) as discussed in Sect. "FOA phase (Local exploitation)".

Step 5. Apply the elitism operator as discussed in Sect. "The elitism operator", and then replace (Pop) with(Pop_{new}).

Step 6. Go back to step 2.

Experiments

To verify the efficiency and effectiveness of our proposed HGA, a sequential abstract cloud service composition $ACSC$ with (n) abstract cloud services is simulated and for each abstract cloud service, the corresponding candidate set of cloud services contains (m) concrete cloud services. For simplicity, we assume that we have four QoS-criteria for each concrete cloud service, two are negatives (response time and price) and the others are positives (availability and reliability) and their values were generated randomly. The random values of response time, price, availability and reliability in each set of candidate cloud services were generated in the intervals [20, 1500] (ms), [2, 15] %, [0.95, 1] $\%and$ [0.4, 1] %, respectively. The weights of these QoS criteria were set to the same as $W\{(w_{time} = 0.25), (w_{price} = 0.25), (w_{ava} = 0.25)$ and $(w_{rel} = 0.25)\}$, and for positive and negative QoS-parameters, their QoS constraints can be calculated as the same in the paper (Wu and Zhu 2013) via the strength of user's end-to-end global QoS constraints (φ) by Eqs. 19 and 20, respectively.

$$\varphi * (agg(q_t^{max}) - agg(q_t^{min})) + agg(q_t^{min}) \quad (19)$$

$$agg(q_t^{max}) - \varphi * (agg(q_t^{max}) - agg(q_t^{min})) \quad (20)$$

where, the percentage (φ) is used to represent the strength of user's end-to-end global QoS constraints. $agg(q_t^{max})$ and $agg(q_t^{min})$ denote the maximum and minimum possible aggregated values of the tth QoS criterion of $ACSC$, respectively.

The proposed algorithm HGA and the compared algorithms are coded on MATLAB R2013a, and we run them on a computer with 2.6 GHz and 2 GB of RAM under windows 7(32 bit) system. Each simulation on the following runs 20 times and the results are averaged.

Parameter setting of HGA

The proposed HGA contains six key parameters: Population size ($PopSize$), Probability of crossover (P_c), Probability of mutation (P_m) , neighbors size (SN) and the number of the selected positions (L) in the smell-based search process, and the maximum number of iteration ($MaxItr$). Among these, MaxItr is set to 1000. In order to investigate the influence of the rest of parameters on the performance of the proposed algorithm, the Taguchi method of design of experiment (DOE) Montgomery (2005) is utilized, the five parameters $PopSize$, P_c ,P_m, SN and L of HGA are tuned in a random dataset, where the number of abstract cloud services (n) is set to 15, and the related number of concrete cloud services (m) for each abstract cloud service is set to 300 with the percentage (φ) of end-to-end global QoS constraints set to 0.4. The levels of these five parameters are given in Table 5 in which we use four levels for each factor, so an orthogonal array $l_{16}(4^5)$ is used to calibrate these parameters. To analyze the significance rank of each parameter, the proposed algorithm is run 20 times independently for each combination of factors values. The levels' values of each factor and the average optimal fitness value ($AOFV$: which is regarded as the response value variable) are given in Table 6, where the highest AOFV value is, the better the parameter combination values is; and by using the statistical analysis tool Minitab, the trend of each factor level is shown in Table 7 and Fig. 6, respectively.

It can be observed from Table 7 and Fig. 6 that $PopSize$ and P_c are the most significant parameters of the proposed algorithm. Large values of $PopSize$ and P_c are very helpful for global exploration. Consequently, the problem's solution is attained with a good quality. As for SN and L , it can be

Table 5 The parameter levels of HGA

Level	Parameter				
	P_c	P_m	$PopSize$	SN	L
1	0.75	0.15	20	1	1
2	0.80	0.20	40	3	2
3	0.85	0.25	70	5	3
4	0.90	0.30	100	7	5

Table 6 Orthogonal array and AOFV values

Expt number	Factor					AOFV
	P_c	P_m	$PopSize$	SN	L	
1	0.75	0.15	20	1	1	0.52272
2	0.75	0.20	40	3	2	0.72736
3	0.75	0.25	70	5	3	0.77175
4	0.75	0.30	100	7	5	0.79479
5	0.80	0.15	40	5	5	0.74997
6	0.80	0.20	20	7	3	0.61356
7	0.80	0.25	100	1	2	0.79516
8	0.80	0.30	70	3	1	0.74947
9	0.85	0.15	70	7	2	0.79556
10	0.85	0.20	100	5	1	0.79481
11	0.85	0.25	20	3	5	0.56837
12	0.85	0.30	40	1	3	0.75003
13	0.90	0.15	100	3	3	0.79504
14	0.90	0.20	70	1	5	0.79553
15	0.90	0.25	40	7	1	0.77321
16	0.90	0.30	20	5	2	0.63630

Table 7 Response value (AOFV)

Level	P_c	P_m	$PopSize$	SN	L
1	0.7042	0.7158	0.5852	0.7159	0.7101
2	0.7270	0.7328	0.7501	0.7101	0.7386
3	0.7272	0.7271	0.7781	0.7382	0.7326
4	0.7500	0.7326	0.7950	0.7443	0.7272
Delta	0.0459	0.0170	0.2097	0.0342	0.0285
Rank	2	5	1	3	4

shown from Fig. 6 that they have small behavior effect on the proposed algorithm. If SN is very large value, it means that the exploitation is over-focused. As a result, we'll get more computation time on the proposed algorithm with a little improvement of the solution quality, because there is a number of SN evaluation times for each selected individual in the local search process; a medium value of SN is recommended. As for the number of the individual's modified positions L in the smell-based search process; if it is with a large value, it means that high modification is applied in the selected individual causing no improvement in the quality of solution. So, in order to enhance the solution quality, a slight modification in the chosen individual is preferred. As for a probability of mutation P_m, it can be seen from Table 7 that it has a little influence on the proposed algorithm; so in our HGA, its value is set to 0.2. According to the above analysis, the optimal values of theses parameters are set as $P_c = 0.9$, $P_m = 0.20$, $PopSize = 70$, $SN = 5$ and $L = 2$. These parameter values will be used for the proposed HGA in the following simulation and comparison tests.

Comparisons and results

In this subsection, to validate the performance of our proposed algorithm HGA, we compared it with traditional genetic algorithm (TGA) (Canfora et al. 2005), simple fruit fly optimization algorithm (SFOA) (Pan 2012) and the Discrete Gbest-guided Artificial Bee Colony algorithm (DGABC) proposed by Huo et al. (2015), where their experiment results show that DGABC can obtain a better solution within a short period of time than GA, PSO and deferential evolution (DE) algorithm, especially for large scale data. Table 8 details the initial parameters setting values of the compared algorithms, which are used in the rest of the simulations, and the following metrics are used to compare the performance of our HGA:

- *Optimality*: this metric describes the solution's fitness value of the optimal composite cloud services generated by each algorithm.
- *Execution time*: this metric represents the required time to find the optimal composite cloud services for each algorithm.
- *Feasibility rate*: Measures the success rate to obtain feasible solutions for user's requests, which is calculated using the formula given in Eq. 21.

$$FR = \frac{N_{fs}}{N_{rt}}, \tag{21}$$

where N_{rt} is the number of execution times and N_{fs} is the number of feasible solutions found in N_{rt} execution times for each algorithm.

Optimality and execution time

In order to evaluate the optimality and the computation time of our proposed algorithm, two scenarios are considered. The first is the number of abstract services $n = 15$, and the number of related concrete services m varies from 50 to 500 with an increment of 50, and the percentage φ of QoS constraints is set to 0.4. The second is $m = 200$ and n varies from 7 to 25 with an increment of 2, and also the percentage φ is set to 0.4. The stopping criterion for all compared algorithms in these two scenarios is that the best fitness value remains unchanged over the last 50 iterations, or the maximum number of iterations is reached (i.e. $max\_gen\_imp = 50$ or $MaxItr = 1000$).

As observed in Fig. 7 that by varying the number of concrete cloud services m, the fitness value (optimality)

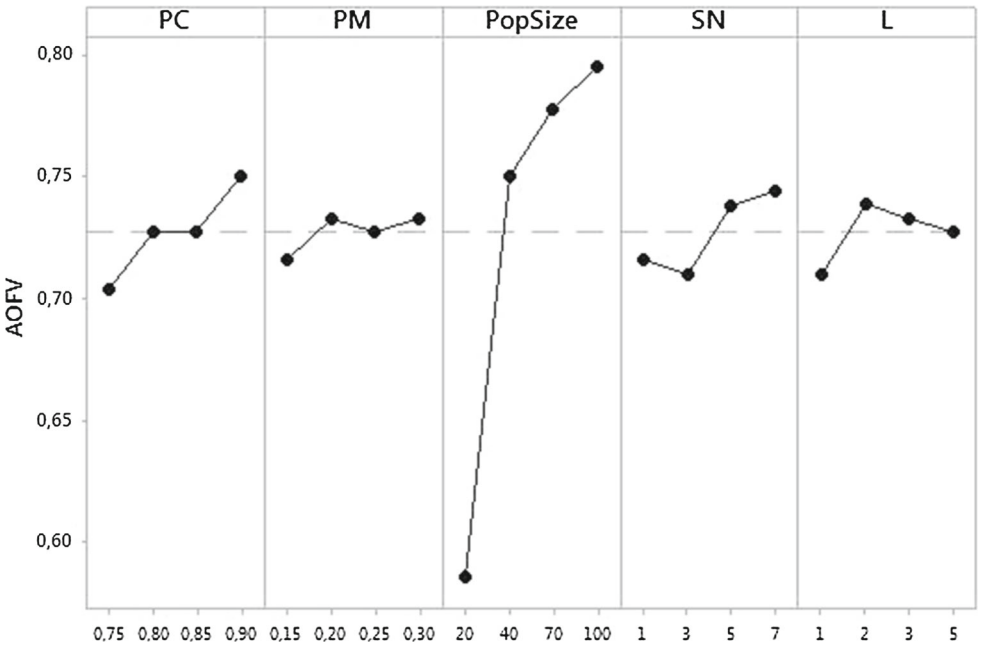

Fig. 6 Factor level trend of HGA

Table 8 Algorithms parameter settings

Algorithms	Parameter settings values					
	$PopSize$	P_c	P_m	SN	L	$Limit$
DGABC (Huo et al. 2015)	70	–	–	–	–	100
TGA (Canfora et al. 2005)	70	0.9	0.2	–	–	–
SFOA (Pan 2012)	70	–	–	–	2	–
HGA	70	0.9	0.2	5	2	–

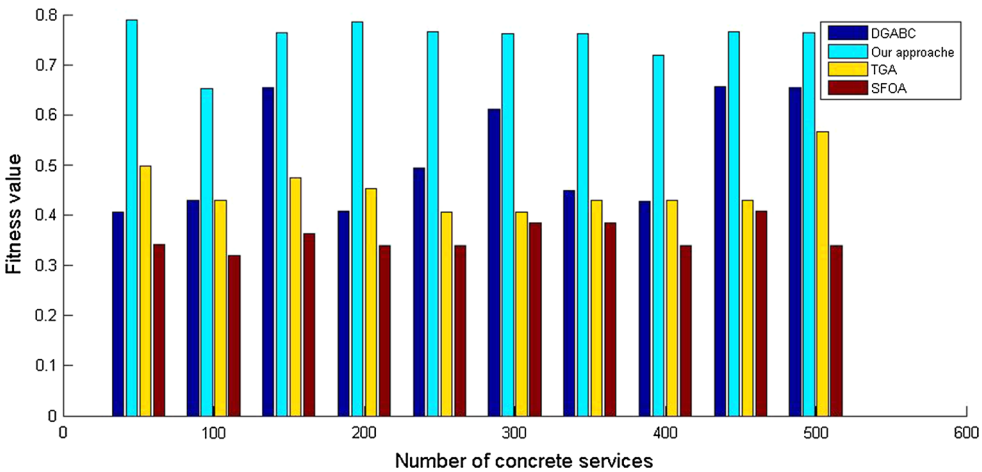

Fig. 7 Optimality comparison (scenario 1: the number of abstract srvices $n = 15$ and the number of concrete services m varies from 50 to 500 with an increment of 50)

of our approach is better than the other compared algorithms. We can see also from Fig. 8, that with the increasing number of abstract cloud services, there are eight cases in which our approach is as optimal as that of DGABC. The corresponding number of the abstract cloud services is (9,13,15,17,19,21,23,25), and the average QoS of HGA is the same as that DGABC's average QoS in the other two cases. However, our HGA is by all means better than TGA and SFOA. It can be concluded that our algorithm performs better than the other compared algorithms do, in terms of optimality (average QoS of composite cloud services).

Figures 9 and 10, display the comparison of computation time for the four algorithms with the increasing number of concrete cloud services m and the number of abstract cloud services n, respectively. As shown in Fig. 9, it can be seen that our algorithm is faster than DGABC, and the SFOA

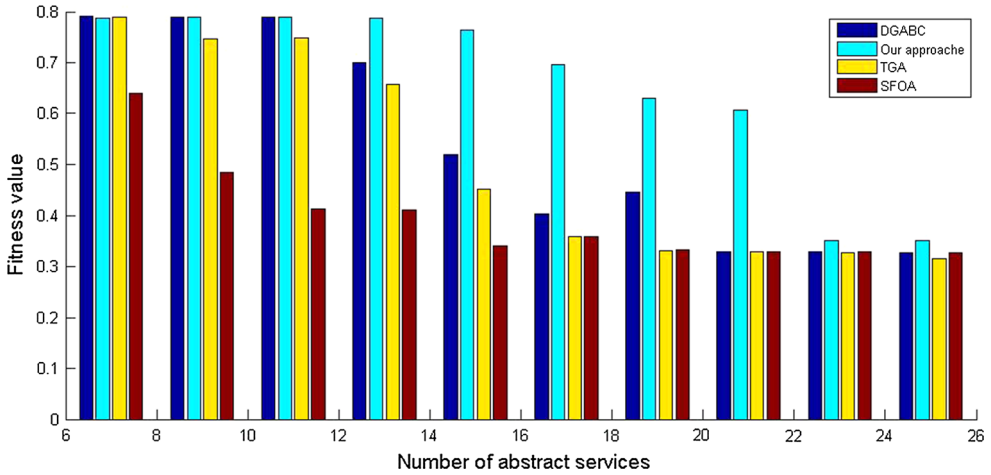

Fig. 8 Optimality comparison (scenario 2: the number of concrete services $m = 200$ and the number of abstract services n varies from 7 to 25 with an increment of 2)

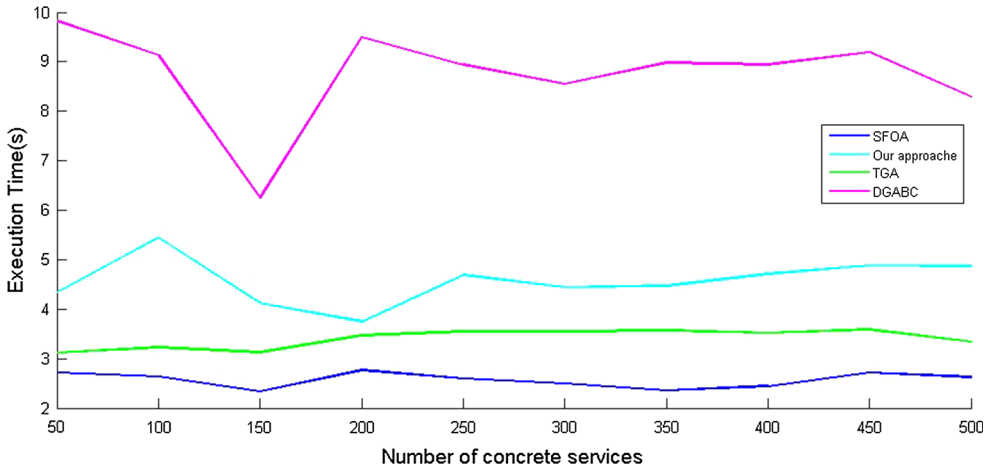

Fig. 9 Computation time comparison (scenario 1: the number of abstract services $n = 15$ and the number of concrete services m varies from 50 to 500 with an increment of 50)

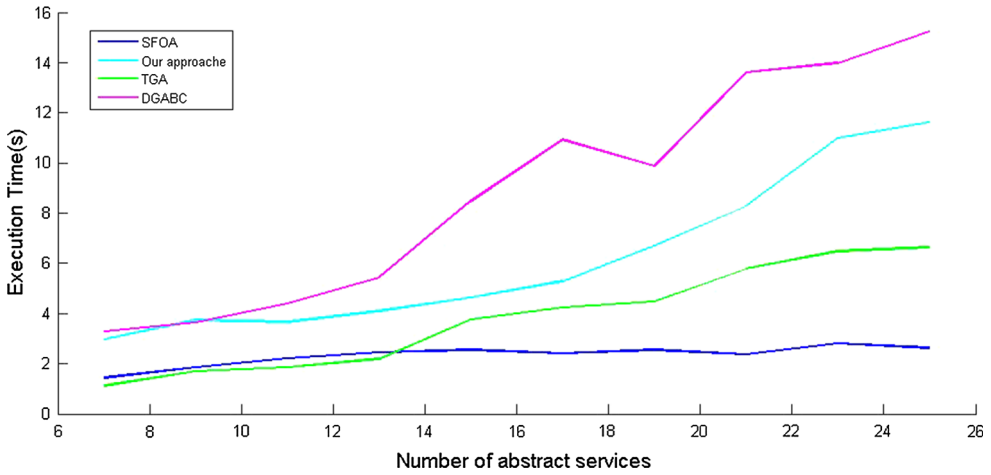

Fig. 10 Computation time comparison (scenario 2: the number of concrete services $m = 200$ and the number of abstract services n varies from 7 to 25 with an increment of 2)

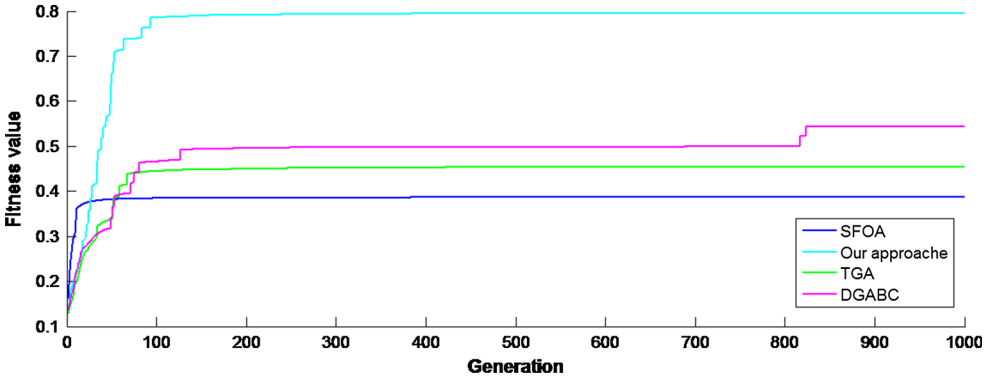

Fig. 11 The convergence curves of the quality of the optimal composite service obtained by the compared algorithms.

requires the least running time with a very low fitness values compared to our algorithm. TGA is a little faster than HGA; however, the HGA can find better solutions than TGA. Moreover, in Fig. 10, HGA is also faster than DGABC, and its execution time slightly increases with the abstract cloud service's growth. Due to the stagnation of SFOA and TGA in local optimum, and especially where n the large value would be, it is shown that their running times are the lowest compared to HGA and DGABC. Thus, these results can prove that HGA is better than DGABC in terms of its execution time, and it has a good tradeoff between the execution time and the optimality compared to the other two algorithms: TGA and SFOA. In order to verify how the compared algorithms find the near-optimal solutions (i.e. the convergence curve), the experiment shown in Fig. 11 presents the convergence curves of the fitness values obtained by these algorithms, where the number of abstract services n is set to 15. The number of candidate services m is set to 200, the percentage φ of QoS constraints is set to 0.4, and the stopping criterion is $MaxItr = 1000$. It can be observed from Fig. 11 that HGA reaches the global optimal fitness value faster than DGABC and TGA; it is shown from Fig. 11 that the solutions found by HGA are worse than the solutions found by SFOA when the number of generations is less than 25. However, when the number of generations is larger than 25, the solutions found by HGA are better than the solutions found by SFOA; the latter converges more quickly to the local optimal in least number of generations. It can be seen also from Fig. 11 that the global searching ability of our HGA largely stronger than the other algorithms are. The best fitness value found by our approach is 0.79513; whereas, the best solutions found by DGABC, TGA and SFOA are (0.54418, 0.45459, 0.3869), respectively.

As far as the optimality, the execution time and the convergence speed are concerned, it can be concluded that HGA outperforms the other three compared algorithms for solving the QoS-aware cloud service composition problem with different scales.

Table 9 Comparison results of the feasibility rate (FR) values for each QoS constraints (φ)

(φ)	TGA (%)	SFOA (%)	DGABC (%)	HGA (%)
$\varphi = 0.2$	**100**	**100**	**100**	**100**
$\varphi = 0.3$	99	82	**100**	**100**
$\varphi = 0.4$	4	0	3	**73**
$\varphi = 0.5$	0	0	0	**1**
$\varphi = 0.6$	**0**	**0**	**0**	**0**
Average	40.60	36.40	40.60	**54.60**

The best percentages are in bold

Feasibility rate

This experiment is to measure the success rate to obtain a feasible solution for the compared algorithms including TGA, SFOA, DGABC and our hybrid genetic algorithm HGA by varying the strength of QoS constraints φ from 0.2 to 0.6 with an increment of 0.1. Table. 9 lists the comparison results of the feasibility rate (FR) values obtained by each compared algorithm for each considered QoS constraints φ. In this experiment, the number of abstract services n is set to 17, the number of candidate cloud services m is set to 400, the stopping criterion is $MaxItr = 1000$, and each experiment is repeated 100 times (i.e.$N_{rt} = 100$). There are five columns in the table. The first column presents the corresponding strength of QoS constraints φ. The following columns present the feasibility rate FR values obtained by each compared algorithm where FR is calculated, using Eq. 21. The last row in the table presents the average value of the FR values obtained by each compared algorithm for all of the five QoS constraints (φ) cases. It can be observed from Table. 9 that, with the increasing of φ values, the FR values for all compared algorithms decrease; this is because the probability to find a feasible solution satisfying all QoS constraints is declined with the increasing of the strength of QoS constraints φ. HGA obtained three best FR values out of five cases compared to TGA and SFOA and the two best

FR values out of five cases compared to DGABC. The proposed HGA obtained an average value with 54.60% for all of the five QoS constraints (φ) cases, which is better than the other compared algorithms. Considering these results, it can be concluded that our proposed algorithm is significantly better than the other three algorithms in terms of feasibility rate.

Effects of user QoS preferences

The weights of the four attributes used in the above simulations are all set to 0.25. In order to verify the effect of using different values of QoS weights on the proposed HGA, 78 vectors of QoS preferences are used to compare and analyze the superiority of the compared algorithms under different user QoS preferences. These vectors are shown in Table 10. In this experiment, the number of abstract services n is set to 15; the number of candidate cloud services m is set to 400 , the percentage φ of QoS constraints is set to 0.4, and the stopping criterion is when the number of iterations during the best solution doesn't improve (max_gen_imp) when being set to 50, or the maximum number of iteration ($MaxItr$) is reached (i.e. $max\_gen\_imp = 50$ or $MaxItr = 1000$). Table 11 lists the comparison results of the optimal fitness values obtained by each compared algorithm for each considered user QoS weights vector. There are five columns in the table. The first column presents the corresponding number of user QoS preference vector where their weight attribute values are shown in Table. 10. The following columns present the optimal fitness values of the compared algorithms for each user QoS preferences vector. The row (average row) in the table presents the average value of the optimal fitness values obtained by each compared algorithm for all QoS weight vectors. The last two rows (quality rate and weakness rate rows) in the table present the proportion in which each algorithm is better than the other algorithms, and the proportion, in which each algorithm is poorer than one or all the other algorithms, respectively. It can be seen from Table 11 that our proposed HGA has obtained sixty best fitness values out of seventy eight cases, which is better than the other three compared algorithms. HGA obtained an average value of 0.69706, which is better than DGBAC, TGA and SFOA where their average values are(0.57974, 0.36020, 0.39321), respectively. The quality rate of HGA is better than the other compared algorithms. Moreover, the weakness rate of our proposed algorithm is the least value compared to the other algorithms. Therefore, it can be concluded that our algorithm verifies the efficiency and the robustness for solving QoS-aware cloud service composition under different user QoS attribute weights.

Table 10 Vectors of QoS preferences

Vectors	Weights (w_{time}, w_{price}, w_{ava}, w_{rel})		
1	(0.1, 0.2, 0.3, 0.4)	40	(0.4, 0.2, 0.2, 0.2)
2	(0.1, 0.2, 0.4, 0.3)	41	(0.3, 0.3, 0.3, 0.1)
3	(0.1, 0.3, 0.2, 0.4)	42	(0.3, 0.3, 0.1, 0.3)
4	(0.1, 0.3, 0.4, 0.2)	43	(0.3, 0.1, 0.3, 0.3)
5	(0.1, 0.4, 0.2, 0.3)	44	(0.1, 0.3, 0.3, 0.3)
6	(0.1, 0.4, 0.3, 0.2)	45	(0.25, 0.25, 0.3, 0.2)
7	(0.2, 0.1, 0.3, 0.4)	46	(0.25, 0.25, 0.2, 0.3)
8	(0.2, 0.1, 0.4, 0.3)	47	(0.25, 0.3, 0.25, 0.2)
9	(0.2, 0.3, 0.1, 0.4)	48	(0.25, 0.2, 0.25, 0.3)
10	(0.2, 0.3, 0.4, 0.1)	49	(0.25, 0.3, 0.2, 0.25)
11	(0.2, 0.4, 0.1, 0.3)	50	(0.25, 0.2, 0.3, 0.25)
12	(0.2, 0.4, 0.3, 0.1)	51	(0.3, 0.2, 0.25, 0.25)
13	(0.3, 0.1, 0.2, 0.4)	52	(0.3, 0.25, 0.2, 0.25)
14	(0.3, 0.1, 0.4, 0.2)	53	(0.3, 0.25, 0.25, 0.2)
15	(0.3, 0.2, 0.1, 0.4)	54	(0.2, 0.3, 0.25, 0.25)
16	(0.3, 0.2, 0.4, 0.1)	55	(0.2, 0.25, 0.3, 0.25)
17	(0.3, 0.4, 0.1, 0.2)	56	(0.2, 0.25, 0.25, 0.3)
18	(0.3, 0.4, 0.2, 0.1)	57	(0.25, 0.25, 0.4, 0.1)
19	(0.4, 0.1, 0.2, 0.3)	58	(0.25, 0.25, 0.1, 0.4)
20	(0.4, 0.1, 0.3, 0.2)	59	(0.25, 0.4, 0.25, 0.1)
21	(0.4, 0.2, 0.1, 0.3)	60	(0.25, 0.1, 0.25, 0.4)
22	(0.4, 0.2, 0.3, 0.1)	61	(0.25, 0.1, 0.4, 0.25)
23	(0.4, 0.3, 0.1, 0.2)	62	(0.25, 0.4, 0.1, 0.25)
24	(0.4, 0.3, 0.2, 0.1)	63	(0.4, 0.1, 0.25, 0.25)
25	(0.4, 0.4, 0.1, 0.1)	64	(0.4, 0.25, 0.1, 0.25)
26	(0.4, 0.1, 0.4, 0.1)	65	(0.4, 0.25, 0.25, 0.1)
27	(0.4, 0.1, 0.1, 0.4)	66	(0.1, 0.4, 0.25, 0.25)
28	(0.1, 0.1, 0.4, 0.4)	67	(0.1, 0.25, 0.4, 0.25)
29	(0.1, 0.4, 0.1, 0.4)	68	(0.1, 0.25, 0.25, 0.4)
30	(0.1, 0.4, 0.4, 0.1)	69	(0.5, 0.25, 0.25, 0)
31	(0.2, 0.2, 0.3, 0.3)	70	(0.5, 0.25, 0, 0.25)
32	(0.2, 0.3, 0.2, 0.3)	71	(0.5, 0, 0.25, 0.25)
33	(0.2, 0.3, 0.3, 0.2)	72	(0, 0.25, 0.25, 0.5)
34	(0.3, 0.3, 0.2, 0.2)	73	(0, 0.25, 0.5, 0.25)
35	(0.3, 0.2, 0.3, 0.2)	74	(0, 0.5, 0.25, 0.25)
36	(0.3, 0.2, 0.2, 0.3)	75	(0, 0.3, 0, 0.7)
37	(0.2, 0.2, 0.2, 0.4)	76	(0, 0, 0.3, 0.7)
38	(0.2, 0.2, 0.4, 0.2)	77	(0.7, 0.3, 0, 0)
39	(0.2, 0.4, 0.2, 0.2)	78	(0.7, 0, 0, 0.3)

Effects of QoS value ranges

This experiment is to verify the performance of our HGA under different QoS value ranges. Table. 12 presents fifteen

Table 11 Comparison results of the optimal fitness values for each user QoS preferences vector

Vectors	DGABC	HGA	TGA	SFOA	Vectors	DGABC	HGA	TGA	SFOA
1	0.7665	**0.79012**	0.023338	0.4079	40	0.38116	**0.74132**	0.42608	0.35914
2	0.50257	**0.76683**	0.43391	0.37083	41	0.34448	0.34601	0.3424	**0.34614**
3	0.74423	**0.79127**	0.020589	0.38028	42	0.65146	**0.78739**	0.46758	0.35208
4	0.37225	**0.74518**	0.37084	0.35086	43	0.5414	**0.78977**	0.49778	0.36249
5	0.65342	**0.78807**	0.54081	0.33678	44	0.74616	**0.78836**	0.47679	0.34276
6	0.43364	**0.61019**	0.34315	0.3458	45	0.43234	**0.69747**	0.38699	0.36657
7	0.67698	**0.78966**	0.036936	0.33902	46	0.67429	**0.78811**	0.51766	0.35748
8	0.52462	**0.78849**	0.45675	0.34841	47	0.47331	**0.69818**	0.38318	0.38513
9	0.72111	**0.79096**	0.019469	0.44417	48	0.69976	0.78583	0.45073	**0.78892**
10	0.35372	**0.37503**	0.35193	0.35374	49	0.63121	**0.7667**	0.40282	0.3815
11	**0.79209**	0.78968	0.60719	0.37649	50	0.62928	**0.76237**	0.4756	0.34128
12	**0.3686**	0.34647	0.34464	0.34795	51	0.67597	**0.78813**	0.4282	0.40701
13	0.76703	**0.78968**	0.024147	0.35527	52	0.60916	**0.76502**	0.40211	0.38159
14	0.48065	**0.72372**	0.45893	0.37229	53	0.63055	**0.65405**	0.4751	0.36217
15	**0.79151**	0.78884	0.021651	0.44341	54	0.51767	**0.76554**	0.51985	0.36248
16	0.35062	0.35155	0.34925	**0.35244**	55	0.5206	**0.78935**	0.45467	0.34326
17	0.58289	**0.78922**	0.44654	0.35445	56	0.71976	**0.78738**	0.47316	0.33884
18	0.33977	**0.3617**	0.33735	0.33939	57	0.35127	0.35227	0.34244	**0.35238**
19	0.62963	**0.78877**	0.49216	0.35543	58	**0.79356**	0.79146	0.015459	0.51384
20	0.49893	**0.65518**	0.36276	0.34253	59	0.34291	**0.36496**	0.34059	0.34364
21	0.6974	**0.78833**	0.58356	0.37424	60	0.76606	**0.79164**	0.014413	0.40436
22	0.34346	**0.36569**	0.34015	0.34459	61	0.59025	**0.76856**	0.41434	0.3481
23	0.56064	**0.7195**	0.46866	0.33065	62	0.65145	**0.78876**	0.51442	0.37645
24	0.33877	0.33854	0.3368	**0.38385**	63	0.60851	**0.76698**	0.40445	0.33769
25	0.33252	**0.4472**2	0.35354	0.3325	64	0.60331	**0.7868**	0.49097	0.37552
26	0.35044	0.35044	0.34761	**0.35189**	65	0.34145	**0.38609**	0.33603	0.3418
27	0.76721	**0.79145**	0.022748	0.44232	66	0.58763	**0.74294**	0.36184	0.34136
28	0.70182	**0.791**	0.02859	0.39179	67	0.39243	**0.74342**	0.36917	0.35041
29	**0.794**	0.7901	0.019126	0.37552	68	**0.78991**	0.78868	0.028408	0.49801
30	0.35417	0.35474	0.35284	**0.35503**	69	**0.79876**	0.79398	0.79472	0.79842
31	0.54073	**0.76559**	0.476	0.34119	70	**0.79474**	0.79192	0.67814	0.44157
32	0.76715	**0.78863**	0.65614	0.33523	71	0.61161	**0.67802**	0.40545	0.33801
33	0.52247	**0.78654**	0.40754	0.34486	72	0.70089	**0.79507**	0.0030822	0.38241
34	0.44971	**0.76564**	0.4279	0.33695	73	0.35869	**0.59581**	0.39991	0.35879
35	0.54295	**0.74175**	0.47561	0.36588	74	0.41033	**0.74878**	0.45518	0.36643
36	0.63004	**0.78716**	0.4474	0.35757	75	**0.79886**	0.79824	0.0058625	0.58213
37	0.76337	**0.789**	0.025664	0.42452	76	**0.79876**	0.79754	0.0031458	0.54415
38	0.37212	**0.72288**	0.39344	0.37251	77	**0.79971**	0.79814	0.79795	0.79966
39	0.47304	**0.6968**	0.38204	0.33875	78	0.79702	**0.79786**	0.75097	0.67889
					Average	0.57974	**0.69706**	0.36020	0.39321
					Quality rate	14 %	**78 %**	0 %	8 %
					Weakness rate	86 %	**22 %**	100 %	92 %

The best values are in bold

Table 12 Sets of QoS value ranges

Sets	The ranges of QoS attributes			
	Time (ms)	Price ($)	Avail (%)	Relia (%)
1	[20, 50]	[2, 5]	[0.1, 1]	[0.9, 1]
2	[20, 2000]	[2, 5]	[0.9, 1]	[0.9, 1]
3	[20, 50]	[2, 70]	[0.9, 1]	[0.9, 1]
4	[20, 50]	[2, 5]	[0.9, 1]	[0.1, 1]
5	[20, 50]	[2, 5]	[0.1, 1]	[0.1, 1]
6	[20, 50]	[2, 70]	[0.9, 1]	[0.1, 1]
7	[20, 2000]	[2, 5]	[0.9, 1]	[0.1, 1]
8	[20, 50]	[2, 70]	[0.1, 1]	[0.9, 1]
9	[20, 2000]	[2, 5]	[0.1, 1]	[0.9, 1]
10	[20, 2000]	[2, 70]	[0.9, 1]	[0.9, 1]
11	[20, 2000]	[2, 70]	[0.1, 1]	[0.9, 1]
12	[20, 2000]	[2, 70]	[0.9, 1]	[0.1, 1]
13	[20, 2000]	[2, 5]	[0.1, 1]	[0.1, 1]
14	[20, 50]	[2, 70]	[0.1, 1]	[0.1, 1]
15	[20, 2000]	[2, 70]	[0.1, 1]	[0.1, 1]

different sets of QoS value ranges. In this experiment, the number of abstract services n is set to 15, the number of candidate cloud services m is set to 400, the percentage φ of QoS constraints is set to 0.4, and the stopping criterion is when the number of iterations during the best solution doesn't improve max_gen_imp when being set to 50 or the maximum number of iteration $MaxItr$ is reached (i.e. $max\_gen\_imp = 50$ or $MaxItr = 1000$) .Table. 13 lists the comparison results of the optimal fitness values obtained by each compared algorithm for each considered set of QoS value ranges. There are five columns in the table. The first column presents the corresponding set of QoS value ranges where their attribute's intervals are shown in Table. 12. The following columns present the optimal fitness values of the compared algorithms for each set of QoS value ranges. The row (average row) in the table presents the average value of the optimal fitness values obtained by each compared algorithm for all sets of QoS value ranges. The last two rows (quality rate and weakness rate rows) in the table present the proportion, in which each algorithm is better than the other algorithms and the proportion, in which each algorithm is poorer than one or all the other algorithms, respectively. It can be seen from Table. 13 that our proposed HGA has obtained nine best fitness values out of fifteen cases, which is better than the other three compared algorithms. HGA has obtained an average value of 0.4316, which is better than DGBAC, TGA and SFOA where their average values are (0.3869, 0.3093, 0.3450), respectively. The quality rate of HGA is better than the other compared algorithms. Moreover, the weakness rate of our proposed algorithm is the least value compared to the other algorithms. Therfore, it can be concluded that our algorithm verifies the efficiency and the robustness for solving QoS-aware cloud service composition under different QoS value ranges.

Table 13 Comparison results of the optimal fitness values for each set of QoS value ranges

Sets	DGABC	HGA	TGA	SFOA
1	0.365	**0.38392**	0.28196	0.30932
2	**0.79443**	0.79144	0.79165	0.70631
3	**0.79446**	0.79279	0.79265	0.75147
4	0.34338	**0.39838**	0.26784	0.28089
5	0.1709	0.15672	0.0	**0.17488**
6	0.33345	**0.46075**	0.29854	0.29682
7	0.3336	**0.5129**	0.2881	0.2961
8	0.3759	**0.3944**	0.1960	0.2960
9	0.3554	**0.4617**	0.2857	0.3260
10	0.7710	**0.7912**	0.7681	0.6322
11	0.3439	**0.4963**	0.3245	0.2807
12	0.3211	**0.4005**	0.3446	0.2909
13	0.1718	0.1314	0.0	**0.1875**
14	0.1499	0.1387	0.0	**0.1696**
15	**0.1797**	0.1635	0.0	0.1768
Average	0.3869	**0.4316**	0.3093	0.3450
Quality rate	20 %	**60 %**	0 %	20 %
Weakness rate	80 %	**40 %**	100 %	80 %

The best values are in bold

Conclusion and future work

In this study, a hybrid approach, using genetic algorithm and fruit fly optimization algorithm, is proposed to solve the QoS-aware cloud service composition. In order to speed up the convergence of the proposed algorithm, an improved initial population based on heuristic local selection method is presented. The HGA combines two phases to perform the evolutionary process search. The global search process is performed by the genetic algorithm phase, which applies a novel roulette wheel selection to avoid premature convergence and getting stack in local optima. To reduce the computation time and to maintain a balance between the exploration and the exploitation abilities of the proposed HGA, the local search process was employed by the fruit fly optimization phase. To prevent the loss of the best solutions during the evolutionary process, the elitism strategy is used as a replacement strategy for each generation of the proposed algorithm. The parameter settings of our HGA were tuned and calibrated using the Taguchi method of design of experiment (DOE); the optimal values of these parameters were suggested. To validate the performance of HGA in terms of optimality, computation

time, convergence speed, the feasibility rate, and the results of comparisons with other algorithms, have demonstrated the effectiveness and the efficiency of the proposed HGA. It is also verified that weights and value ranges of QoS attributes will not affect the effectiveness and the robustness of our HGA.

In the current work, the multi-objective constrained QoS-aware cloud service composition problem is reduced into mono-objective problem by using the simple additive weighting method, this kind of methods has some important drawbacks compared to pareto-based approaches (Cremene et al. 2016). Interdependencies and correlations among cloud services are not considered (Wu et al. 2014; Jin et al. 2015), and also the influence of the distribution of cloud services on distributed clouds is omitted (Wang et al. 2015). Moreover, the formulation of this problem in distributed cloud environment with multiple objectives, constraints and services correlations should also be investigated. In the future, we intend to improve and apply the proposed algorithm to solve this problem under multi-objective constraints in distributed cloud environment, which takes both QoS and interdependencies of cloud services into consideration.

References

Alrifai, M., & Risse, T. (2009). Combining global optimization with local selection for efficient QoS-aware service composition. In: *International World Wide Web Conference Committee, Madrid, Spain* (pp. 881–890).

Alrifai, M., Skoutas, D., & Risse, T. (2010). Selecting skyline services for QoS-based web service composition. In: *International World Wide Web Conference Committee, 2010, Raleigh, North Carolina, USA*.

Ardagna, D., & Pernici, B. (2007). Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, *33*(6), 369–384.

Canfora, G., Di Penta, M., Esposito, R., & Villani, M. L. (2005). An approach for QoS-aware service composition based on genetic algorithms. In: *Proceedings of the Conference on Genetic and Evolutionary Computation* (pp. 1069–1075). Berlin: Springer.

Cremene, M., Suciu, M., Dumitrescu, D., & Pallez, D. (2016). Comparative analysis of multi-objective evolutionary algorithms for QoS-aware web service composition. *Applied Soft Computing*, *39*, 124–139.

Ding, Z., Liu, J., Sun, Y., Jiang, C., & Zhou, Meng Chu. (2015). A transaction and QoS-aware service selection approach based on genetic algorithm. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *45*(7), 1035–1046.

El Haddad, J., Manouvrier, M., & Rukoz, M. (2010). TQoS: Transactional and QoS-aware selection algorithm for automatic web service composition. *IEEE Transactions on Services Computing*, *3*(1), 73–85.

Gao, Z., Chen, J., Qiu, X., & Meng, L. (2009). QoE/QoS driven simulated annealing-based genetic algorithm for Web services selection. *The Journal of China Universities of Posts and Telecommunications*, *16*(Suppl), 102–107.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization & machine learning*. Reading, MA: Addison-Wesley.

Huo, Y., Zhuang, Y., Gu, J., Ni, S., & Xue, Yu. (2015). Discrete gbest-guided artificial bee colony algorithm for cloud service composition. *Applied Intelligence*, *42*, 661–678.

Jin, H., Yao, X., & Chen, Y. (2015). Correlation-aware QoS modeling and manufacturing cloud service composition. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-015-1080-2.

Jula, A., Sundararajan, E., & Othman, Z. (2014). Cloud computing service composition: A systematic literature review. *Expert Systems with Applications*, *41*(8), 3809–3824.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC). *Journal of Global Optimization*, *39*, 459–471.

Liao, J., Liu, Y., Zhu, X., & Wang, J. (2014). Accurate sub-swarms particle swarm optimization algorithm for service composition. *The Journal of Systems and Software*, *90*, 191–203.

Li, Jun-qing, Pan, Quan-ke, Mao, Kun, & Suganthan, P. N. (2014). Solving the steelmaking casting problem using an effective fruit fly optimisation algorithm. *Knowledge-Based Systems*, *72*, 28–36.

Ma, Y., & Zhang, C. (2008). Quick convergence of genetic algorithm for QoS-driven web service selection. *Computer Networks*, *52*, 1093–1104.

Montgomery, D. C. (2005). *Design and analysis of experiments*. Arizona: Wiley.

Mousavi, S. M., Alikar, N., & Niaki, S. T. A. (2015a). An improved fruit fly optimization algorithm to solve the homogeneous fuzzy series-parallel redundancy allocation problem under discount strategies. *Soft Computing*,. doi:10.1007/s00500-015-1641-5.

Mousavi, S. M., Alikar, N., Niaki, S. T. A., & Bahreininejad, A. (2015b). Optimizing a location allocation-inventory problem in a two-echelon supply chain network: A modified fruit fly optimization algorithm. *Computers and Industrial Engineering*, *87*, 543–560.

Pan, W. T. (2012). A new fruit fly optimization algorithm: Taking the financial distress model as an example. *Knowledge-Based Systems*, *26*, 69–74.

Tang, M., & Ai, L. (2010). A hybrid genetic algorithm for the optimal constrained web service selection problem in web service composition. In: *Proceeding of the 2010 world congress on computational intelligence* (pp. 1–8). Barcelona.

Tao, F., LaiLi, Y., Xu, L., & Zhang, L. (2013). FC-PACO-RM: A parallel method for service composition optimal-selection in cloud manufacturing system. *IEEE Transactions on Industrial Informatics*, *9*(4), 2023–2033.

Tao, F., Zhao, D., Hu, Y., & Zhou, Z. (2008). Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system. *IEEE Transactions on Industrial Informatics*, *4*(4), 315–327.

Wang, L., Shi, Y., & Liu, S. (2015). An improved fruit fly optimization algorithm and its application to joint replenishment problems. *Expert Systems with Applications*, *42*, 4310–4323.

Wang, S., Sun, Q., Zou, H., & Yang, F. (2013). Particle swarm optimization with skyline operator for fast cloud-based web service composition. *Mobile Network and Application*, *18*, 116–121.

Wang, D., Yang, Y., & Mi, Z. (2015). A genetic-based approach to web service composition in geo-distributed cloud environment. *Computers and Electrical Engineering*, *43*, 129–141.

Wang, L., Zheng, X. L., & Wang, S. Y. (2013). A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem. *Knowledge-Based Systems*, *48*, 17–23.

Wu, Q., & Zhu, Q. (2013). Transactional and QoS-aware dynamic service composition based on ant colony optimization. *Future Generation Computer Systems*, *29*, 1112–1119.

Wu, Q., Zhu, Q., & Zhou, M. (2014). A correlation-driven optimal service selection approach for virtual enterprise establishment. *Journal of Intelligent Manufacturing*, *25*, 1441–1453.

Xue, X., Wang, S., & Lu, B. (2016). Manufacturing service composition method based on networked collaboration mode. *Journal of Network and Computer Applications*, *59*, 28–38.

Zeleny, M. (1982). *Multiple criteria decision making*. New York: McGraw-Hill.

Zeng, L. Z., Benatallah, B., Ngu, A. H. H., Dumas, M., Kalagnanam, J., & Chang, H. (2004). QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, *30*(5), 311–327.

Zhang, Y., Cui, G., Wang, Y., Guo, X., & Zhao, Shu. (2015). An optimization algorithm for service composition based on an improved FOA. *Tsinghua Science and Technology*, *20*(1), 90–99.

Zhang, B., Pan, Q.-K., Zhang, X.-L., & Duan, P.-Y. (2015). An effective hybrid harmony search-based algorithm for solving multidimensional knapsack problems. *Applied Soft Computing*, *29*, 288–297.

Zhao, J., & Xiaofang, Y. (2015). Multi-objective optimization of stand-alone hybrid PV-wind-diesel-battery system using improved fruit fly optimization algorithm. *Soft Computing*. doi:10.1007/s00500-015-1685-6

Zheng, X., Wang, L., & Wang, S. (2014). A novel fruit fly optimization algorithm for the semiconductor final testing scheduling problem. *Knowledge-Based Systems*, *57*, 95–103.