International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 24 (2018) pp. 17055-17065
© Research India Publications. http://www.ripublication.com

An Efficient QoS-Aware Cloud Service Composition Using a Hybrid Genetic–Fruit Fly Optimization Approach

Vivek Kumar Jaiswal [1] and Priyanka[2]

[1] Department of Information Technology, Rajkiya Engineering College Bijnor, Bijnor, UP, India

[2] Department of Computer Science and Engineering, Indian Institute of Technology (Indian School of Mines) IIT(ISM) Dhanbad, Dhanbad, Jharkhand, India

ABSTRACT

The widespread adoption of cloud computing has resulted in a large number of cloud services that provide similar functionality while exhibiting varying Quality of Service (QoS) characteristics. This growing diversity makes the process of cloud service composition increasingly complex, particularly when multiple QoS constraints must be satisfied at the same time. The problem becomes computationally demanding as the number of candidate services increases, and it is widely recognized as an NP-hard optimization problem. In this work, a hybrid optimization approach is introduced to address the challenges of QoS-aware cloud service composition. The proposed framework integrates a Genetic Algorithm with the Fruit Fly Optimization Algorithm (HGA) in order to exploit their complementary strengths. The Genetic Algorithm is used to explore the global search space effectively, whereas the Fruit Fly Optimization Algorithm is applied to enhance local search and improve convergence efficiency. Service composition solutions are evaluated using the QWS dataset, and an improved roulette-wheel selection strategy is incorporated to strengthen solution selection during the evolutionary process. Additionally, a localized refinement mechanism is employed to limit computational overhead while maintaining a balanced exploration–exploitation trade-off. The proposed approach is evaluated based on solution quality, computational efficiency, convergence behavior, and feasibility rate, and its performance is compared with that of the gbest-guided artificial ant bee colony algorithm. Experimental results indicate that the proposed hybrid framework consistently produces high-quality service compositions and demonstrates robust performance under different composition scenarios.

Keywords : Cloud service composition; Quality of Service (QoS); Hybrid optimization; Genetic Algorithm; Fruit Fly Optimization Algorithm; Metaheuristic techniques; Service selection; Cloud computing

1. Introduction

Service-oriented computing has played an important role in improving how computing resources and services are shared across diverse platforms. By enabling interoperability among heterogeneous systems, it has laid the foundation for flexible and scalable service composition. The emergence of cloud computing has further strengthened this paradigm by offering on-demand access to computing resources, allowing organizations especially small and medium-sized enterprises to avoid the high cost of building and maintaining dedicated data centers. Although this model significantly lowers operational expenses, it has also led to an overwhelming growth in the number of cloud services available online.

As the cloud ecosystem continues to expand, selecting suitable services and combining them into an effective composite solution has become increasingly challenging. Many cloud services deliver comparable functionality, which makes functional requirements alone insufficient for service selection. Instead, non-functional factors, commonly known as Quality of Service (QoS) attributes, have gained considerable importance. Attributes such as response time, availability, reliability, and cost strongly influence the overall performance of a service composition and directly affect user satisfaction. Consequently, QoS-aware decision-making has become a central concern in cloud service composition.

QoS-aware cloud service composition involves selecting one service for each task in a workflow while satisfying multiple QoS constraints simultaneously. The number of possible service combinations grows exponentially as the number of candidate services increases, making the problem computationally intensive and inherently NP-hard. Over the years, researchers have proposed various solution approaches, ranging from exact optimization techniques to heuristic and meta-heuristic algorithms. While exact methods can guarantee optimal solutions, they often fail to scale efficiently for large problem instances. On the other hand, many meta-heuristic approaches provide reasonable solutions within acceptable time limits but may suffer from slow convergence or premature stagnation.

To address these challenges, this study introduces a hybrid optimization framework that combines the Genetic Algorithm (GA) with the Fruit Fly Optimization Algorithm (FOA). The motivation behind this integration is to exploit the complementary strengths of both techniques. GA is effective in exploring the global search space, whereas FOA is well suited for rapid convergence and local refinement. By coordinating these two mechanisms, the proposed approach improves search efficiency and enhances solution quality while keeping computational cost under control. Experimental evaluations demonstrate that the proposed method consistently outperforms the conventional GA and the gbest-guided discrete artificial bee colony (DGABC) algorithm in terms of convergence speed and overall solution quality.

2. Related Work

Before cloud computing became widely adopted, research on service composition was mainly conducted within Service-Oriented Architecture (SOA) and grid computing environments. In SOA-based systems, service composition generally involves designing workflows by mapping abstract tasks to appropriate concrete services. In grid computing, the focus is more aligned with resource scheduling, where the objective is to efficiently allocate distributed computational resources to meet execution requirements [1].

Cloud service composition extends these earlier models while introducing additional challenges unique to cloud environments. In particular, it requires the discovery, selection, and integration of cloud services while satisfying user-defined non-functional requirements and Service Level Agreement (SLA) constraints. Important considerations include automation of the composition process, execution control (centralized or decentralized), service compatibility, and consistency. Moreover, cloud-specific factors such as cost efficiency, scalability, responsiveness, reliability, and long-term service availability significantly influence composition decisions. Several studies have addressed cloud service composition using traditional algorithmic approaches; however, these methods often face limitations when dealing with large-scale service repositories [2].

To improve the quality of composed services, hybrid optimization strategies have been explored in prior work. A two-phase optimization framework combining global optimization with local service selection was proposed in [3], where global QoS constraints were decomposed into local constraints using Mixed Integer Programming (MIP). This transformation enabled parallel service selection and reduced the overall search complexity. Similarly, a dynamic and reliable service composition framework was introduced in [4], where a Cultural Genetic Algorithm (CGA) was employed in the initial phase to identify promising composition structures. In the subsequent phase, global QoS constraints were converted into local constraints, and concrete services were selected using an enhanced case-based reasoning approach. Although these methods demonstrated improved QoS satisfaction, they primarily emphasized solution optimality while paying limited attention to computational efficiency.

As the demand for scalable solutions increased, researchers shifted toward combinatorial and meta-heuristic techniques to obtain near-optimal solutions within acceptable execution time [5]. The approach presented in [6] focused on optimizing QoS attributes while maintaining reasonable computational overhead. Genetic Algorithm–based methods have also been widely studied. For example, the work in [7] incorporated user preferences and SLA constraints into the service composition process and applied skyline techniques to improve convergence speed and service quality. To further reduce execution time, a GA-based method for web service composition was proposed in [8], ensuring functional correctness while optimizing QoS parameters.

Other population-based optimization algorithms have also been investigated. Particle Swarm Optimization (PSO) with integer-based encoding was applied in [9] to efficiently explore the solution space, where skyline operators and binary selection mechanisms were used to eliminate low-quality candidate services. Although such meta-heuristic approaches generally outperform classical methods in terms of execution time, their performance tends to degrade as the number of available services increases. Consequently, scalability remains a persistent challenge in large-scale cloud service composition problems [10]. Karimi et al. [11] proposed a QoS-aware service composition framework using evolutionary optimization to address the trade-off between solution optimality and computational efficiency in dynamic cloud environments. Hayyolalam et al. [12] proposed a survey paper on QoS aware cloud service composition and selection in cloud environment. A QoS aware cloud service composition technique based on fuzzy set theory (FST) and genetic algorithm (GA), a triangular fuzzy genetic algorithm (TGA) is proposed for solving the service composition problem [13]. Liu et al. [14] investigated QoS-aware service composition in cloud manufacturing environments and highlighted the limitations of one-to-one mapping–based service composition in terms of overall QoS and composition success rate. To overcome these issues, they introduced a synergistic elementary service group–based composition model and employed an improved genetic algorithm to manage the increased optimization complexity, demonstrating superior performance over conventional approaches. However, the computational overhead of group-based composition remains a challenge as the scale of candidate services continues to grow. Gabrel et al. [15] investigated QoS-aware automatic service composition by modeling the selection process as an optimization problem, where global QoS constraints are derived from individual service attributes to guide optimal composition.

3. CLOUD SERVICE SELECTION BASED ON QoS PARAMETER

The fig. 1 presents a structured approach for selecting cloud services, in which each abstract task is associated with several candidate services identified during the discovery phase. Quality of Service attributes, such as response time, availability, success ability, compliance, price , latency, and throughput, are then evaluated to choose the most suitable service from each layer, resulting in an optimized composite cloud service.

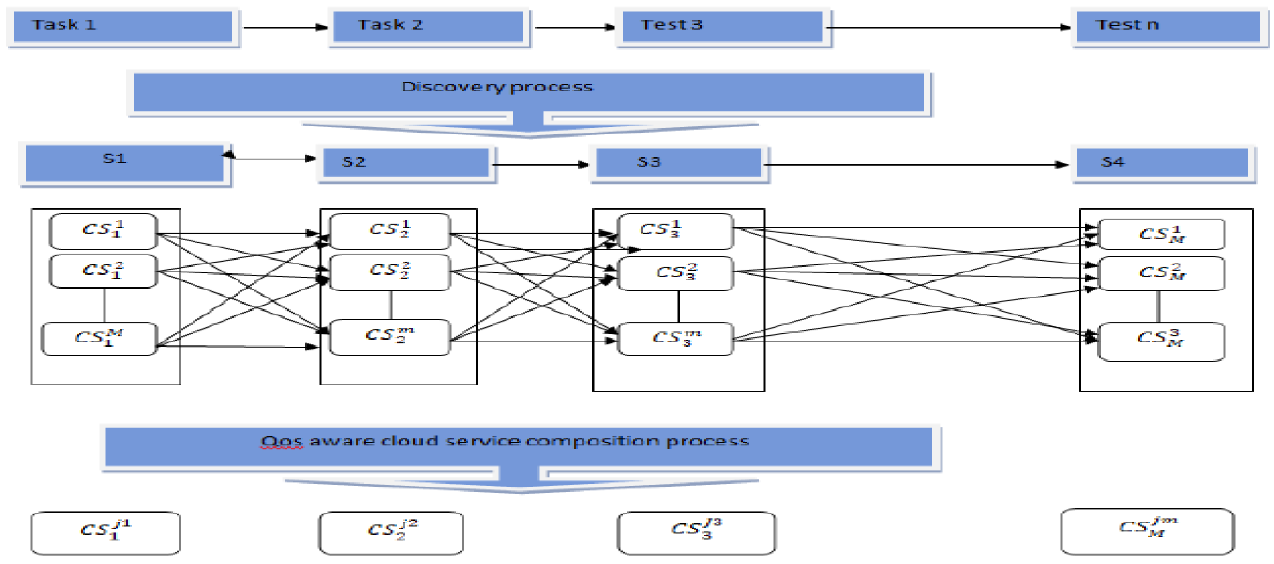

Fig .1 Cloud service composition process

3.1 Problem statement and workflow description

To formally describe the QoS-aware cloud service composition problem, the following definitions and assumptions are considered.

Step 1:
A complex service request is decomposed into a set of atomic tasks denoted as: ACSC = {T_1, T_2, …, T_n}, where each Ti represents an individual atomic task and n is the total number of such tasks. For each atomic task Ti, a corresponding candidate cloud service set Si = {CS_{i1}, CS_{i2}, …, CS_{im}} is defined, where CS_{ij} represents the jth concrete cloud service capable of executing task T_i, and m denotes the number of candidate services available for each task.

Step 2:
Each cloud service is characterized by a set of Quality of Service (QoS) attributes defined as: QoS = {q1, q2, …, qr}, where r denotes the total number of QoS parameters. These attributes are categorized into positive QoS attributes (QoS+) and negative QoS attributes (QoS−). For negative QoS attributes, such as response time and cost, lower values indicate better performance.

Step 3:
User-defined global constraints are represented by the set: GCst = {Cst_1, Cst_2, …, Cst_k}, where each Cst_t ($t \leq k$

and $k \leq r$) specifies a constraint applied to the corresponding QoS attribute qt.

Step 4:
A weight vector $W = \{w_1, w_2, \ldots, w_r\}$ is assigned to represent the importance of each QoS attribute. Each wt lies in the range [0,1], and the sum of all weights satisfies: Σ wt = 1.

3.2 Genetic Algorithm (GA)
Genetic Algorithm (GA) is a population-based optimization technique inspired by natural evolution. It starts with a randomly generated population of candidate solutions, where each individual represents a possible cloud service composition. A fitness function evaluates the quality of each individual based on QoS satisfaction. Through selection, crossover, and mutation operations, the population evolves toward optimal solutions.

General Steps of GA :
1. Initialize a population of candidate solutions.
2. Evaluate the fitness of each individual.
3. Repeat until termination condition is met:
- Select individuals for reproduction.
- Apply crossover to generate offspring.
- Perform mutation to maintain diversity.
- Recalculate fitness values.

4. Fruit Fly Optimization Algorithm (FOA)

3.3 Fruit fly optimization (FOA)

Fruit Fly Optimization Algorithm (FOA) is a swarm intelligence-based global optimization method inspired by the food-searching behavior of fruit flies. The algorithm utilizes smell-based and vision-based search mechanisms.

Step 1: Initialization
Randomly, initialize the fruit fly population in the search space. Define population size (SN) and maximum generations (MAX_GEN).

Step 2: Smell-Based Search
Generate fruit flies near the current position and evaluate their fitness values.

Step 3: Vision-Based Search
Identify the fruit fly with the best fitness value and move the population toward this optimal position.

Step 4: Termination
Repeat the search process until the maximum number of generations is reached.

4. Cloud Service Composition

The objective of this work is to design an efficient cloud service composition framework capable of generating near-optimal composite cloud services within a limited computational time. To accomplish this goal, a hybrid optimization strategy, referred to as the Hybrid Genetic–Fruit Fly Algorithm (HGA), is proposed. The framework integrates the global search strength of the Genetic Algorithm (GA) with the local exploitation capability of the Fruit Fly Optimization Algorithm (FOA). The proposed approach consists of multiple stages, including solution encoding, enhanced population initialization, fitness evaluation, genetic evolution, and fruit fly–based refinement. A modified roulette-wheel selection mechanism is introduced in the GA phase to improve selection efficiency and exploration performance. To further control execution time and maintain a balance between exploration and exploitation, FOA is applied after each genetic evolution cycle. Moreover, an elitism strategy is employed to ensure that high-quality solutions are preserved during the evolutionary process.

4.1 Encoding Phase

In the proposed HGA model, each individual in the population represents a complete cloud service composition solution. The solution is encoded as an integer-based vector, where each position corresponds to an atomic task, and the integer value stored at that position indicates the selected concrete cloud service from the associated candidate service set.

4.2 Population Initialization Phase

In conventional evolutionary optimization techniques such as Genetic Algorithms (GA) and Fruit Fly Optimization Algorithm (FOA), the initial population is typically generated using random sampling. Although this approach is computationally simple, it often produces individuals with low solution quality and limited diversity, which may slow down convergence and increase the likelihood of premature stagnation. In contrast, an initial population that simultaneously maintains high solution quality and sufficient diversity is more effective in guiding the search process toward optimal solutions within a shorter convergence time.

To address this limitation, the proposed model incorporates a heuristic local optimization–based selection strategy for population initialization. Instead of relying solely on randomly generated individuals, this strategy refines the initial population by constructing improved solutions using QoS-aware evaluation, thereby enhancing the overall search efficiency from the early stages of evolution.

4.2.1 Local Optimization Selection Method

The local optimization selection method is designed to generate an individual (denoted as *ind*) with a high level of solution quality. First, all Quality of Service (QoS) attributes are normalized to a unified numerical range of [0, 1] in order to eliminate scale inconsistencies among heterogeneous attributes. The normalization procedures for negative and positive QoS attributes of a composite cloud service (CCS) are defined in Equations (1) and (2), respectively. The normalized value of $Q(CCS)q_t \in [0, 1]$ for negative and positive QoS attributes are shown is Equations (3) and (4), respectively. Following normalization, each QoS attribute value is multiplied by its corresponding user-defined preference weight *wt*, reflecting the relative importance of that attribute. The weighted normalized values are then aggregated to compute the overall score of the CCS, as expressed in Equation (5). This aggregated score serves as a quantitative measure of the solution quality and is subsequently used to guide the selection of high-quality individuals during population initialization. By integrating normalization, preference weighting, and local optimization–based selection, the proposed initialization mechanism produces a diverse set of promising individuals, which significantly improves convergence speed and solution robustness in the subsequent evolutionary phases.

Normalization of negative QoS attributes (i.e. $q_t \in QoS^-$)

$$\mathbf{QoS^-(q_t)} = \frac{agg(qtmax)-agg(qt)}{agg(qtmax)-agg(qtmin)} \qquad (1)$$

Normalization of positive QoS attributes (i.e. $q_t \in QoS^+$)

$$\mathbf{QoS^+(q_t)} = \frac{agg(qt)-agg(qtmax)}{agg(qtmax)-agg(qtmin)} \qquad (2)$$

The normalized value of $Q(CCS)q_t \in [0, 1]$ for negative and positive QoS attributes are shown is Equations (3) and (4), respectively

$$\mathbf{Q(CCS)qt} = \begin{cases} \frac{agg(qtmax)-agg(qt)}{agg(qtmax)-agg(qtmin)}, & if\ agg(qtmin) \neq agg(qtmax) \\ 1, & if\ agg(qtmin) = agg(qtmax) \end{cases} \qquad (3)$$

$$Q(CCS)qt = \begin{cases} \frac{agg(qt)-agg(qtmax)}{agg(qtmax)-agg(qtmin)}, & if\ agg(qtmin) \neq agg(qtmax) \\ 1, & if\ agg(qtmin) = agg(qtmax) \end{cases} \qquad (4)$$

$$Score\ (ccs) = \sum_{x=1}^{r} (ccs)t * wt \qquad (5)$$

After completing the normalization process, the following procedure is carried out.

Step 1: The local score of each concrete cloud service in the candidate set CS_i^j is computed using the Simple Additive Weighting (SAW) method (Eq. 6) .

$$Scorel(Csij) = \sum_{qt\in QoS-} \frac{qt,i\ max-qt,i\ j}{qt,i\ \max-qt,i\ min} * Wt + \sum_{qt\in QoS-} \frac{-qt,i\ min+qt,i\ j}{qt,i\ \max-qt,i\ min} \qquad (6)$$

where, $q_{t,\ i}{}^{j}$ represents the value of the tth QoS criterion for the jth concrete cloud service within the ith candidate set S_i. The symbols $q_{t,i}^{min}$□ and $q_{t,i}^{max}$ denote the minimum and maximum values of the tth attribute observed in the candidate set S_i , respectively, while w_t indicates the user-assigned preference weight associated with the tth QoS attribute.

Step 2:

For each candidate cloud service set S_i, an individual *ind* corresponds to the ith position in the solution vector does.

do

Select two distinct concrete cloud services, CS_i^j and CS_i^k randomly from S_i using a binary tournament selection mechanism. The local scores of the selected services are then compared.

1. **If** the local score of $Score_l\ (CS_i^k) > Score_l\ (CS_i^j)$ at ith the position of *ind* is updated with the index k;

2. **Else**, it is replaced with the index j.

3. **End If**

End For

4.2.2 Initial Population Construction Procedure

The process of generating the initial population is carried out as follows:

Step 1: A high-quality candidate solution is first produced using the proposed local optimization-based selection strategy.

Step 2: Let the iteration counter be initialized as itr = 1. The following steps are executed until itr reaches the predefined population size, denoted by *PopSize*.

Step 3: A new optimized solution is generated using the same local optimization approach. If this solution is distinct from all existing individuals in the current population, it is added to the population and the iteration counter is incremented by one. Otherwise, the solution is discarded and the generation process is repeated until a unique solution is obtained.

4.3 Crossover Operator

After the parent selection stage, pairs of individuals are recombined using a crossover mechanism to generate new candidate solutions. This process integrates genetic information from both parents to form offspring with potentially improved characteristics. Two crossover strategies are adopted in the proposed framework, namely single-point crossover and double-point crossover.

In the single-point crossover approach, a crossover position is selected at random along the chromosome. The gene segments located beyond this position are then exchanged between the two parent chromosomes, resulting in two new offspring (as illustrated in Fig. 2a). In contrast, the double-point crossover method involves selecting two distinct crossover positions. The gene segment lying between these two points is swapped between the parent individuals; thereby producing new chromosomes with mixed genetic structures (see Fig. 2b).

To maintain diversity in the evolutionary process, a probabilistic mechanism is employed to select the crossover type. Specifically, a random value $r \in [0, 1]$ is generated. When $r \leq 0.5$, single-point crossover is applied; otherwise, the algorithm performs a double-point crossover during reproduction.

4.4 Mutation Operator

Following the crossover operation, mutation is applied to further enhance population diversity and prevent premature convergence. The mutation strategy operates by randomly selecting a gene position within the newly generated individual, where each gene represents an abstract cloud service. The selected gene is then replaced with an alternative concrete cloud service chosen randomly from the corresponding candidate service set (as shown in Fig. 2c).

This mutation mechanism enables the exploration of new regions within the search space by introducing controlled random variations, thereby increasing the likelihood of discovering high-quality solutions in subsequent generations.

4.5 FOA Phase (Local Exploitation)

The Fruit Fly Optimization Algorithm (FOA) phase is applied as a local exploitation mechanism to refine the high-quality solutions obtained from the genetic phase. For each promising solution produced by the genetic operators, FOA is employed to further enhance solution quality by exploring its local neighbourhood. During the smell-based foraging stage, a set of *SN* neighbouring solutions is generated around each selected individual. These neighbours are created using a mutation-based operation, where certain decision variables of the selected solution are randomly modified to produce nearby candidate solutions.

In the vision-based search stage, all *SN* neighbouring individuals of a given solution (*ind*) are evaluated according to the fitness function. The best-performing neighbour, referred to as *BestInd*, is then identified. If *BestInd* yields a better fitness value than the original individual *Ind*, it replaces *ind* in the population; otherwise, the original solution is retained. This process ensures that only improvements are accepted, thereby strengthening local search capability.

The FOA procedure applied to each solution generated by the genetic phase can be summarized as follows:

Step 1: For every individual ind_i, produced after the genetic phase, where i=1, 2,…,Pop Size, execute the subsequent steps.

Step 2: Compute the selection probability p_n of each individual ind_i, using Equations (17-18).

4.3 Fitness Evaluation

In the proposed framework, each individual in the generated population is assigned a fitness value that reflects its performance within the search space. The fitness evaluation mechanism is designed to distinguish between feasible and infeasible solutions while guiding the evolutionary process toward high-quality compositions.

To penalize constraint violations, a penalty function *Pn(ind)* is defined in Eq. 7 as follows:

$$\boldsymbol{Pn(ind)} = \begin{cases} \sum_{t=1}^{k} \boldsymbol{CstVt(ind)g * pt}, & \boldsymbol{if\ ind\ is\ infeasible} \\ \boldsymbol{0}, & \boldsymbol{if\ ind\ is\ feasible} \end{cases} \quad \textbf{(7)}$$

where, $CstV_t$(ind) represents the degree of constraint violation corresponding to the t^th QoS attribute, as formulated in Eq. (8). The parameter g controls the severity of the penalty and is set to g=2 in this study. The weighting factor $p_t \in [0, 1]$ for t = 1, 2, ……, k, indicates the relative importance of the t^th constraint, subject to the condition $\sum_{t=1}^{k} pt = 1$. This penalty-based fitness formulation ensures that feasible individuals are always favored over infeasible ones, while infeasible solutions are proportionally penalized based on the magnitude of their constraint violations.

$$\boldsymbol{Cstvt(ind)} = \begin{cases} \dfrac{\max(0, agg(qt) - Cstt)}{Cstt} & if\ \text{qt} \in \text{QoS}- \\ \dfrac{\max(0, agg(qt) - Cstt)}{Cstt} & if\ \text{qt} \in \text{QoS}+ \end{cases} \quad \textbf{(8)}$$

where, Cst_t and agg (q_t) represent the constraint threshold and the aggregated value of the t^th QoS attribute, respectively. For attributes with positive constraints, feasibility is satisfied when agg (q_t) ≤ , whereas for negatively constrained attributes, the condition agg(q_t) ≥ Cst_t must hold, as defined in Eq. (5). The constraint threshold Cst_t belongs to the global constraint set GCst. The term agg (qt) denotes the cumulative value of the t^th QoS attribute associated with an individual solution.

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

Where, *pn(ind)* represents the penalty assigned to an individual, while Score(ind) denotes its aggregated score as defined in Eq. (3). As indicated by Eq. (9), feasible solutions are always assigned higher fitness values than infeasible ones, thereby guiding the search process toward valid and high-quality solutions.

The dissimilarity between any two individuals is measured using the Hamming distance metric (Eq. 10), which quantifies the number of differing positions between their respective representations.

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

where, ind_k denotes the gene value at the i^th position of the k^th individual. The diversity measure div(ind_k), which quantifies the dissimilarity between an individual and the remaining members of the population, is defined in Eq. (11). Based on this diversity measure, the updated selection score $(indj)$ is computed as expressed in Eq. (12). This score is obtained using the Simple Additive Weighting (SAW) method, which integrates both fitness and diversity information into a single scalar value for each individual.

$$\boldsymbol{div(ind_j)} = \sum_{h=1}^{PopSize} \boldsymbol{std(fit(indj, indh))} \quad \textbf{(11)}$$

In this context, j ≠ h, PopSize represents the total number of individuals in the population, and std denotes the standard deviation. The term fit($indj$,$indh$) refers to the fitness values associated with individuals $indj$ and $indh$, respectively.

$$\boldsymbol{Scr_{sel}(ind_j) = N_d(ind_j)*wd + N_f(indj)*wf} \quad \textbf{(12)}$$

Where ,

$$\boldsymbol{Nf(indj)} = \begin{cases} \boldsymbol{1} & \boldsymbol{if(max(fitpop) = min(fitpop)} \\ \dfrac{fit(indj) - min(fitpop)}{max(fitpop) - min(fitpop)} & \boldsymbol{if\ (max(fitpop) \neq min(fit(pop)} \end{cases} \quad \textbf{(13)}$$

$$\boldsymbol{Nf(indj)} = \begin{cases} \boldsymbol{1} & \boldsymbol{if(max(divpop) = min(divpop)} \\ \dfrac{div(indj) - min(divpop)}{max(divpop) - min(divpop)} & \boldsymbol{if\ (max(divpop) \neq min(div(pop)} \end{cases} \quad \textbf{(14)}$$

According to Eqs. (12) – (14), fit($indj$) represents the fitness value of the j^th individual, while div(indj) denotes its corresponding diversity measure. The terms min($fitPop$) and max($fitPop$), indicate the minimum and maximum fitness values observed in the current population Pop, respectively. The weighting parameters w_f and w_d (Eqs. 15 - 16), are employed to regulate the relative influence of fitness and diversity components during the selection process.

$$\boldsymbol{Wd} = \frac{\boldsymbol{counter}}{\boldsymbol{2*maxitr}} + \boldsymbol{0.5} \quad \textbf{(15)}$$

$$\boldsymbol{wf = 0.5 - \frac{counter}{2 * maxitr}} \qquad \textbf{(16)}$$

Here, maxitr denotes the maximum number of iterations, while counter represents the current iteration index of the proposed algorithm. A new individual for the next generation, denoted as is selected using the following procedure.

Step 1: Generate a random real number r ∈ [0, 1] using a uniform random function.
Step 2: If r ≤ p1, the first individual ind_1 is selected. Otherwise, select the individual ind_i *(i =*1, ... *,PopSize),* such that $p_{i-1} < r \le pi$, where pi represents the cumulative selection probability of the ith individual in the current population Pop, computed as defined in Eqs. (17 - 18).

$$\boldsymbol{P1 = \frac{scrsel(ind1)}{\sum_{i=1}^{popSize} scrsel(indi)}} \qquad \textbf{(17)}$$

$$\boldsymbol{Pi = \frac{scrsel(indi)}{\sum_{j=1}^{popSize} scrsel(indj)} + p_{i-1},\ if,\ 2 \le j \le popsize} \qquad \textbf{(18)}$$

4.4 Genetic Phase (Global Exploration)

4.4.1 Selection Operator

In this phase, a modified roulette wheel selection strategy is introduced, where the selection probability of each chromosome is determined by jointly considering its fitness and diversity. This approach increases the likelihood of selecting individuals that are both well-performing and diverse, thereby enhancing population variability and avoiding premature convergence. To compute the updated selection scores, the diversity of each individual is quantified by measuring its dissimilarity from all other individuals within the current population.

4.4.2 Crossover Operator

After the parent selection stage, pairs of individuals are recombined using a crossover mechanism to generate new candidate solutions. This process integrates genetic information from both parents to form offspring with potentially improved characteristics. Two crossover strategies are adopted in the proposed framework, namely single-point crossover and double-point crossover.

In the single-point crossover approach, a crossover position is selected at random along the chromosome. The gene segments located beyond this position are then exchanged between the two parent chromosomes, resulting in two new offspring (as illustrated in Fig. 2a).

In contrast, the double-point crossover method involves selecting two distinct crossover positions. The gene segment lying between these two points is swapped between the parent individuals; thereby producing new chromosomes with mixed genetic structures (see Fig. 2b).

To maintain diversity in the evolutionary process, a probabilistic mechanism is employed to select the crossover type. Specifically, a random value r ∈ [0, 1] is generated. When r ≤ 0.5, single-point crossover is applied; otherwise, the algorithm performs a double-point crossover during reproduction.

4.4.3 Mutation Operator

Following the crossover operation, mutation is applied to further enhance population diversity and prevent premature convergence. The mutation strategy operates by randomly selecting a gene position within the newly generated individual, where each gene represents an abstract cloud service. The selected gene is then replaced with an alternative concrete cloud service chosen randomly from the corresponding candidate service set (as shown in Fig. 2c).This mutation mechanism enables the exploration of new regions within the search space by introducing controlled random variations, thereby increasing the likelihood of discovering high-quality solutions in subsequent generations

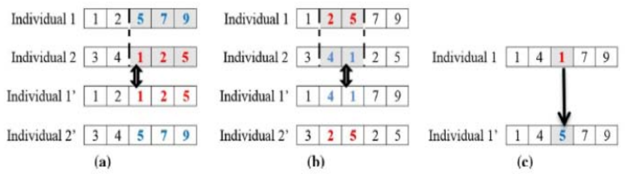

Fig. 2 Genetic operators used in the evolutionary phase: (a) single-point crossover, (b) double-point crossover, and (c) mutation operation.

5. Results and Analysis

5.1 Experimental Setup

To evaluate the performance of the proposed approach, a sequential Abstract Cloud Service Composition (ACSC) scenario is considered. In this setup, the composition consists of *n* abstract cloud services, where each abstract service is associated with a candidate set of *m* concrete cloud services. The experiments are conducted using the QWS dataset, which includes eight Quality of Service (QoS) attributes: response time, availability, throughput, success ability, reliability, , and compliance are treated as positive attributes, where higher values represent improved service quality. This experimental configuration enables a comprehensive assessment of the effectiveness and robustness of the proposed model under diverse QoS constraints.

5.2 Parameter setting of the proposed model

Table 1: Quantitative parameters used in simulation environment

	Cst	Weight	Penalty
Response time	0.92	0.1	0.2
Availability	0.89	0.2	0.1
Throughput	0.011	0.05	0.1
Success ability	0.12	0.05	0.2
Reliability	0.11	0.2	0.05
Compliance	0.21	0.1	0.05
Price	0.119	0.1	0.1
Latency	0.98	0.2	0.2

Table 1 presents the quantitative configuration of QoS-related parameters, including constraint thresholds, weighting coefficients, and penalty values adopted for the experimental evaluation. To assess the effectiveness of the proposed model, a comparative evaluation is carried out against two benchmark algorithms, namely the Simple Genetic Algorithm (GA) and the discrete gbest-guided Ant Bee Colony (DGABC) algorithm. The comparison is performed based on two key performance criteria: optimality and execution time.

Execution time: It represents the computational efficiency of the algorithms and is defined as the total time required identifying the optimal composite cloud service. This metric highlights the algorithm's capability to achieve high-quality solutions within a reasonable time frame, which is crucial for practical cloud service composition scenarios. Figure 3 presents a comparative analysis of the execution times of GA, DGABC, and HGA for varying numbers of abstract services. The results indicate that GA and DGABC generally achieve lower execution times, reflecting their ability to identify suitable composite services with reduced computational overhead. In contrast, HGA exhibits comparatively higher and more consistent execution times across all service configurations, suggesting increased processing complexity. DGABC shows performance close to GA and, in some cases, slightly better, highlighting its computational efficiency. Overall, the figure demonstrates that GA and DGABC are more suitable for time-sensitive cloud service composition scenarios.

Optimality: It reflects the quality of the obtained solution and is measured through the fitness value of the best composite cloud service identified by each algorithm. A higher fitness value indicates a more efficient and well-optimized service composition under the given QoS constraints. The fig, 4 illustrates the convergence characteristics of GA, DGABC, and HGA in terms of fitness value over successive generations. he proposed HGA demonstrates faster convergence and achieves superior solution quality compared to GA and DGABC as the number of iterations increases It can be observed that all three algorithms show a steady improvement in solution quality as the iterations progress. HGA consistently achieves the highest fitness values, indicating its stronger capability to reach superior composite service solutions. DGABC demonstrates stable convergence with moderate fluctuations, while GA converges more slowly and attains comparatively lower fitness values. Overall, the fig. 4 highlights the effectiveness of HGA in achieving better optimization performance within fewer generations. As illustrated in Fig. 5, the proposed approach achieves higher fitness values than the benchmark algorithms (GA and DGABC) when the number of abstract services varies from 5 to 9 with 200 candidate services per task.

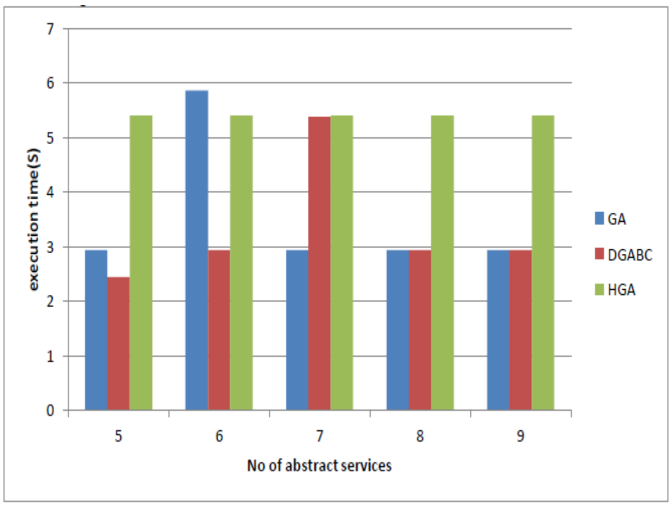

Fig. 3 Computational time comparison of GA, DGABC, and HGA under Scenario 1, where each abstract service is associated with 200 concrete services (m = 200) and the number of abstract services varies from 5 to 9.

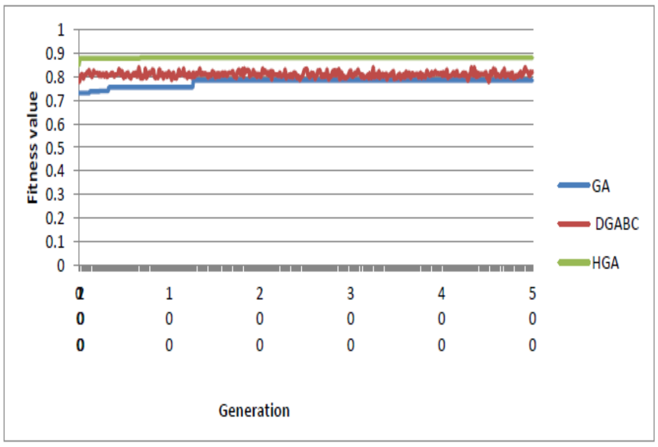

Fig. 4 Convergence behavior of the compared algorithms (GA, DGABC, and HGA) illustrating the improvement in the quality of the optimal composite service across iterations.

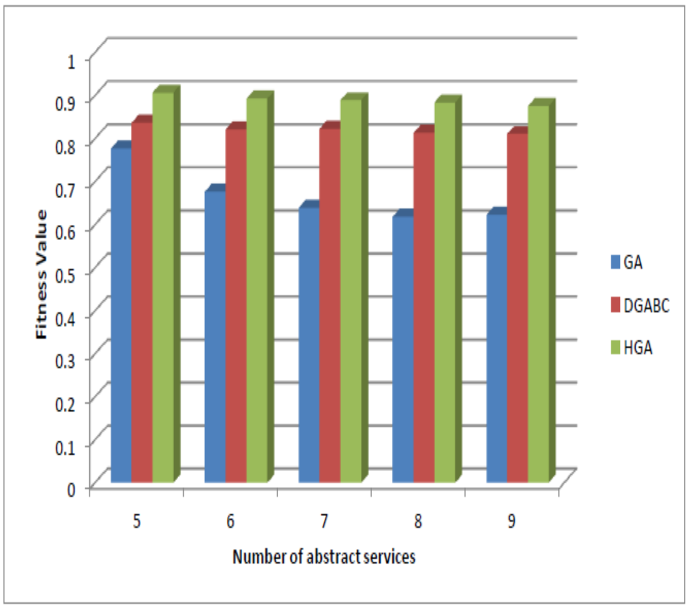

Fig. 5 Comparison of fitness value of GA, DGABC, and HGA optimality for different numbers of abstract services ranging from 5 to 9, with each abstract service having 200 candidate concrete services ($m = 200$).

6. Conclusion and Future Work

This study evaluated the proposed model by analyzing its performance in terms of solution optimality and computational time. Comparative experiments with existing algorithms demonstrate that the proposed approach is both efficient and effective in generating high-quality composite cloud services while maintaining lower execution time. The experimental results confirm its capability to achieve superior fitness values and faster convergence under varying service composition scenarios.

Despite these advantages, the current approach has certain limitations when compared with Pareto-based multi-objective techniques. Specifically, the multi-objective QoS-aware cloud service composition problem is transformed into a single-objective formulation using the Simple Additive Weighting (SAW) method, which may lead to information loss among conflicting objectives. Moreover, interdependencies and correlations among cloud services are not explicitly modeled, and the influence of service distribution across distributed cloud servers is not considered.

Future research will focus on addressing these limitations by formulating the problem within a fully distributed cloud environment that incorporates multiple objectives, complex constraints, and service interdependencies. The proposed model will be extended to a multi-objective optimization framework to simultaneously optimize diverse QoS attributes while accounting for correlations among services. This extension is expected to improve the practicality and robustness of cloud service composition in real-world distributed cloud infrastructures

References

[1] Z. Ye, X. Zhou, and A. Bouguettaya, “Genetic algorithm based QoS-aware service compositions in cloud computing,” in *Proc. Int. Conf. Database Systems for Advanced Applications (DASFAA)*, Berlin, Heidelberg, Germany: Springer, pp. 321–334, 2011.

[2] Z. Yong, L. Wei, L. Junzhou, and Z. Xiao, “A novel two-phase approach for QoS-aware service composition based on history records,” in *Proc. IEEE 5th Int. Conf. Service-Oriented Computing and Applications*, pp. 1–8, 2018.

[3] M. Alrifai, T. Risse, and W. Nejdl, “A hybrid approach for efficient web service composition with end-to-end QoS constraints,” *ACM Transactions on the Web*, vol. 6, no. 2, pp. 1–31, 2012.

[4] Z.-Z. Liu, D.-H. Chu, Z.-P. Jia, J.-Q. Shen, and L. Wang, “A two-stage approach for reliable dynamic web service composition,” *Knowledge-Based Systems*, vol. 97, pp. 123–143, 2016.

[5] Q. Wu, Q. Zhu, and M. Zhou, “A correlation-driven optimal service selection approach for virtual enterprise establishment,” *Journal of Intelligent Manufacturing*, vol. 25, no. 6, pp. 1441–1453, 2014.

[6] M. Teixeira, R. Ribeiro, C. Oliveira, and R. Massa, “A quality-driven approach for resource planning in service-oriented architectures,” *Expert Systems with Applications*, vol. 42, no. 12, pp. 5366–5379, 2015.

[7] Y. Wang and Z. Mi, “A genetic-based approach to web service composition in geo-distributed cloud environments,” *Computers & Electrical Engineering*, vol. 43, pp. 129–141, 2015.

[8] A. S. da Silva, H. Ma, and M. Zhang, “Genetic programming for QoS-aware web service composition and selection,” *Soft Computing*, vol. 20, no. 10, pp. 1–17, 2016.

[9] S. Wang, Q. Sun, H. Zou, and F. Yang, “Particle swarm optimization with skyline operator for fast cloud-based web service composition,” *Mobile Networks and Applications*, vol. 18, no. 1, pp. 116–121, 2013.

[10] Z. Zhao, X. Hong, and S. Wang, “A web service composition method based on merging genetic algorithm and ant colony algorithm,” in *Proc. IEEE Int. Conf. Computer and Information Technology (CIT)*, pp. 1007–1011, 2015.

[11] Karimi, M. B., Isazadeh, A., & Rahmani, A. M., "QoS-aware service composition in cloud computing using data mining techniques and genetic algorithm," *The Journal of Supercomputing*, *73*(4), 1387-1415, 2017.

[12] Hayyolalam, V., & Kazem, A. A. P., "A systematic literature review on QoS-aware service composition and selection in cloud environment," *Journal of Network and Computer Applications"*, *110*, 52-74, 2017.

[13] Xu, J., Guo, L., Zhang, R., Hu, H., Wang, F., & Pei, Z., "QoS-aware service composition using fuzzy set theory and genetic algorithm," *Wireless Personal Communications*, *102*(2), 1009-1028, 2018.

[14] Liu, B., & Zhang, Z., "QoS-aware service composition for cloud manufacturing based on the optimal construction of synergistic elementary service groups," *The International Journal of Advanced Manufacturing Technology*, *88*(9), 2757-2771, 2017.

[15] Gabrel, V., Manouvrier, M., Moreau, K., & Murat, C., " QoS-aware automatic syntactic service composition problem: Complexity and resolution," *Future Generation Computer Systems*, *80*, 311-321, 2018.