Neural Computing and Applications (2022) 34:9043–9068
https://doi.org/10.1007/s00521-022-06925-y

ORIGINAL ARTICLE

Modified firefly algorithm for workflow scheduling in cloud-edge environment

Nebojsa Bacanin[1] · Miodrag Zivkovic[1] · Timea Bezdan[1] · K. Venkatachalam[2] · Mohamed Abouhawwash[3,4]

Received: 13 July 2021 / Accepted: 4 January 2022 / Published online: 2 February 2022

Abstract

Edge computing is a novel technology, which is closely related to the concept of Internet of Things. This technology brings computing resources closer to the location where they are consumed by end-users—to the edge of the cloud. In this way, response time is shortened and lower network bandwidth is utilized. Workflow scheduling must be addressed to accomplish these goals. In this paper, we propose an enhanced firefly algorithm adapted for tackling workflow scheduling challenges in a cloud-edge environment. Our proposed approach overcomes observed deficiencies of original firefly metaheuristics by incorporating genetic operators and quasi-reflection-based learning procedure. First, we have validated the proposed improved algorithm on 10 modern standard benchmark instances and compared its performance with original and other improved state-of-the-art metaheuristics. Secondly, we have performed simulations for a workflow scheduling problem with two objectives—cost and makespan. We performed comparative analysis with other state-of-the-art approaches that were tested under the same experimental conditions. Algorithm proposed in this paper exhibits significant enhancements over the original firefly algorithm and other outstanding metaheuristics in terms of convergence speed and results' quality. Based on the output of conducted simulations, the proposed improved firefly algorithm obtains prominent results and managed to establish improvement in solving workflow scheduling in cloud-edge by reducing makespan and cost compared to other approaches.

Keywords Edge computing · Swarm intelligence · Workflow scheduling · Firefly algorithm · Genetic operator · Quasi-reflection-based learning

✉ K. Venkatachalam
venkatachalam.kandasamy@uhk.cz

Nebojsa Bacanin
nbacanin@singidunum.ac.rs

Miodrag Zivkovic
mzivkovic@singidunum.ac.rs

Timea Bezdan
tbezdan@singidunum.ac.rs

Mohamed Abouhawwash
abouhaww@msu.edu

1 Singidunum University, Danijelova 32, Belgrade 11000, Serbia

2 Department of Applied Cybernetics, Faculty of Science, University of Hradec Králové, 50003 Hradec Králové, Czech Republic

3 Department of Mathematics, Faculty of Science, Mansoura University, Mansoura 35516, Egypt

4 Department of Computational Mathematics, Science, and Engineering (CMSE), Michigan State University, East Lansing, MI 48824, USA

1 Introduction

Internet of Things (IoT) provides a set of mechanisms aimed at offering an environment in which various remote devices are connected through the Internet. These devices should be able to collect, process, and share data. The IoT refers to the next generation of engineered systems that require tight integration of computing, communication, and control technologies to achieve stability, performance, reliability, robustness, and efficiency in dealing with physical systems of many application domains.

One of the main challenges in cloud-edge environments is to find an efficient task scheduling algorithm that is able to minimize completion time and cost simultaneously, as two contradictory objectives. In this paper, we address the workflow scheduling problem in cloud-edge environments, which is NP-hard by its nature. The solution space grows exponentially with an increase in the number of computing tasks. Thus, it becomes virtually impossible to find an optimal scheduling scheme [40]. Therefore, since in such scenarios classical deterministic algorithms cannot generate satisfying solutions within a reasonable period of time, different intelligent heuristic and metaheuristics approaches can be employed for finding solutions to this problem.

The main motivation behind the research proposed in this manuscript is to implement an efficient algorithm for workflow scheduling in hybrid cloud-edge environments. The basic assumption behind the conducted research is that solving an important challenge of workflow scheduling in cloud-edge can be further improved by using enhanced metaheuristics.

To efficiently address this challenge, we devised and implemented an enhanced firefly algorithm (FA). The FA is well-known and widely applied approach from the swarm intelligence family, that is able to successfully tackle a variety of NP-hard problems. The FA has proven as a robust optimizer and was successfully validated against many standard benchmarks [44], and practical problem instances [45].

Following the no free lunch theorem, an universal algorithm that solves all possible optimization problems does not exist. We have tested a considerable amount of famous swarm intelligence algorithms, both on this particular problem and on other different optimization problems in our previous research. FA metaheuristics has proven to show good exploitation capabilities, and in most cases it achieves solid performances. Based on our previous research, we have seen that FA can be further improved by utilizing different mechanisms, and that was the main motivation why we have selected it for solving this particular problem.

However, since the original implementation of the FA expresses some deficiencies in terms of insufficient exploration and inadequately established balance between intensification and diversification, which are elaborated in this paper, we devised an improved FA approach by introducing genetic operators and a quasi-reflection-based learning mechanism in its basic version.

According to the above statements, we formulated the following research question: *"Is it viable to establish further improvements in addressing workflow scheduling in a hybrid cloud-edge environment by using improved state-of-the-art FA metaheuristics"*. Based on the research question, the objectives, as well as contributions, of the conducted research are defined as follows:

- Devising enhanced version of the FA metaheuristics that outperforms basic implementation in terms of the quality of solutions and convergence speed and
- Improvements in solving workflow scheduling challenges in a cloud-edge environment by establishing efficient scheduling strategy based on the improved FA approach that obtains better values of performance indicators than strategies that are based on other state-of-the-art heuristics and metaheuristics.
- Validation of the efficiency of the proposed scheduling strategy throughout comprehensive simulations with different workflow samples, that can prove the superior performances of the proposed approach in comparison with other cutting-edge methods.

Following the most commonly used practice from modern computer science, the devised improved FA was firstly validated against a set of 10 modern CEC19 benchmarks and compared with the original FA and 9 other state-of-the-art metaheuristics. Afterward, the devised improved FA was utilized to solve the workflow scheduling task in cloud computing, that belongs to the group of classical NP-hard optimization challenges in computer science. The optimization of the workflow scheduling requires simultaneous accomplishment of several mutually conflicting goals, including the cost and makespan reduction. Consequently, 4 simulations for practical workflow scheduling in a cloud-edge environment were conducted and the performance of our improved FA was compared against the original FA, as well as with 8 other state-of-the-art metaheuristics and heuristics.

The remainder of the manuscript is structured as follows. In Sect. 2, the background of task scheduling and swarm intelligence along with the relevant literature review is presented. The mathematical formulation of the workflow scheduling model in the cloud-edge environment that was used in simulations is shown in Sect. 3. The original FA approach was described in Sect. 4, while a detailed description of our proposed enhanced FA's inner workings along with elaborated deficiencies of original FA is shown in Sect. 5. Section 6 is divided into two parts. In the first part, conducted empirical simulations for modern benchmarks are shown, while the second part provides details regarding the experiments conducted for practical workflow scheduling challenge. In both parts, a comparative analysis with other approaches is included. Finally, Sect. 7 concludes this manuscript and provides possible directions for future research in this prominent domain.

2 Background and related work

This section first provides the background for task and workflow scheduling in cloud systems. This is followed by a brief overview of the nature inspired metaheuristics and swarm intelligence, where the selected FA approach belongs to. Finally, this section is concluded by the survey of practical swarm intelligence solutions in the cloud/edge systems that are relevant for the research proposed within this paper.

2.1 Task scheduling

The objective of the task scheduling algorithm in cloud/-cloud-edge environments is to minimize the performance indicators (makespan, mean response time, energy consumption costs, etc.) of the system to provide on-demand and timely satisfaction of end-users' need for resources. Some of the simplest methods from this category include first come first served (FCFS), last come first served (LCFS), min-min, and join the shortest queue (JQS) policy. These methods can obtain satisfying results in tackling this challenge; for example, it is shown that JSQ, which assigns a new job to the server with the fewest tasks, in combination with FCFS minimizes the mean delay of the cloud system [22].

List-scheduling heuristics creates a list of tasks by assigning them some priority value. After that, the task with the highest priority is scheduled on the processor that allows the earliest start time. This procedure is repeated until the task-list is empty [12]. The earliest time first (ETF) algorithm adopts the strategy of selecting the task with the shortest start time on all processors. The starting time is determined by several factors: the completion time of preceding tasks, communication delay, and allocation of the task and its predecessors. The heterogeneous earliest-finish-time (HEFT) algorithm assigns a task from the list to the processor based on the task's earliest estimated finish time.

2.2 Nature inspired and swarm intelligence metaheuristics

Many real-world challenges, such as cloud/cloud-edge scheduling, belong to the group of NP-hard problems that cannot be solved in polynomial time by utilizing any of the traditional (classical) deterministic methods. However, in this case, stochastic approaches, which include metaheuristics, by finding an approximate solution within a satisfactory reasonable time, can be applied.

Metaheuristics that have been inspired by nature (bio-inspired, nature-inspired) can be divided into two distinctive groups. Algorithms that belong to the first group are known as evolutionary algorithms (EA), while the second group are swarm intelligence algorithms. One famous representative of the EA is a genetic algorithm (GA), which has been applied to a large number of real-life NP-hard problems, including load balancing and scheduling challenges from the cloud computing paradigm [42].

On the other hand, swarm intelligence algorithms are inspired by social behavior, collaboration, and communication among groups of relatively primitive individuals such as bees, ants, bats, fish, and fireflies. Swarm intelligence approaches were utilized with great success in solving a broad range of real-world problems in practice, including prolonging the network lifetime, clustering, and sensor localization in wireless sensor networks [4, 48, 50], cloud optimization problems [2, 8, 10], artificial neural networks optimization [7, 20, 27, 36], machine learning-based COVID-19 cases prediction [49, 52], and MRI classification optimization to name a few [5, 9].

2.3 Swarm intelligence applications in the cloud scheduling domain

In the following few paragraphs, some of the selected swarm intelligence applications for cloud/cloud-edge scheduling are briefly shown. Research published in [26] proposes a novel task scheduling algorithm based on bacterial foraging optimization (BFO), with the goal to reduce the idle time of virtual machines. The proposed method obtained excellent results. In [33], the authors presented a quantum-behaved PSO (QPSO) algorithm for minimizing job completion time (makespan). The obtained results proved that, when compared to other well-known algorithms, this approach is effective in reducing makespan due to the small value of the observed relative deviation.

Xie et al. in their work [46] implemented a novel directional and non-local-convergent particle swarm optimization (DNCPSO) algorithm for solving workflow scheduling in a cloud-edge environment. By using DNCPSO, the authors were able to dramatically reduce the task completion time and cost. The results show that this method can obtain better performance than other PSO variants, as well as other state-of-the-art heuristics and metaheuristics methods. Some of the other swarm algorithms that were adopted for cloud computing scheduling include whale optimization algorithm (WOA) [34], elephant herding optimization (EHO) [35], and artificial flora (AF) [3].

Cloud workflow scheduling problem was also recently targeted by metaheuristics approaches. An improved variant of Harris' hawks optimization (HHO) was used in [51] to solve the workflow scheduling in cloud-edge systems with very promising results. The experiments were executed by using CloudSim environment and benchmark workflow models with a goal to minimize cost and makespan. Research presented in [28] used improved chaotic binary variant of gray wolf optimizer (GWO) to improve the workflow scheduling in green cloud systems. The proposed approach was tested with CloudSim with a goal to minimize cost, makespan, and energy consumption. Another approach based on the self-adaptive fruit fly algorithm (SA-FFOA) was utilized in [1]. The proposed method was tested against traditional metaheuristics approaches, and experimental findings have shown superior performance in terms of cost and flow time. Finally, a whale optimizer (WOA) variant was used in another recent publication [37] to tackle the workflow scheduling. The findings indicate that the WOA-based approach outperformed other traditional methods for both makespan and overall cost.

The are several technical gaps that were observed in existing works related to workflow scheduling, and that inspired the design proposed in this paper. The geographical distribution of data, the emerging cloud-edge technology, heterogeneous data, and the efficient management of distributed data pose an important set of challenges even for large companies with a lot of resources at their disposal [18]. The enormous amount of workload must be processed, with the request to accomplish multiple often conflicting criteria, such as load balancing, cost reduction, overall execution time reduction, energy efficiency and green operation (reduction in the carbon pollution) [17, 19]. These challenges must be tackled with the development of the new algorithms tailored to address multiple opposing goals while keeping in mind scalability as one of the priorities.

3 System model formulation

The general structure of cloud-edge computing is formed by three distinctive layers. The end-user layer consists of smart IoT devices (human wearable devices, phones, etc.) that send requests for service to the devices in the edge-layer. Edge devices can process time-sensitive tasks and reduce latency to the minimum. On the top side, the edge-layer is connected to the cloud system which contains the infrastructure for providing on-demand resource availability and data storage.

Edge cloud computing has undoubtedly offered numerous advantages; however, it still has serious challenges that need to be addressed. The first challenge in such a complex environment, consisting of numerous resources of different types, is scheduling business tasks. This challenge typically considers execution in real-time, together with a huge amount of flowing data in this complex environment. Therefore, it is required to find a scheduling approach that is effective in making all tasks completed on time, resulting in real-time execution. The second challenge considers processing business workflows, with the goal of complete execution of workflows by taking into account QoS (quality of service) requirements (for example cost and deadline).

Additionally, some tasks are flexible in terms of resource allocation, while others with real-time and location requirements are considered inflexible. Therefore, the scheduling algorithm must be able to efficiently determine resource allocation for every task, while minimizing the cost and required deadline for task completion. Workflow scheduling in the cloud-edge environment, which is the focus of research conducted within this paper, is considered by nature to be an NP-hard problem by nature.

As part of the research published in [46], the authors have provided a good example to illustrate the business workflow scheduling problem in a cloud-edge environment. They consider a video surveillance/object tracking (VSOT) application, under the scenario with two requirements: mobile and geo-distribution, respectively. Two main features that must be taken into account for different types of applications are real-time execution and transmission-intensive requirement.

In the context of the given example, workflow scheduling in the cloud-edge system can be formulated as follows—how to assign different type resources and different processing powers to the tasks belonging to workflow, in order to minimize the overall completion time and cost.

Generally speaking, a typical workflow contains multiple repeatable business activities which usually provide services for business applications. It is a sequence of several different tasks, which can be represented in the form of a directed acyclic graph (DAG). Therefore, the DAG model can be directly utilized to represent a workflow application, as shown in the example given in Fig. 1. Every task is represented as a node in DAG, while every dependency relation is denoted with an edge.

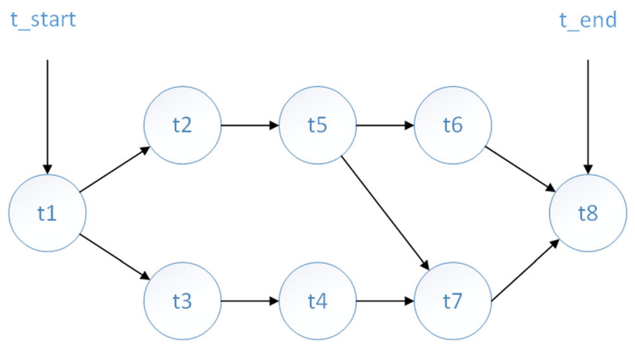

Fig. 1 A simple DAG of a typical workflow application

DAG can be formally defined as $G = (T, E)$, where T denotes a collection of tasks, and E stands for the set of temporal dependencies and communication constraints between individual tasks. Every task in the set of tasks, $t_s \in T(T = \{t_1, t_2, \ldots, t_n\})$, has a specified computational workload cw_s. The edges in DAG are directed, and edge $e_{ij} = (t_i, t_j)$ denotes that the task t_j must be executed after the task t_i has been completed. Additionally, edge e_{ij} has an assigned weight, cv_{ij}, which is a non-negative value that represents the amount of data transfer from the task t_i to the task t_j.

The observed VSOT scenario given in [46] assumes numerous business workflows that result in a specific set of requirements needed to be addressed, including a real-time response, a deadline for the completion of tasks, a balanced usage of different cloud and edge resources, etc. Therefore, the proposed model must take into account makespan, cost, and objective function.

In the given context, makespan can be defined as the maximum time for execution of the observed workflow application. In the observed scenario, the cloud-edge resources can be separated into two distinctive groups, namely cloud servers, and edge servers. Each task belonging to the observed workflow can be executed on both types of servers, based on the chosen resource allocation strategy. When a task t_s has been allocated to a specific server, its execution time is calculated as given in Eq. (1):

$$T^l_{t_s} = \frac{cw_s}{\delta_l} \tag{1}$$

where δ_l denotes the processing rate of the server l and cw_s represents the computational workload. If we take that $ct(e^l_{ij})$ is the time required for data transfer from the task t_i to the task t_j, it can be calculated as given in Eq. (2):

$$ct(e^l_{ij}) = \frac{cv_{ij}}{B} \tag{2}$$

In the previous equation, cv_{ij} specifies the amount of data transfer between tasks i and j, and B denotes the network bandwidth between the servers where tasks are allocated. Every task t_s is allocated to exactly one server; it has the start time ST_{t_s}, given with Eq. (3), and finish time FT_{t_s}, calculated with Eq. (4).

$$ST_{t_s} = max\{FT_{t_p} + ct(e^l_{ps}), t_p \in pre(t_s)\} \tag{3}$$

$$FT_{t_s} = ST_{t_s} + T^l_{t_s} \tag{4}$$

In Eq. (3), $pre(t_s)$ marks the collection of tasks that are direct predecessors of the observed task t_s. When observing an application workflow as a whole, the total amount of time required for workflow completion can be calculated as the time from the start of the first task in the workflow, until the completion of the last task, as given in Eq. (5):

$$T_{\text{total}} = \max\{FT_{ts}, t_s \in T\} \tag{5}$$

The second important parameter, namely the cost, can be considered twofold in the given context. For both cloud and edge resources, computational cost and communication cost can be identified. The first type of cost, the computational cost from the task t_i to the task t_j on the server l, can be calculated by using Eq. (6):

$$C^l_{t_s} = pp_l \cdot (FT_{t_s} - ST_{t_s}) \tag{6}$$

where pp_l represents the price for the processing unit on the observed server l.

The second type of cost, communication cost from the task t_i to the task t_j on the server l, can be calculated by using Eq. (7):

$$cc(e^l_{ij}) = cp_l \cdot ct(e^l_{ij}) \tag{7}$$

where cp_l represents the price of the communication unit on the observed server l.

The total cost can be calculated as the sum of computational and communication costs by using Eq. (8):

$$C_{\text{total}} = \sum_{l=1}^{m} \sum_{s=1}^{n} C^l_{t_s} + \sum cc(e^l_{ij}) \tag{8}$$

Finally, after defining the calculation method for both the makespan and the cost, it is possible to define the combined objective function that takes into account the makespan and the total cost, in the observed cloud-edge system, as given in Eq. (9):

$$f = \kappa T_{\text{total}} + (1 - \kappa) \cdot C_{\text{total}} \tag{9}$$

As it can be seen from Eq. (9), the weighted sum approach is used for defining the objective function. The value κ is a coefficient between 0 and 1, and it is used to determine the relative weights of each objective. The goal is to find the function value f that minimizes these two main components, therefore helping to achieve the balance (trade-off) between the makespan and the cost for the observed workflow scheduling problem.

4 Original FA

The firefly algorithm (FA) was first introduced by Yang in [44]. This well-known algorithm was inspired by the fireflies, more precisely, their social behavior and flashing characteristics. Since the real firefly ecosystem is very complex and sophisticated, the FA metaheuristic is its approximation.

The actual formulation of attractiveness is a crucial challenge that should be addressed in the implementation of the FA. Typically, the attractiveness of the firefly depends on its brightness, which in turn depends on the objective function. For minimization problems, the brightness of the firefly on the location x can be calculated by using Eq. (10) [44]:

$$I(x) = \begin{cases} \frac{1}{f(x)}, & \text{if } f(x) > 0 \\ 1 + |f(x)|, & \text{otherwise} \end{cases} \tag{10}$$

where $I(x)$ denotes the attractiveness, and $f(x)$ represents the value of the objective function at the location x. It is clear that Eq. (10) represents fitness functions.

If we observe mathematical properties, both light intensity and attractiveness are monotonously decreasing functions, as the distance from the source of light increases, the light intensity, and the attractiveness will drop. This can be expressed with Eq. (11):

$$I(r) = \frac{I_0}{1 + \gamma r^2} \tag{11}$$

where $I(r)$ represents the intensity of light, r denotes the distance, and I_0 stands for the light intensity at the source location. Additionally, light is partially absorbed by the surrounding air, causing light to become weaker. These phenomena can be modeled by the light absorption coefficient γ.

According to the literature survey, almost all implementations of the FA take into account the combined impact of both the inverse square law for the distance and absorption of light, by approximating them with the following Gaussian form [44]:

$$I(r) = I_0 \cdot e^{-\gamma r^2} \tag{12}$$

Attractiveness β of an individual firefly is relative depending on the distance between the observer and the target firefly. On the other hand, the attractiveness is directly proportional to the light intensity of the given firefly, which can be calculated with Eq. (12):

$$\beta(r) = \beta_0 \cdot e^{-\gamma r^2} \tag{13}$$

where the parameter β_0 denotes the attractiveness at a distance $r = 0$. Eq. (13) also helps in determining the characteristic distance $\Gamma = 1/\sqrt{\gamma}$ where the attractiveness of the firefly changes significantly from β_0 to $\beta_0 e^{-1}$. However, in practice, Eq. (13) is often replaced by Eq. (14) [44]:

$$\beta(r) = \frac{\beta_0}{1 + \gamma r^2} \tag{14}$$

The arbitrary firefly i moves to the new position (in the next iteration $t + 1$) in the direction of the brighter, more attractive firefly j, which can be modeled as:

$$x_i^{t+1} = x_i^t + \beta_0 \cdot e^{-\gamma r_{i,j}^2}(x_j^t - x_i^t) + \alpha^t(\kappa - 0.5) \tag{15}$$

where para

$$x_i^{t+1} = x_i^t + \beta_0 \cdot e^{-\gamma r_{i,j}^2}(x_j^t - x_i^t) + \alpha^t(\kappa - 0.5) \tag{16}$$

where parameter β_0 denotes the attractiveness at a distance $r = 0$, α marks a randomization parameter, κ represents a random number drawn from either uniform or Gaussian distribution, and meter β_0 denotes the attractiveness at a distance $r = 0$, α marks a randomization parameter, κ represents a random number drawn from either uniform or Gaussian distribution, and $r_{i,j}$ represents the distance between two observed fireflies i and j. The positions of fireflies are updated sequentially, by comparison and update of each pair of fireflies in each iteration.

The distance between two observed fireflies i and j can be calculated by using the Cartesian distance, as shown in Eq. (17):

$$r_{i,j} = ||x_i - x_j|| = \sqrt{\sum_{k=1}^{D}(x_{i,k} - x_{j,k})^2} \tag{17}$$

where D marks the number of parameters for the given problem. The appropriate values for most problems are $\beta_0 = 1$ and $\alpha \in [0, 1]$.

5 Improved FA metaheuristics

In this section, we first describe the drawbacks of the original FA metaheuristics. Afterward, we justify and provide details of mechanisms that we incorporated in the basic FA to overcome the observed deficiencies. Finally, we present the implementation details and inner workings of the proposed improved FA version.

5.1 Drawbacks of the basic FA

It can observed that in early iterations the basic FA algorithm may be stuck in sub-optimal search space domains due to the lack of exploration power. As a consequence of this deficiency, in some runs, the search process converges to under optimum solutions and the mean results' quality is not satisfying.

The basic FA version does not employ a clearly emphasized diversification equation. The balance between intensification and exploration is established by utilizing the randomization parameter (step size) α, as the third component of the FA's search equation (Eq. 16).

The value of parameter α has a great influence on the searching ability. If its value is larger, the global search (diversification) is more emphasized. In the original FA implementation, the value of α was fixed during the whole run time, taking values from the interval [0, 1] [44], and the distance between individuals that is decreasing in later iterations was not taken into account [23]. In some improved versions, this parameter is set to be dynamic by gradually decreasing during the course of execution [41]. However, the dynamic behavior of this parameter alone is not sufficient for establishing a balanced trade-off between exploitation and exploration, which is moved toward (in favor of) exploitation.

Moreover, on top of these deficiencies, the intensification process of the original FA can be further enhanced. The utilization of dynamic parameter α is not sufficient enough to overcome these deficiencies and some other mechanisms are needed.

5.2 Proposed improvements

To address the observed drawbacks of the original FA and to further improve intensification, our proposed upgraded implementation incorporates the following in its basic version:

- Genetic operators—uniform crossover and Gaussian mutation and
- Quasi-reflection-based learning mechanism.

All proposed changes address both—the absence of explicit diversification and the inappropriate balance between intensification and diversification of the basic FA, and also make the exploitation process more efficient.

Genetic operators help in improving exploration in early iterations of the algorithm's run when the search process did not converge to the region where an optimum solution resides. This has direct implications for the convergence (mean values). On the contrary, in later iterations, with the premise that the promising domain of the search space is identified, applying genetic operators enables fine exploitation around the current best solutions and this influences the quality of the final result (the best solution).

According to previous researches, by incorporating quasi-reflection-based learning in metaheuristics, both solutions diversity and convergence rate can be improved [16]. In the initialization phase, as well as in early iterations of a run, quasi-reflection-based learning may increase population diversity, while at the later phases, this mechanism enhances convergence speed.

Moreover, we have also utilized a dynamically adjusted step size parameter α. In early iterations, this parameter is larger and is directed toward the global search (exploration). As iterations progress, the value of this parameter decreases and is more oriented toward exploitation (local search).

Inspired by included upgrades, proposed improved FA metaheuristics is named genetic operators quasi-reflected FA (GOQRFA).

5.2.1 Genetic operators

As noted, in our proposed GOQRFA we have incorporated uniform crossover and Gaussian mutation genetic operators to efficiently combine existing and/or random solutions with the goal of improving exploration and exploitation processes. With that in mind, we introduce terminology from the GA in our approach.

The GA approach represents each potential problem solution as a chromosome, which is further divided into genes. For example, an objective function that should be minimized/maximized is represented as a chromosome, while each parameter denotes one gene. In our approach, we have applied the probability of crossover and mutation on the gene level (gene probability—GP).

In our approach, we utilized a uniform crossover operator. In this type of crossover, each gene (solution parameter) with a probability p will be exchanged between two parent solutions. If the value of p is close to 0.5, then the gene exchange will be frequent and the uniform crossover will show global (exploratory) behavior.

However, if the value of p is closer to 0 or 1, the fewer number of genes will be exchanged between two parents and the uniform crossover will exhibit local tendency (exploitation).

In our proposed approach, by applying uniform crossover to each pair of parents, only one offspring solution is created. Here, we also adapt different terminology, and call offspring a hybrid solution. For example, if two best solutions (x_{best1} and x_{best2}) from the population are combined, each parameter j of the hybrid solution (x_{hyb}) is calculated in the following way:

$$x_{hyb,j} = \begin{cases} x_{best1,j}, & \text{if } \phi \leq p \\ x_{best2,j}, & \text{otherwise} \end{cases} \tag{18}$$

where ϕ is a pseudo-random number drawn from the uniform distribution.

When such a hybrid solution is generated, it is being subdued to mutation. Among various types of mutations used in methods from modern literature, uniform, polynomial, and Gaussian are considered to be classical ones [13]. Since the Gaussian mutation operator proved to be very efficient for tackling NP-hard problems by preventing the loss of diversity during the search process, we used this operator to overcome deficiencies of the original FA.

Gaussian mutation, that was first proposed by Bӓck and Schwefel [6], is based on the Gaussian density function defined as:

$$f_{\text{gaussian}(0,\sigma^2)}(\theta) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\theta^2}{2\sigma^2}} \tag{19}$$

where the variance of candidate solutions is denoted as σ^2.

Gaussian density function can be reduced for the mean of 0 and the standard deviation σ of 1. In the context of our proposed approach, Gaussian distributed random vector $G(\theta)$ can be generated and applied for each parameter j of the solution h_{hyb}:

$$x_{\text{hyb,j}} = x_{\text{hyb,j}} \cdot (1 + G(\theta_j)) \tag{20}$$

5.2.2 Quasi-reflection-based learning

The basic assumption that inspired us to incorporate this modification in the original FA is that when generating new solutions by using opposite numbers, there is more chance that such solutions will be close to the optimum region, than the random ones. The idea is to generate the opposite solution of the feasible solution created by using the metaheuristics search process, to evaluate the opposite solution, and to select the better one.

The first mechanism that supports this logic is opposition-based learning (OBL) introduced in [38]. The OBL proved to be an efficient technique that can substantially enhance the metaheuristics search process.

Let x_j denotes j-the parameter of the solution x and the x_j^o represents its opposite number. The opposite number of the j-th parameter of an individual x can be calculated as follows:

$$x_j^o = lb_j + ub_j - x_j \tag{21}$$

where $x_j \in [lb_j, ub_j]$ and $lb_j, ub_j \in R, \forall j \in 1, 2, 3, ...D$. Notations lb_j and ub_j represent lower and upper bound of the j-th parameter, respectively, and D denotes the number of solution dimensions (parameters).

One instance of OBL that also proved to be very effective is quasi-opposition-based learning (QOBL), introduced by [32]. Research work presented in the modern literature proved that quasi-opposite numbers can be more effective in tackling NP-hard continuous problems than the opposite numbers. A quasi-opposite number x_j^{qo} of x_j can be expressed as:

$$x_j^{qo} = \text{rnd}\left(\frac{lb_j + ub_j}{2}, x_j^o\right) \tag{22}$$

where $\frac{lb_j + ub_j}{2}$ represents the arithmetic mean (the center) of the interval $[lb_j, ub_j]$, while $\text{rnd}\left(\frac{lb_j + ub_j}{2}, x_j^o\right)$ generates uniformly distributed pseudo-random from the interval $\left[\frac{lb_j + ub_j}{2}, x_j^o\right]$.

Finally, an improved mechanism called quasi-reflection-based learning (QRBL) was proposed recently by [15], based on the QOBL principles.

The quasi-reflected component j of the solution x (x_j^{qr}) can be determined as a reflection of x^{qo} in the following way:

$$x_j^{qr} = \text{rnd}\left(\frac{lb_j + ub_j}{2}, x_j\right) \tag{23}$$

where $\frac{lb_j + ub_j}{2}$ represents the arithmetic mean (center) of the interval $[lb_j, ub_j]$, while $\text{rnd}\left(\frac{lb_j + ub_j}{2}, x_j\right)$ generates uniformly distributed pseudo-random from the interval $\left[\frac{lb_j + ub_j}{2}, x_j\right]$.

5.2.3 Dynamic parameter α

As already pointed out, with the larger parameter α in the FA's basic search equation (Eq. (16)), the global search (exploration) dominates. As the algorithm converges to the optimum regions during iterations, the value of this parameter should decrease with the goal of gradually redirecting the search from global to local (exploitation).

Due to this reason, we have adopted a dynamic step α from [41], which gradually decreases from its initial value α_0, until it reaches the minimum threshold $\alpha_{\min}$ as iterations advance, in our approach:

$$\alpha^{t+1} = \alpha^t \cdot \left(1 - \frac{t}{\text{MaxIter}}\right) \tag{24}$$

where t and $t+1$ denote current and next iterations, respectively, while the *MaxIter* is the maximum iteration number in one run of an algorithm.

5.3 Working details of proposed GOQRFA

Taking into account the improvements of the original FA metaheuristics showed in Sect. 5.2, details of the proposed GOQRFA can be given.

First, the exploration–exploitation trade-off is enhanced by using uniform crossover and Gaussian mutation genetic operators. However, in early iterations, exploration should be amplified, while in later cycles, with the assumption that search has converged to optimum region, fine-tuned exploitation around the current best solutions should be dominant.

For that reason, we employed two different uniform crossover mechanisms—one for exploration and one for exploitation. We introduced two additional control parameters: the number of replaced solutions *nrs*, which is used for adjusting the number of worst solutions from the population that are replaced with hybrid solutions, and exploration break point (*ebp*) that controls which of the above-mentioned crossover mechanisms will be triggered.

In the first *ebp* iterations, the diversification uniform crossover (*DUC*) mechanism is triggered in the following way: *nrs* worst solutions from the population are replaced with the hybrid solutions generated by performing recombination between the completely random solution (x_{rnd}) from the search domain and randomly chosen existing solutions from the population (x_{prnd}) by using Eq. (18) with probability p on the gene level. A completely random solution is created in the same way as in the initialization phase (Eq. 28). If the $nrs > 1$, for each worst solution, a new pair of completely random and random existing solutions from the population is chosen. The *DUC* mechanism is executed in each iteration.

After *ebp*, with the goal of improving exploitation around the current best solutions, intensification uniform crossover (*IUC*) mechanism is triggered as follows: *nrs* worst solutions from the population are replaced with the hybrid solutions generated by performing recombination between the first best ($x_{\text{best}1}$) and the second-best ($x_{textbest2}$) solutions from the population by utilizing Eq. (18) with uniform crossover probability p on the gene level. If $nrs > 1$, then for each replaced solution, the new hybrid is generated. By conducting extensive empirical simulations, we have observed that if the *IUC* is triggered too frequently, the population may lose diversity and may converge to sub-optimal solutions. As the iterations progress, the *IUC* frequency should increase because the algorithm is progressing toward the optimum region of the search space. To control this behavior, in each iteration, the *IUC* is executed with probability *IUCp* (*IUC* probability), which is increasing directly proportionally to the current iteration from the initial value $IUCp_0$. We modeled this behavior similarly as in the case of dynamic parameter α:

$$IUCp^{t+1} = IUCp^t \cdot \left(1 + \frac{t}{MaxIter}\right) \tag{25}$$

In both cases (*IUC* and *DUC*), Gaussian mutation on the gene level (gene probability - GP) is performed for each hybrid solution by applying Eq. (20). However, different mutation probabilities are applied for each gene (solution's parameter)—mutation probability for diversification (*mpd*) and mutation probability for intensification (*mpi*) for hybrids generated during *DUC* and *IUC* phases, respectively. Since stronger exploration is required in the first *ebp* iterations, the *mpd* is higher than *mpi*. We have empirically determined best values for *mpd* and *mpi*, as shown in Eqs. (26) and (27), respectively.

$$mpd = \frac{1}{D} \tag{26}$$

$$mpi = \frac{1}{2D} \tag{27}$$

where D represents the number of solutions' parameters (gene numbers in the context of GA).

Moreover, to further improve solutions' diversity and convergence speed we incorporated the QRBL strategy. This procedure is first added in the initialization phase to improve the diversity of the initial population. The QRBL is also introduced in the solutions' position update phase with the goal of enhancing the search process convergence rate.

In the initialization phase, each solution parameter j for every solution x_i from the initial population ($P_{init} = \{X_{i,j}\}, i = 1,2,3\ldots,NS; j = 1,2,,\ldots D$) is generated by using standard initialization expression [39]:

$$x_{i,j} = lb_j + (ub_j - lb_j) \cdot \text{rand} \tag{28}$$

where rand is an uniformly distributed random number from the interval [0, 1].

Then, the QRBL is applied (Eq. 23) to determine quasi-reflective solution of each individual from the population, and quasi-reflective initial population ($P^{qr}_{init} = \{X^{qr}_{i,j}\}, i = 1, 2, 3..., NS; j = 1, 2, , ...D$) is generated. Both populations are then merged together ($P_{init} \cup P^{qr}_{init}$) and sorted in descending order according to the fitness value and *NS* best solutions are selected as the new initial population.

Similarly, during the update phase, in each iteration, quasi-reflective population P^{qr} of updated population *P* is created. Also, like in the initialization phase, populations are then merged ($P \cup P^{qr}$) and sorted in descending order according to the fitness value, and *NS* best solutions are selected to propagate to the next iteration.

Pseudo-code for generating quasi-reflective population in the update phase is given in Algorithm 1. The same can be applied for creating P^{qr}_{init}.

Algorithm 1 Updating phase by using QRBL mechanism

```
Generate updated population $P$ by employing the standard FA search and genetic operators
for $i = 1$ to $NS$ do
    for $j = 1$ to $D$ do
        if $x_{i,j} < \frac{lb_j + ub_j}{2}$ then
            $x^{qr}_{i,j} = x_{i,j} + \left(\frac{lb_j + ub_j}{2} - x_{i,j}\right) \cdot rand$
        else
            $x^{qr}_{i,j} = \frac{lb_j + ub_j}{2} + \left(x_{i,j} - \frac{lb_j + ub_j}{2}\right) \cdot rand$
        end if
    end for
    Add solution $x^{qr}_{i,j}$ to $P^{qr}$
end for
Merge $P$ and $P^{qr}$ $(P \cup P^{qr})$
Select $NS$ best solutions
```

The basic search procedure is the same as in the original FA metaheuristics (Eq. 16), as described in Sect. 4.

Control parameters of proposed GOQRFA are summarized in Table 1. Besides parameter description and notation, in the third column, we show parameter origin (FA standard or GOQRFA specific) and type (static or dynamic).

High-level steps of proposed GOQRFA can be summarized as follows:

1. Initialize main metaheurisitcs parameters, search space and GOQRFA control parameters
2. Generate random population P_{init} and apply QRBL mechanism to create quasi-reflective population P^{qr}_{init}. Select best *NS* solutions from $P_{init} \cup P^{qr}_{init}$ and form population *P*
3. Perform solutions' update phase in population *P* by using the FA search equation
4. Execute *DUC* or *IUC* mechanisms and replace *nrs* worst solutions from the population with hybrid individuals
5. Generate quasi-reflective population P^{qr} from the current population *P*, merge two populations and select the best *NS* individuals
6. Output the best solution if the number of iterations reaches *MaxIter*, otherwise return to (3)

Detailed pseudo-code of proposed GOQRFA metaheuristics is given in Algorithm 2, while the visual representation of GOQRFA steps in the form of a flowchart diagram is depicted in Fig. 2.

Table 1 GOQRFA control parameters summary

Parameter Description	Notation	Type
Number of solutions in population	*NS*	FA standard (static)
Maximum iteration number	*MaxIter*	FA standard (static)
Absorption coefficient	γ	FA standard (static)
Attractiveness parameter at $r = 0$	β_0	FA standard (static)
Randomization (step) parameter	α	FA standard (dynamic)*
Initial value of step parameter	α_0	FA standard (static)
Minimum value of step parameter	α_{min}	FA standard (static)
Exploration break point	*ebp*	GOQRFA specific (static)
Number of replaced solutions	*nrs*	GOQRFA specific (static)
Uniform crossover probability	*p*	GOQRFA specific (static)
Intensification uniform crossover probability	*IUCp*	GOQRFA specific (dynamic)**
Initial value of intensification uniform crossover probability	$IUCP_0$	GOQRFA specific (static)
Mutation probability for diversification	*mpd*	GOQRFA specific (fixed)
Mutation probability for intensification	*mpi*	GOQRFA specific (fixed)

*Changes according to Eq. (24)

**Changes according to Eq. (25)

Algorithm 2 The GOQRFA pseudo-code

```
Initialize main metaheuristics control parameters $NS$ and $MaxIter$
Initialize search space parameters $D$, $ub_j$ and $lb_j$
Initialize GOQRFA control parameters $\gamma$, $\beta_0$, $\alpha_0$, $\alpha_{min}$, $nrs$, $ebp$, $p$, $IUC_{p0}$, $mpd$ and $mpi$
Generate initial random population $P_{init} = \{X_{i,j}\}, i = 1,2,3...,NS; j = 1,2,,...D$ using Eq. (28) in the search space
Generate quasi-reflective initial population $P^{qr}_{init}$ by using QRBL strategy according to Algorithm 1
Intensity of light $I_i$ (fitness) at position $x_i$ is defined by $f(x)$
Create population $P$ by merging populations $P_{init}$ and $P^{qr}_{init}$, calculate fitness of all solutions, sort individuals in descending order according to quality and select the best $NS$ individuals
while $t < MaxIter$ do
    for $i = 1$ to $NS$ do
        for $k = 1$ to $i$ do
            if $I_k < I_i$ then
                Move solution $k$ in the direction of individual $i$ in $D$ dimensions (Eq. (16))
                Attractiveness changes with distance $r$ as $\exp[-\gamma r]$ (Eq. (13))
                Evaluate new solution, replace the worse individual with the better one and update intensity of light (fitness)
            end if
        end for
    end for
    if $t \leq ebp$ then
        Execute $DUC$ mechanism and replace $nrs$ worst solutions with hybrid between $x_{rnd}$ and $x_{prnd}$ individuals using Eq. (18) and apply Gaussian mutation (Eq. (20)) with $mpd$ probability
    else
        Execute $IUC$ mechanism with probability $IUC_p(t)$ and replace $nrs$ worst solutions with hybrid between $x_{best1}$ and $x_{best2}$ individuals using Eq. (18) and apply Gaussian mutation (Eq. (20)) with $mpi$ probability
    end if
    Generate quasi-reflective population $P^{qr}$ by using the QRBL strategy according to Algorithm 1
    Merge populations $P$ and $P^{qr}$, calculate fitness of all solutions, sort individuals in descending order according to quality and select best $NS$ individuals
    Update dynamic parameters $\alpha$ and $IUC_p$ for next iteration $t+1$ according to eqs. (24) and (25), respectively
end while
Return the best individual $x_i$ from the population
```

5.3.1 Algorithm complexity

The number of objective function evaluations is usually taken to calculate swarm intelligence algorithm complexity [45]. In the basic FA algorithm, fitness is calculated in the initialization phase and in the solutions' updating phase. In the updating phase, the basic FA has one main loop for iterations t and two inner loops going through NS solutions [45].

Thus, including the initialization phase, the complexity in the worst case of the basic FA metaheuristics is $O(NS) + O(NS^2 t)$. However, if NS is relatively large, it is possible to use one inner loop by ranking the attractiveness or brightness of all fireflies using sorting algorithms, and in this case complexity is $O(NS) + O(NSt \log(NS))$ [45].

The complexity of the GOQRFA is higher than the original FA due to the application of the QRBL mechanism. The UCD and UCI mechanisms are not counted since in these phases, objective function evaluations are performed only nrs times in each iteration, and nrs is typically small (1 or 2). The QRBL is applied in the initialization phase and after the solution's update phase in each iteration, and in both cases, an additional NS number of function evaluations are performed. Therefore, the complexity of the proposed GOQRFA in the worst-case scenario can be expressed as: $2 \cdot O(NS) + O(NS^2 t) + O(SNt)$.

Also, it should be noted that the GOQRFA employs more control parameters than the basic FA. Therefore, it requires more time to obtain the suboptimal set of parameters. However, substantial performance improvements of GOQRFA over the original FA, as it is shown in Sect. 6, justify more control parameters.

6 Simulations and discussion

Following the good practice from modern computer science literature, the simulations, and experimental section is divided into two parts. In the first part, we show simulations on a modern set of ten CEC 2019 benchmarks, where we compare our proposed GOQRFA with the basic FA implementation and nine other state-of-the-art metaheuristics, as it was performed in [29] (Sect. 6.1). Later, in the second part of this section, we show the evaluation of proposed metaheuristics for business workflow scheduling in the heterogeneous cloud-edge environment by conducting several categories of experiments, similarly as in [46] (Sect. 6.2)

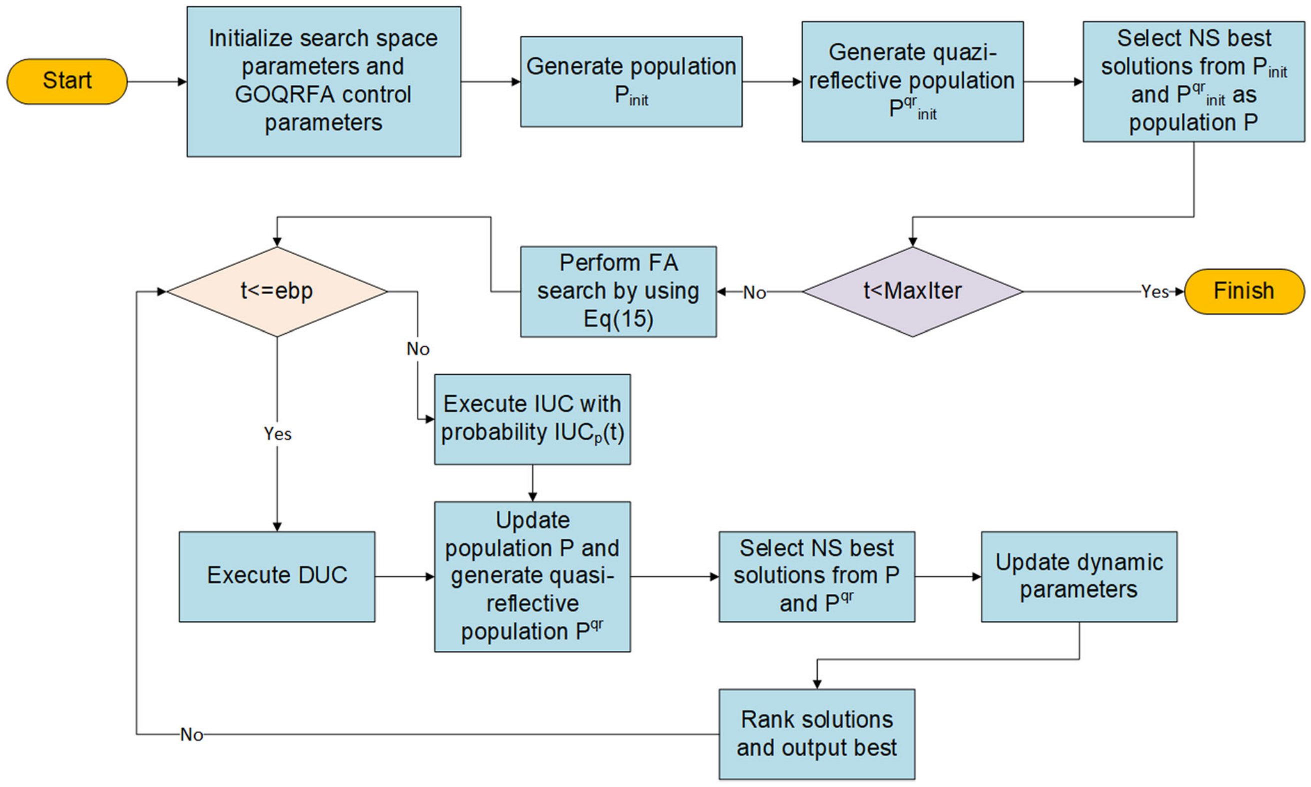
Start
Initialize search space parameters and GOQRFA control parameters
Generate population P_init
Generate quazi-reflective population P^qr_init
Select NS best solutions from P_init and P^qr_init as population P
t<MaxIter
Yes
Finish
No
Perform FA search by using Eq(15)
t<=ebp
No
Execute IUC with probability IUC_p(t)
Yes
Execute DUC
Update population P and generate quasi-reflective population P^qr
Select NS best solutions from P and P^qr
Update dynamic parameters
Rank solutions and output best
No

Fig. 2 Flowchart of proposed GOQRFA metaheuristics

To validate improvements of GOQRFA over the original FA, we have implemented and tested both metaheuristics for the purpose of this research. For each algorithm, we created two implementations—one in Python and one in Java. Moreover, with the goal of visualizing the results, we have employed data science Python libraries: scipy, pandas, pyplot, and seaborn. Unconstrained benchmarks' simulations were conducted with both implementations and generated results, which are reported in the experimental section, are the same. However, since the simulator used for cloud-edge scheduling experiments is implemented in Java, we have utilized only Java GOQRFA and FA implementations in these experiments.

We did not implement and tested metaheuristics and heuristics, that were taken for the comparative analysis, and its results were retrieved from [29] and [46] for unconstrained benchmarks and cloud-edge workflow scheduling comparative analysis, respectively.

All experiments were conducted on Intel® CoreTM i7-8700K CPU and 32GB of RAM running under Windows 10 x64 operating system computer platform.

6.1 CEC 2019 modern benchmarks simulations

The performances of the proposed GOQRFA approach were validated against the most recent and challenging benchmark test functions suite, known as CEC 2019 benchmarks [31]. This suite of benchmark functions was created with a goal to evaluate metaheuristics for single-objective optimizing problems. All ten provided functions are minimization problems.

Benchmark CEC01, CEC02 and CEC03 have the dimensionality of 9, 16 and 18, with boundaries $[-8192, 8192]$, $[-16384, 16384]$ and $[-4, 4]$, respectively. All remaining benchmarks have ten dimensions and the boundaries $[-100, 100]$, and they are shifted and rotated [31].

The performances of the suggested GOQRFA method on CEC 2019 test suite were evaluated and compared with the original FA metaheuristics, and nine other state-of-the art metaheuristics, namely EHOI, EHO, SCA, SSA, GOA, WOA, BBO, MFO and PSO, whose results were taken from the [29]. In order to provide fair comparative analysis, we have utilized the same setup as in [29]. For all metaheuristics, the initial population size was set to fifty, while the maximal amount of iterations was limited to 500. Every algorithm was executed 30 times on each benchmark. The

Table 2 GOQRFA control parameters value utilized in simulations

Parameter and notation	Value
Absorption coefficient γ	1.0
Attractiveness parameter at $r = 0$ β_0	1.0
Randomization (step) parameter α	Eq. (24)
Initial value of step parameter α_0	0.5
Minimum value of step parameter α_{min}	0.1
Exploration break point *ebp*	*MaxIter*/3
Number of replaced solutions *nrs*	1
Uniform crossover probability *p*	0.5
Intensification uniform crossover probability *IUCp*	Eq. (25)
Initial value of intensification uniform crossover probability $IUCP_0$	0.2
Mutation probability for diversification *mpd*	Eq. (26)
Mutation probability for intensification *mpi*	Eq. (27)

mean and standard deviation of fitness values are noted and presented in Table 3, where the best mean values are in boldface.

The GOQRFA control parameter values that were used in experiments are summarized in Table 2.

The results from Table 3 clearly indicate that the proposed GOQRFA metaheuristics achieved the best fitness value on most of the modern CEC 2019 benchmarks over 30 independent runs. Only in case of two functions, GOQRFA is the second best approach.

A statistical Friedman's nonparametric test was executed in order to prove the significant difference between the suggested GOQRFA, the basic FA and other metaheuristics algorithms that were included in the comparative analysis. This statistical test utilizes rank information of the data in order to verify significant differences, as stated in [21]. In accordance with the Friedman's test rank, the algorithms that have lower rankings are considered more efficient than the algorithms that have a higher ranking.

Table 4 summarizes the results of the applied Friedman test. It can be seen that the ranking of the proposed GOQRFA over different benchmark functions has small values, in comparison with the basic FA and other metaheuristics included in the analysis.

Additionally, Holm's step-down procedure is used as a post hoc procedure to analyze the significance level of the compared algorithms. The Holm's step-down procedure result is reported in Table 5, which shows that the proposed method outperforms all other comparable methods, at 0.05, as well as at 0.1 significance level.

Figure 3 presents the convergence graph of the original FA and the proposed GOQRFA algorithms on the 10 CEC 2019 benchmark functions. Based on the convergence graph, we can conclude that GOQRFA improves the convergence of the original FA.

6.2 Workflow scheduling in cloud-edge environment simulations

In this subsection, we first show the simulation environment and workflow models utilized in experiments along with the GOQRFA control parameters' setup. Afterward, we present the solutions encoding strategy and adaptations of GOQRFA for this practical challenge and discuss the weight coefficient of the makespan and the total cost objectives. Finally, we present a comparative analysis with other state-of-the-art heuristics and metaheuristics that were tested for the same workflow scheduling in cloud-edge environment problem instances and under the same experimental conditions.

6.2.1 Simulation environment, workflow models and parameters' setup

To validate the performance of the proposed GOQRFA on the real-world problem of workflow-scheduling in a cloud-edge environment and to make an objective comparative analysis with results of state-of-the-art heuristics and metaheuristics that were showed in [46], we used the same simulation environment and workflow models (datasets) as in this paper, that was published in an outstanding international journal.

To simulate a real-world distributed cloud-edge environment, we have employed WorkflowSim-1.0 in all experiments, which is a well-known open-source workflow simulator, created by the Pegasus WMS group at the University of Southern California [14]. This simulator was chosen as it generates an environment very similar to the real distributed system. Additionally, it provides numerous algorithms for clustering, task scheduling, provisioning of the resources, and so on, making it ideal for experiments

Table 3 Comparative analysis of the results achieved by the basic FA and proposed GOQRFA with other metaheuristics algorithms on 10 modern CEC 2019 benchmark functions

Function	Stats	EHOI	EHO	SCA	SSA	GOA	WOA	BBO	MFO	PSO	FA	GOQRFA
CEC01	Mean	4.69E+04	1.41E+07	9.54E+09	2.73E+09	1.53E+10	1.08E+10	3.40E+10	6.65E+09	8.67E+11	2.58E+05	**4.22E+04**
	Std	2.87E+03	7.78E+06	7.88E+09	2.58E+09	3.05E+10	8.72E+09	2.58E+10	8.51E+09	9.23E+11	5.15E+04	2.81E+0.5
CEC02	Mean	1.73E+01	1.73E+01	1.75E+01	1.73E+01	1.74E+01	1.73E+01	9.19E+01	1.73E+01	9.99E+03	3.81E+01	**1.28E+00**
	Std	1.18E-15	4.59E-15	4.22E-02	7.98E-05	1.49E-02	2.82E-03	2.68E+01	3.74E-15	3.84E+03	2.55E-01	1.31E+01
CEC03	Mean	1.27E+01	1.27E+01	1.27E+01	1.27E+01	1.27E+01	1.27E+01	1.27E+01	1.27E+01	1.27E+01	1.01E+01	**1.00E+0.1**
	Std	1.87E-15	1.87E-15	1.04E-04	2.37E-15	1.17E-04	1.44E-07	2.63E-07	3.48E-05	6.52E-04	4.71E-01	1.06E+00
CEC04	Mean	1.27E+01	1.52E+01	1.08E+03	3.36E+01	1.47E+02	3.10E+02	7.84E+01	1.34E+02	7.30E+01	3.62E+00	**2.66E+00**
	Std	3.95E+00	6.26E+00	3.91E+02	1.18E+01	1.98E+02	1.24E+02	2.64E+01	1.78E+02	7.77E+00	4.69E-01	1.01E+00
CEC05	Mean	1.04E+00	1.04E+00	2.19E+00	1.21E+00	1.34E+00	1.61E+00	1.28E+00	1.14E+00	1.53E+00	1.05E+00	**1.03E+00**
	Std	2.12E-02	2.22E-02	7.65E-02	1.15E-01	1.25E-01	4.04E-01	9.84E-02	7.99E-02	1.21E-01	1.51E-02	1.89E-02
CEC06	Mean	8.29E+00	9.52E+00	1.08E+01	3.69E+00	6.22E+00	8.98E+00	5.84E+00	5.30E+00	1.06E+01	1.75E+00	**1.15E+00**
	Std	8.19E-01	1.27E+00	7.42E-01	1.43E+00	1.29E+00	1.07E+00	6.48E-01	2.18E+00	6.69E-01	1.49E-02	5.07E-02
CEC07	Mean	1.40E+02	1.84E+02	6.56E+02	2.88E+02	2.96E+02	4.48E+02	**4.82E+00**	3.16E+02	6.14E+02	9.24E+01	5.12E+00
	Std	1.04E+02	1.47E+02	1.46E+02	2.27E+02	1.71E+02	2.22E+02	1.26E+02	2.12E+02	1.61E+02	2.92E+00	1.07E+02
CEC08	Mean	2.72E+00	2.84E+00	6.03E+00	5.16E+00	5.47E+00	5.79E+00	4.65E+00	5.73E+00	5.15E+00	2.08E+00	**1.87E+00**
	Std	8.77E-01	1.15E+00	5.43E-01	6.35E-01	8.04E-01	7.88E-01	1.12E+00	5.84E-01	7.42E-01	3.21E-01	5.54E-01
CEC09	Mean	2.35E+00	2.36E+00	9.99E+01	2.43E+00	2.47E+00	4.73E+00	3.49E+00	2.55E+00	2.88E+00	**1.41E+00**	1.56E+00
	Std	6.23E-03	1.29E-02	9.30E+01	4.46E-02	7.25E-02	7.77E-01	2.30E-01	6.01E-02	9.67E-02	2.03E-01	9.74E-0.3
CEC10	Mean	1.98E+01	2.03E+01	2.05E+01	2.00E+01	2.01E+01	2.02E+01	2.01E+01	2.02E+01	2.04E+01	2.10E+01	**2.00E+00**
	Std	1.50E+00	9.77E-02	8.13E-02	8.35E-02	9.07E-02	4.86E-02	2.36E-02	1.46E-01	9.96E-02	4.85E-04	1.50E-05

Table 4 Comparative analysis of Friedman Test results for 10 modern CEC2019 benchmark functions

Function	EHOI	EHO	SCA	SSA	GOA	WOA	BBO	MFO	PSO	FA	GOQRFA
CEC01	2	4	7	5	9	8	10	6	11	3	1
CEC02	4	4	8	4	7	4	10	4	11	9	1
CEC03	7	7	7	7	7	7	7	7	7	2	1
CEC04	3	4	11	5	9	10	7	8	6	2	1
CEC05	2.5	2.5	11	6	8	10	7	5	9	4	1
CEC06	7	9	11	3	6	8	5	4	10	2	1
CEC07	4	5	11	6	7	9	1	8	10	3	2
CEC08	3	4	11	7	8	10	5	9	6	2	1
CEC09	3	4	11	5	6	10	9	7	8	1	2
CEC10	2	8	10	3	4.5	6.5	4.5	6.5	9	11	1
Mean	3.75	5.15	9.8	5.1	7.15	8.25	6.55	6.45	8.7	3.9	1.2
Rank	2	5	11	4	8	9	7	6	10	3	1
Statistic	56.96818182										
p-value	7.91E-13										

Table 5 Holm’s step-down procedure result

Comparison	*p* value	Rank	0.05/(k-i)	0.1/(k-i)
GOQRFA vs SCA	3.35E-09	0	0.00500	0.01000
GOQRFA vs PSO	2.14E-07	1	0.00556	0.01111
GOQRFA vs WOA	1.00E-06	2	0.00625	0.01250
GOQRFA vs GOA	3.02E-05	3	0.00714	0.01428
GOQRFA vs BBO	1.55E-04	4	0.00833	0.01667
GOQRFA vs MFO	2.00E-04	5	0.01000	0.02000
GOQRFA vs EHO	3.87E-03	6	0.01250	0.02500
GOQRFA vs SSA	4.28E-03	7	0.01667	0.03333
GOQRFA vs FA	3.44E-02	8	0.02500	0.05000
GOQRFA vs EHOI	4.28E-02	9	0.05000	0.10000

and simulations of the workflow scheduling in the cloud-edge environment. This simulator is widely used by the academic society for numerous other fields of research as well, such as fault tolerance, energy consumption, and resource scheduling.

As in [46], we utilized Pegasus’s workflow generators in the conducted experiments. The Pegasus group has implemented a collection of workflow generators, based on the available data gathered from the real-life execution environments of numerous real-world scientific workflows. Workflow models that have been released by the Pegasus group and freely available in the workflow generator include CyberShake, Epigenomics, Inspiral, Montage, and Sipht.

For each of the implemented workflows, the properties which specify the number of nodes and edges, the average size of data, and mean task reference time based on Xeon@2.33 GHz CPU (cu = 8) are presented in Table 6 [47]. Each of the workflows has a characteristic DAG structure. For instance, CyberShake has three options for the number of task nodes, 30, 50, and 100. This can be observed as DAG structures with low, medium, and high number of task nodes in the conducted experiments. As those workflows are frequently utilized in practical applications, we have used them in conducted experiments in order to be able to measure the performance, evaluate and compare scheduling algorithms.

The structures of all employed workflows (CyberShake, Epigenomics, Inspiral, Montage, and Sipht) are available on https://confluence.pegasus.isi.edu/display/pegasus/WorkflowHub.

The parameters for the cloud/edge servers, with different processing power and communication capabilities, that were used in simulations, are given in Table 7. There are six cloud and four edge servers in the system, and their characteristics include the following: processing rate and cost, available bandwidth, and the cost of communication. Again, we have used cloud and edge servers with the same characteristics as in [46].

Finally, since by conducting empirical simulations on standard unconstrained benchmarks (Subsection 6.1), we have established satisfying control parameters’ adjustments for the proposed GOQRFA, and since adaptations of the GOQRFA for practical workflow scheduling do not include modifying metaheuristics for discrete optimization, in all simulations we used GOQRFA parameters as shown in Table 2. Similarly, we used the same settings of the original FA as it is shown in Table 2.

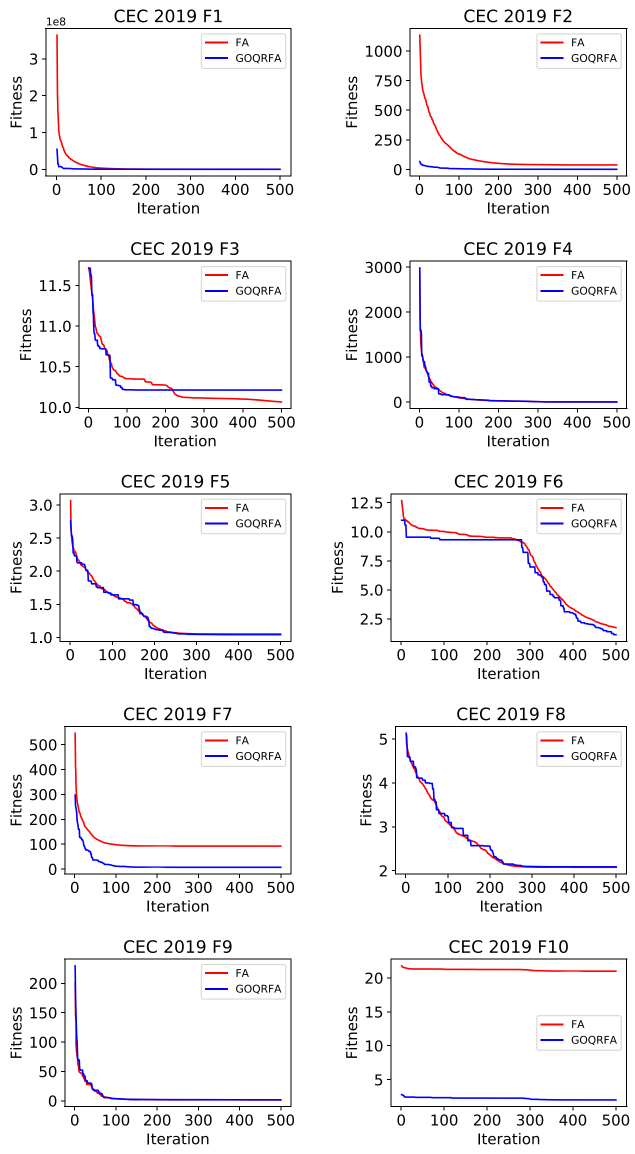

Fig. 3 Convergence graph of the CEC2019 benchmark function

Moreover, for the purpose of objective comparative analysis with approaches presented in [46], we have executed both metaheuristics (GOQRFA and original FA with dynamic step size) in 300 iterations (*MaxIter*=300) with 45 and 50 individuals in the population (*NS*=45, *NS*=50) for GOQRFA and basic FA, respectively. Approaches shown in [46] were tested with a population size of 50, however, based on the GOQRFA complexity in terms of function evaluations, we decreased the population size to make a fair comparative analysis. To neutralize the impact of randomness on algorithms' performance, each algorithm is executed in 30 independent runs, and mean results were recorded.

6.2.2 Encoding and adaptations of proposed algorithm

To adapt the proposed GOQRFA for workflow scheduling in a cloud-edge environment, we used a similar encoding strategy as in [46] and [34]. Each potential solution (GOARFA individual) represents a scheduling plan for the tasks submitted by end-users to the cloud-edge system. The length of each solution is equal to the number of tasks in the workflow (DAG), for example, if there are 10 tasks in the workflow, consequently the solution's dimension (*D*) is 10.

In the algorithm, each task in a solution (workflow) is mapped to the cloud or the edge server's ID. The assigned server is responsible for the task execution. The ID of the servers is from 0 to 9. The cloud servers have IDs from 0 to 5, while the edge servers have IDs from 6 to 9 and their configuration is detailed in Table 7. An example of the solution encoding is illustrated in Fig. 4a, where a workflow contains 10 tasks, and each task is assigned to a server ID.

However, some of the submitted tasks should be executed in almost real-time, and such tasks must be assigned to edge servers for fast processing because the transfer time between the end-user device and the edge server is significantly lower than from the end-user device and the cloud server. We implemented this similarly as in [46] by setting priorities for these tasks to differentiate them from the common tasks that can be executed on either cloud or the edge server. An example of tasks with priority is given in Fig. 4b, where tasks 3 and 7 have real-time requirements and were scheduled for execution on edge servers with ids 8 and 7, respectively. All tasks with priority can only be assigned, at initialization as well as at each update, to a server with ID between 6 and 9.

In the state-of-the-art approach presented in [46], the authors used combined objective function f (Eq. (9)) for fitness. In our GOQRFA proposed approach, the fitness of each solution is determined in the same way as it is expressed brightness in the original FA approach:

$$fit(x) = \begin{cases} \dfrac{1}{f(x)}, \text{ if } f(x) > 0 \\ 1 + |f(x)|\text{, otherwise} \end{cases} \tag{29}$$

where $fit(x)$ and $f(x)$ denote the fitness and objective function value for solution x, respectively. The scheduling algorithm implementation is detailed in Algorithm 3.

Algorithm 3 GOQRFA scheduling algorithm pseudo-code

Input: The opulation size NS (total number of scheduling plans (workflows or solutions)); the number of tasks (dimension D); the cloud server and the edge server set $S = \{0, \cdots, 9\}$; the maximum number of iterations ($MaxIter$); Iteration counter ($t = 0$); GOQRFA control parameters (γ, β_0, α_0, α_{min}, nrs, ebp, p, IUC_{p0}, mpd and mpi).
Output: The best scheduling plan (X_{best})

```
for $i = 1$ to $NS$ do
    Randomly initialize the scheduling plan $X_i$
    Update the scheduling plan $X_i$ by utilizing $QRBL$ strategy according to Algorithm 1
    Calculate the fitness value of $X_i$
end for
Sort the scheduling plans according to the fitness value
Save the current best scheduling plan with the lowest fitness value
while $t < MaxIter$ do
    Update the scheduling plan by utilizing the original FA position update
    if $t \leq ebp$ then
        Execute $DUC$ mechanism
    else
        Execute $IUC$ mechanism
    end if
    Update the population by utilizing $QRBL$ strategy according to Algorithm 1
    Update dynamic parameters $\alpha$ and $IUC_p$
end while
Return the best scheduling plan ($X_{best}$)
```

Also, since there are many tasks in the utilized workflow models (Table 6), we did not adapt GOQRFA meta-heuristics for discrete optimization problems, and the results are rounded to the closest integer value. Based on the conducted empirical tests, better results can be obtained by using this approach.

6.2.3 Weight coefficient simulations

The objective function (Eq. 9) is determined by balancing between makespan and total cost objectives using a weight coefficient κ. Following a similar strategy as in [46], we wanted to establish κ value that will obtain the lowest objective.

We have performed a set of simulations for *Montage 100* workflow with the values of κ between 0.1 and 0.9 using the step size of 0.1 for the original FA and proposed GOQRFA. We wanted to find a value of κ that will generate the lowest objective function value. Based on empirical simulations, our GOQRFA and the original FA generate good results for this *Montage* workflow with 100 task nodes.

Each simulation is conducted in 30 independent runs and the mean results of makespan, total cost, and combined objective (makespan + total cost) are reported. Simulation results of GOQRFA, the FA, and directional and non-local-convergent PSO (DNCPSO) for makespan, cost, and combined objective are summarized in Table 6.2.3, and corresponding charts are shown in Fig. 5. The DNCPSO approach results were taken for the purpose of comparative analysis, and they are retrieved from [46].

According to the analysis of the results from Table 8 and Fig. 5 using an "elbow" method, the optimal value of κ, that generates the lowest combined objective value, is 0.8.

The same conclusion was derived in [46], where DNCPSO was tested for the same *Montage 100* workflow. With the increase of κ, the influence of makespan in the combined objective is increasing, while the influence of cost is decreasing. Moreover, by analyzing the results from Table 8, it can be seen that the makespan is more susceptible to changes in κ than the cost. For example, the variance of the makespan of our proposed GOQRFA is 71.80, while the same value for the cost is only 1.20. It further means that on average, the makespan has a higher influence on the combined objective than the cost and it is better to set a large weight coefficient κ for the makespan in the combined objective function (Eq. 9).

Moreover, by observing the results from Table 8 and Fig. 5, it can be seen that for all values of weight coefficient κ, our proposed GOQRFA obtains better results for makespan, cost, and combined objective than both—the original FA and DNCPSO. It is also interesting to note that the DNCPSO proved to be significantly better meta-heuristics than the original FA by reaching better results for all three metrics.

Since we concluded that the value of $\kappa = 0.8$ obtains the best objective function value, by significantly minimizing makespan and only slighting increasing the cost, we used this setting for the purpose of comparative analysis with other state-of-the-art approaches, that was given in Sect. 6.2.4.

Table 6 Properties of the DAGs in practice

DAG	Nodes	Edges	Average data size (MB)	Average task runtime (per time unit)
CyberShake_30	30	112	747.48	23.77
CyberShake_50	50	188	864.74	29.32
CyberShake_100	100	380	849.60	31.53
Epigenomics_24	24	75	116.20	681.54
Epigenomics_46	46	148	104.81	844.93
Epigenomics_100	100	322	395.10	3954.90
Inspiral_30	30	95	9.00	206.78
Inspiral_50	50	160	9.16	226.19
Inspiral_100	100	319	8.93	206.12
Montage_25	25	95	3.43	8.44
Montage_50	50	206	3.36	9.78
Montage_100	100	433	3.23	10.58
Sipht_30	30	91	7.73	178.92
Sipht_60	60	198	6.95	194.48
Sipht_100	100	335	6.27	175.55

Table 7 Parameter list of cloud and edge computing resource used in simulations

Server type	Server ID	Processing rate (MIPS)	Processing cost (time unit)	Bandwidth (Mbps)	Communication cost (time unit)
Cloud servers	0	5,000	0.5	800	0.5
	1	5,000	0.5	500	0.4
	2	3,500	0.4	800	0.5
	3	3,500	0.4	500	0.4
	4	2,500	0.3	800	0.5
	5	2,500	0.3	500	0.4
Edge servers	6	1,500	0.2	1,500	0.7
	7	1,500	0.2	1,000	0.6
	8	1,000	0.1	1,500	0.7
	9	1,000	0.1	1,000	0.6

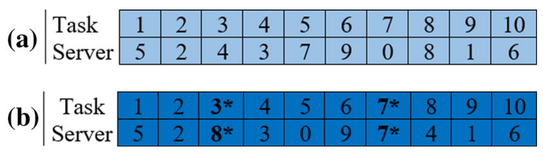
(a)
Task 1 2 3 4 5 6 7 8 9 10
Server 5 2 4 3 7 9 0 8 1 6
(b)
Task 1 2 3* 4 5 6 7* 8 9 10
Server 5 2 8* 3 0 9 7* 4 1 6

Fig. 4 Solutions encoding: **a** mapping between common tasks and servers **b** mapping between priority tasks and servers

6.2.4 Comparative analysis and discussion

In this subsection, we show a comparative analysis with other state-of-the-art metaheuristics and heuristics that were tested under the same conditions for the same workflow models (Table 6).

Following practice from [46], we have first compared the proposed GOQRFA with the original FA, DNCPSO [46] and other classical state-of-the-art metaheuristics and heuristics by using five workflow DAGs with fewer number of task nodes: *CyberShake*, *Epigenomics*, *Inspiral*, *Montage* and *Sipht* with 30,24,30,25 and 30 task nodes, respectively. Heuristics that were taken for the comparative analysis are heterogeneous earliest finish time (HEFT) [11] and MIN-MIN [43]. Moreover, besides the two above-mentioned metaheuristics approaches (the original FA and DNCPSO), we have also included PSO and GA in the analysis.

The objective function for all approaches included in the comparative analysis was adjusted with $\kappa = 0.8$ according to Eq. (9). As already mentioned, for the purpose of comparative analysis, we implemented and integrated into WorkflowSim the proposed GOQRFA and the original FA, while simulation results for other heuristics and metaheuristics were retrieved from [46].

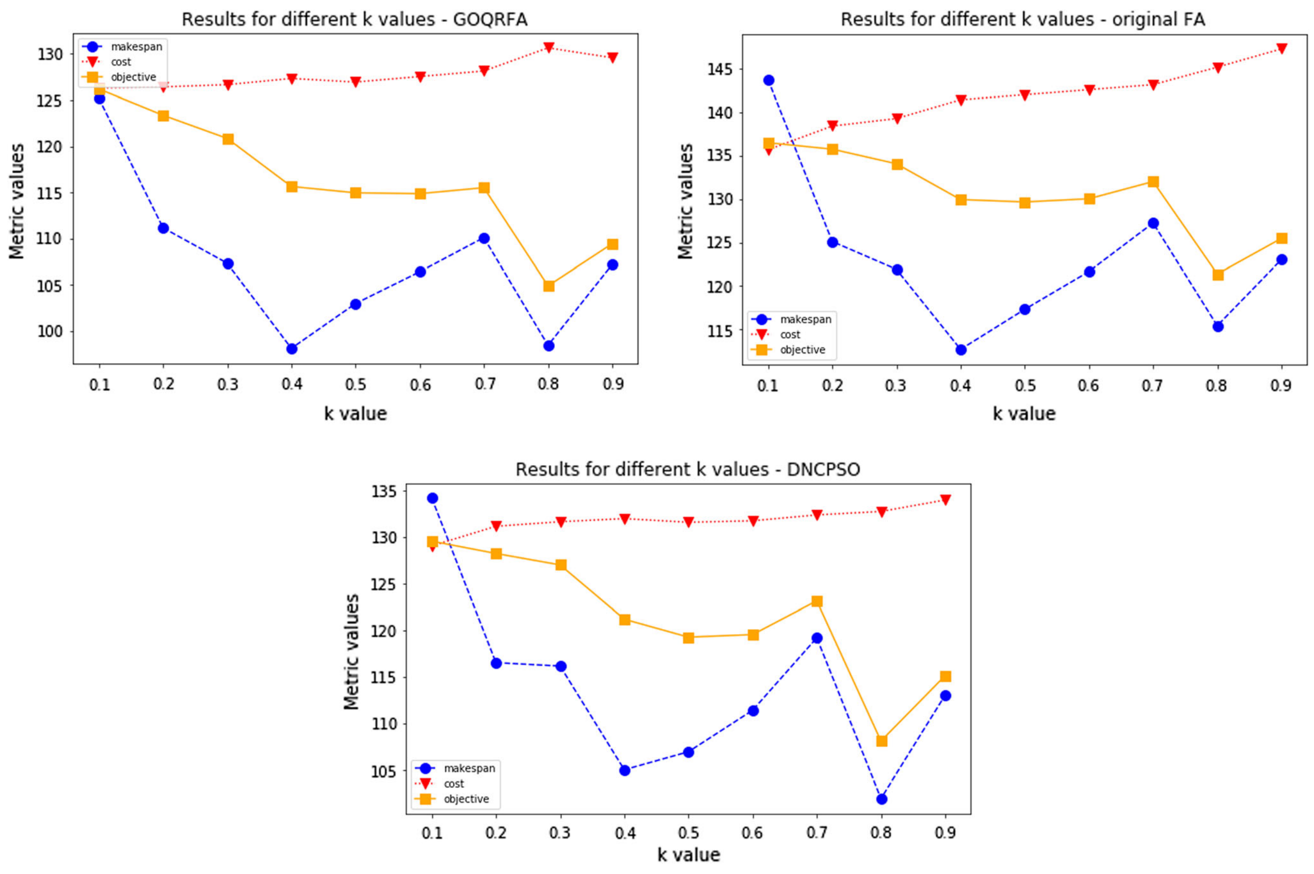

Fig. 5 Makespan, cost and combined objectives for different values of weight coefficient κ for three approaches

A visual representation of comparative analysis for the makespan, the cost, and the combined objective is shown in Fig. 6.

On average, according to the simulation results from Fig. 6, by taking into account all test instances, the proposed GOQRFA establishes the best performance in terms of all three metrics (the makespan, the cost, and the combined objective). By comparing metaheuristics approaches, it can be concluded that the original PSO obtains a worse performance. This is expected since the PSO does not employ an efficient mechanism for exploration and it is susceptible to being trapped in some of the sub-optimal regions of the search space. The GA establishes a better performance than the original PSO for all three indicators because by using a proper encoding strategy, GA can be easily adapted for discrete optimization problems such as cloud-edge workflow scheduling. The original FA metaheuristics obtains slightly better results than the original PSO. Based on the observed drawbacks of the basic FA (Subsection 5.1), like PSO, FA can easily be trapped in sub-optimal domains of the search region due to the lack of exploration power. However, the exploitation procedure of the original FA is better than in the case of PSO and the FA manages to establish a better performance in this type of challenge.

The DNCPSO as efficient state-of-the-art metaheuristics overcomes deficiencies of the original PSO and proves to be an effective optimizer for cloud-edge workflow scheduling [46]. However, our proposed GOQRFA proved to be even more efficient. For example, in the case of *Cybershake 30* instance, GOQRFA and DNCPSO both establish the same makespan value, but GOQRFA obtains a lower objective because it manages to slightly improve the cost compared to the DNCPSO. On the other hand, in the case of *Inspiral 30* and *Sipht 30* benchmarks, GOQRFA obtains a slightly higher cost than the DNCPSO, but a lower combined objective due to the shorter makespan. In the test instance, *Montage 25* GOQRFA establishes a better makespan, while both approaches generate the same cost value and finally, for the benchmark *Epigenomics 24*, GOQRFA obtains better values for the makespan and the cost and consequently for the combined objective.

The comparison between metaheuristics and HEFT and MIN-MIN heuristics is biased in favor of heuristics due to the experiment setup. Heuristics, as static scheduling algorithms, allocate resources to tasks before the start of the scheduling process, and they are oriented toward

Table 8 Makespan, cost and combined objective function values for different κ weight coefficients

κ	Metric	DNCPSO	FA	GOQRFA
0.1	makespan	134.15	143.75	125.21
	cost	129.02	135.70	126.32
	combined objective	129.53	136.50	126.20
0.2	makespan	116.52	125.11	111.17
	cost	131.13	138.43	126.43
	combined objective	128.20	135.77	123.38
0.3	makespan	116.16	121.92	107.34
	cost	131.62	139.29	126.67
	combined objective	126.98	134.07	120.87
0.4	makespan	105.03	112.7	98.12
	cost	131.95	141.45	127.34
	combined objective	121.18	129.95	115.65
0.5	makespan	106.96	117.32	102.98
	cost	131.56	142.02	126.95
	combined objective	119.26	129.67	114.97
0.6	makespan	111.41	121.67	106.42
	cost	131.72	142.60	127.55
	combined objective	119.53	130.04	114.87
0.7	makespan	119.19	127.25	110.11
	cost	132.35	143.17	128.15
	combined objective	123.14	132.02	115.52
0.8	makespan	101.96	115.43	98.45
	cost	132.72	145.19	130.66
	combined objective	108.11	121.38	104.89
0.9	makespan	113.03	123.05	107.21
	cost	133.95	147.3	129.59
	combined objective	115.12	125.47	109.45

minimizing the makespan. As a consequence, the makespan for HEFT and MIN-MIN is generally short; however, the cost is relatively high. An exception from this is the simulation with *Sipht 30* test instance. In this case, MIN-MIN heuristics maps certain tasks with heavy computation requirements to servers with a low processing rate and the makespan indicator is higher. With the established value of $\kappa = 0.8$, the makespan has much more influence on the combined objective than the cost and the value of this indicator for both heuristics is relatively low.

Despite the good makespan and the combined objective values of HEFT and MIN-MIN heuristics, the cost is high which is not acceptable in real-work workflow applications. Metaheuristics, as dynamic scheduling algorithms, obtain more stable results than heuristics and provide a better balance between the makespan and the cost and that is why they are more suitable for real-world environments.

In the second experiment suite, we have compared GOQRFA with DNCPSO, three other state-of-the-art improved PSO metaheuristics and the original FA for five workflows with more task nodes: *CyberShake 100*, *Epigenomics 100*, *Inspiral 100*, *Montage 100* and *Sipht 100*. In this simulation, we wanted to see how our approach performs on larger datasets.

Besides the original FA and DNCPSO in the comparative analysis, we have included the following enhanced metaheuristics: a multi-objective PSO (MPSO) [30], a hybrid GA-PSO (HGAPSO) [25], and a cooperative multi-swarm PSO (DMXPSOCLS) [24]. The results for DNCPSO, MPSO, HGAPSO, and DMXPSOCLS were retrieved from [46], while the results for GOQRFA and FA were generated by running simulations.

Based on the empirical simulation results, which are visualized in Fig. 7, like in the first comparative analysis, on average our proposed GAQRFA performs better than the other state-of-the-art metaheuristics included in the analysis. Again, the second-best approach is DNCPSO. For *Epigenomics 100* and *Montage 100* workflows, GOQRFS obtained better results for all three indicators: the makespan, the cost, and the combined objective than DNCPSO. In the *Inspiral 100* test instance, the GOQRFS established a better makespan with a slightly higher cost than the DNCPSO and better combined objective value. Only in *CyberShake 100* and *Sipht 100* DNCPSO slightly outperformed our GOQRFA.

As expected, the original FA obtained substantially worse performance metrics than GOQRFA for all the test instances. On average, FA performed similarly as HGAPSO.

The goal of the third conducted simulation is to validate the convergence speed of the proposed GOQRFA against the original FA. For this purpose, we employed five DAGs with medium number of task nodes: *CyberShake*, *Epigenomics*, *Inspiral*, *Montage* and *Sipht* with 50,46,50,50 and 60 task nodes, respectively. Based on the obtained results in 10 independent runs, we have generated convergence speed graphs for the combined objective function, which are shown in Fig. 8.

From the presented graphs, it can be stated that the GOQRFA substantially enhances the convergence rate of the original FA. After the initialization phase (iteration 0), the QRBL mechanism of GOQRFA establishes better solutions' diversity with a better objective function value in all the test instances, as can be seen from the graphs.

As iterations progress, GOQRFA converges much faster than the original FA and obtains the best solution within fewer iterations. In all simulations, GOQRFA manages to reach the best solution within 50-150 iterations. In contrast, the original FA reached the best solution in most simulations in the last 30% iterations. For example, in

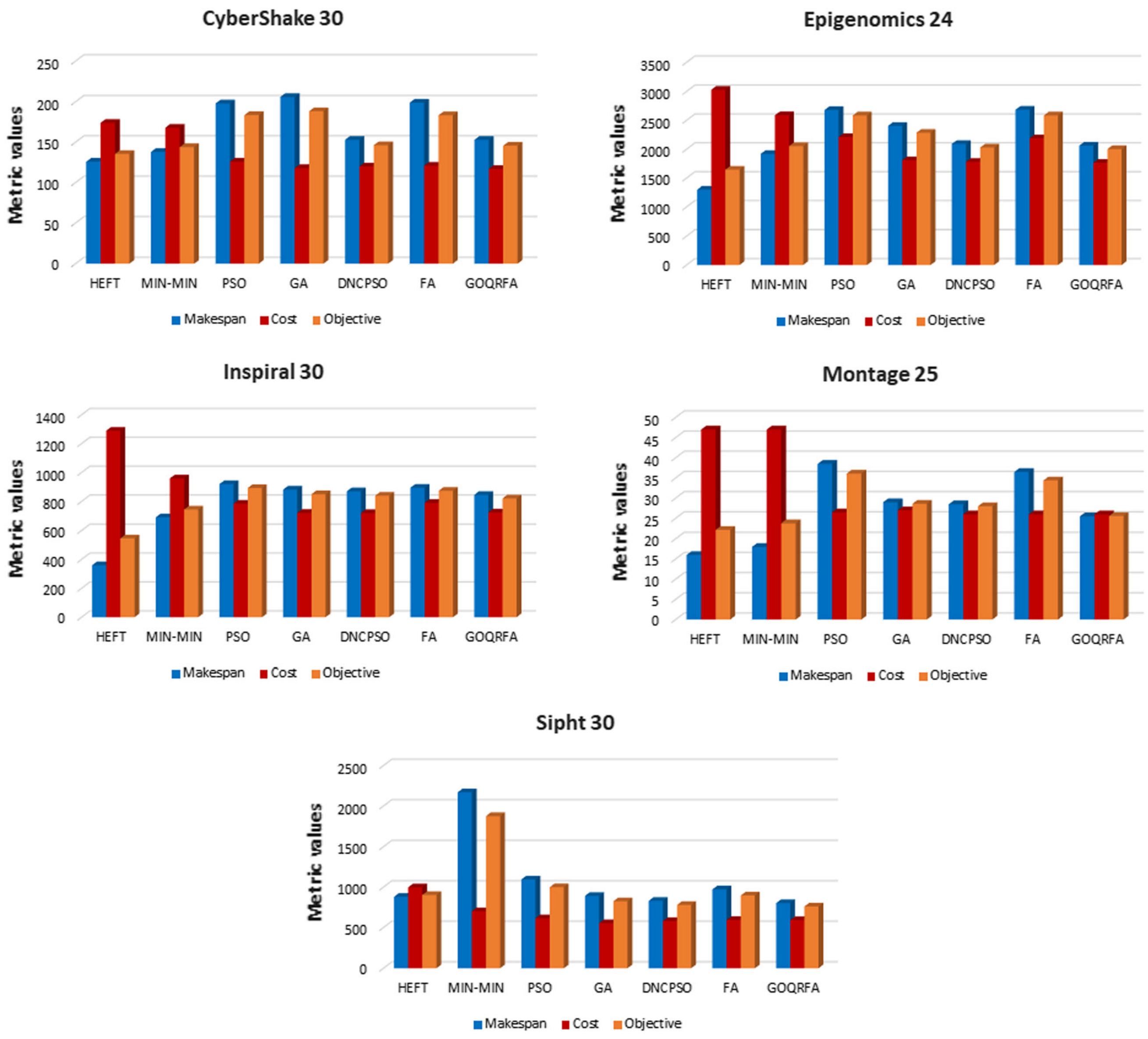

Fig. 6 Simulation 1 results—a comparison between GOQRFA, the original FA, DNCPSO and other metaheuristics and heuristics for datasets with fewer task nodes

CyberShake 50 simulation, GOQRFA converges to the best solution after the first 100 iterations, while the basic FA obtained the best solution after all available 300 iterations. Both metaheuristics showed good performance in *Sipht 60* test instance, where the GOQRFA reached the best solutions after 50 and the original FA after 100 iterations.

Also, the presented graphs show the influence of the novel GOQRFA's exploration mechanism *DUC* (refer to Sect. 5), which is triggered in the first 100 iterations, according to the *ebp* parameter adjustment (refer to Table 2). The experimental results illustrate that the *DUC* enables GOQRFA to quickly converge to optimal regions of the search domain in the early iterations. Afterward, the *IUC* mechanism that conducts a fine search in the optimum region is triggered.

Finally, motivated by the conducted simulations in [46], we have experimented with randomly generated workflow by DAG generator to validate the efficiency of the proposed GOQRFA for minimizing the communication cost and time against the original FA approach. The comparative analysis between GOQRFA and the original FA for communication time and cost indicators for the random workflow is shown in Fig. 9.

From the presented figure, it can be clearly seen that for both indicators (communication time and cost) the proposed GOQRFA substantially outperforms the original FA by establishing lower values for these indicators. The

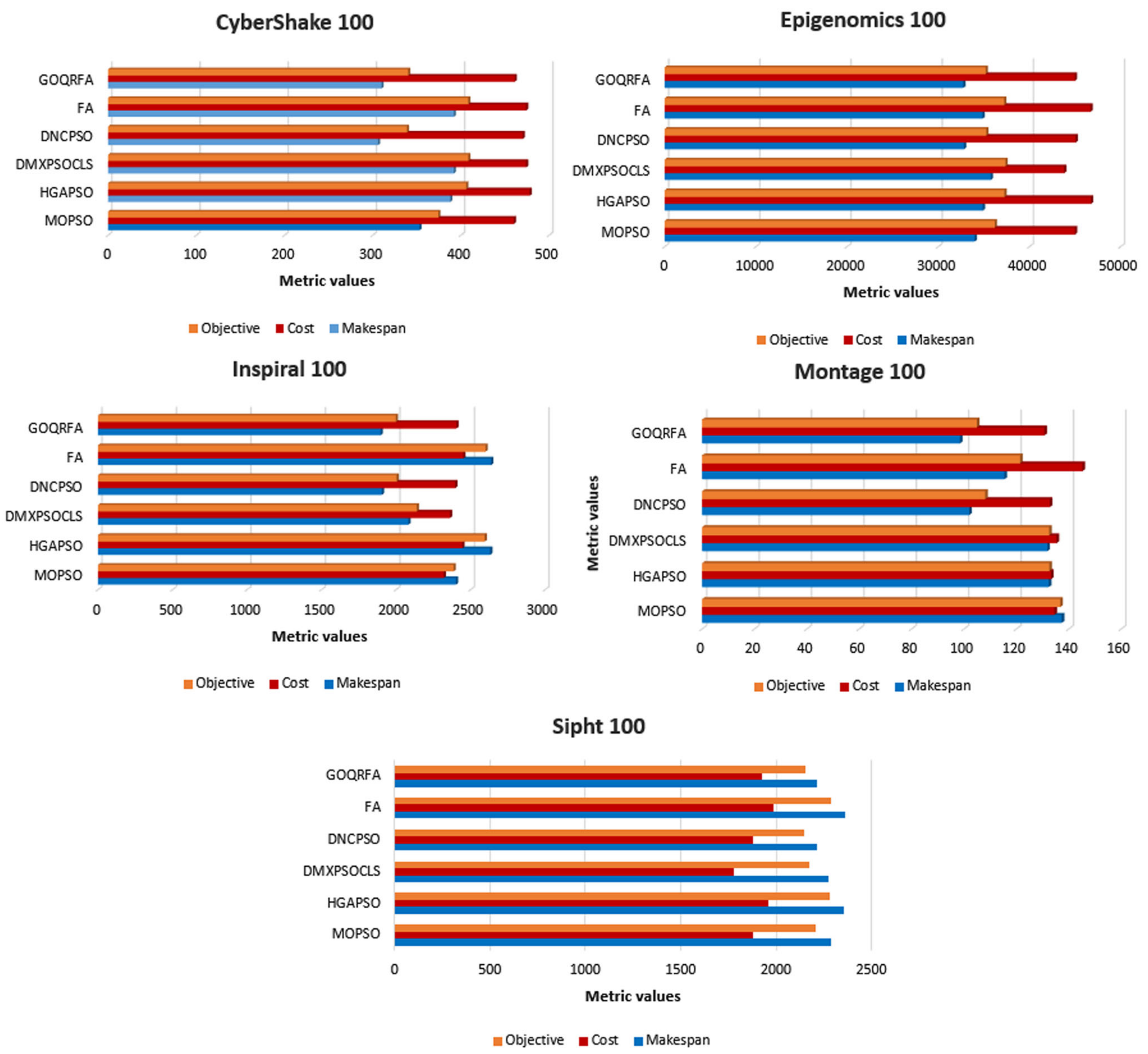

Fig. 7 Simulation 2 results—a comparison between GOQRFA, the original FA, DNCPSO and other improved PSO implementations for workflow models with 100 task nodes

GOQRFA generates a more efficient scheduling plan than the original FA by establishing better utilization of edge computing resources. The scheduling strategy generated by the original FA transfers more tasks for execution on the cloud and as a consequence, communication time and cost are higher.

7 Conclusion

In the research shown in this manuscript, we have proposed and implemented a novel efficient algorithm to tackle the problem of workflow scheduling in an edge-cloud environment. The main challenge in such environments is to find an efficient scheduling algorithm that can simultaneously minimize contradictory objectives, which are in the case of this experiment completion time (makespan) and cost. Since the solution space grows exponentially with the increase in computing tasks, it is practically impossible to find an optimal scheduling plan. Although many metaheuristics have already been implemented to solve this challenge, results still can be improved by utilizing enhanced methods.

To solve the problem of workflow scheduling, we have implemented an enhanced version of well-known FA swarm intelligence metaheuristics. We have chosen FA

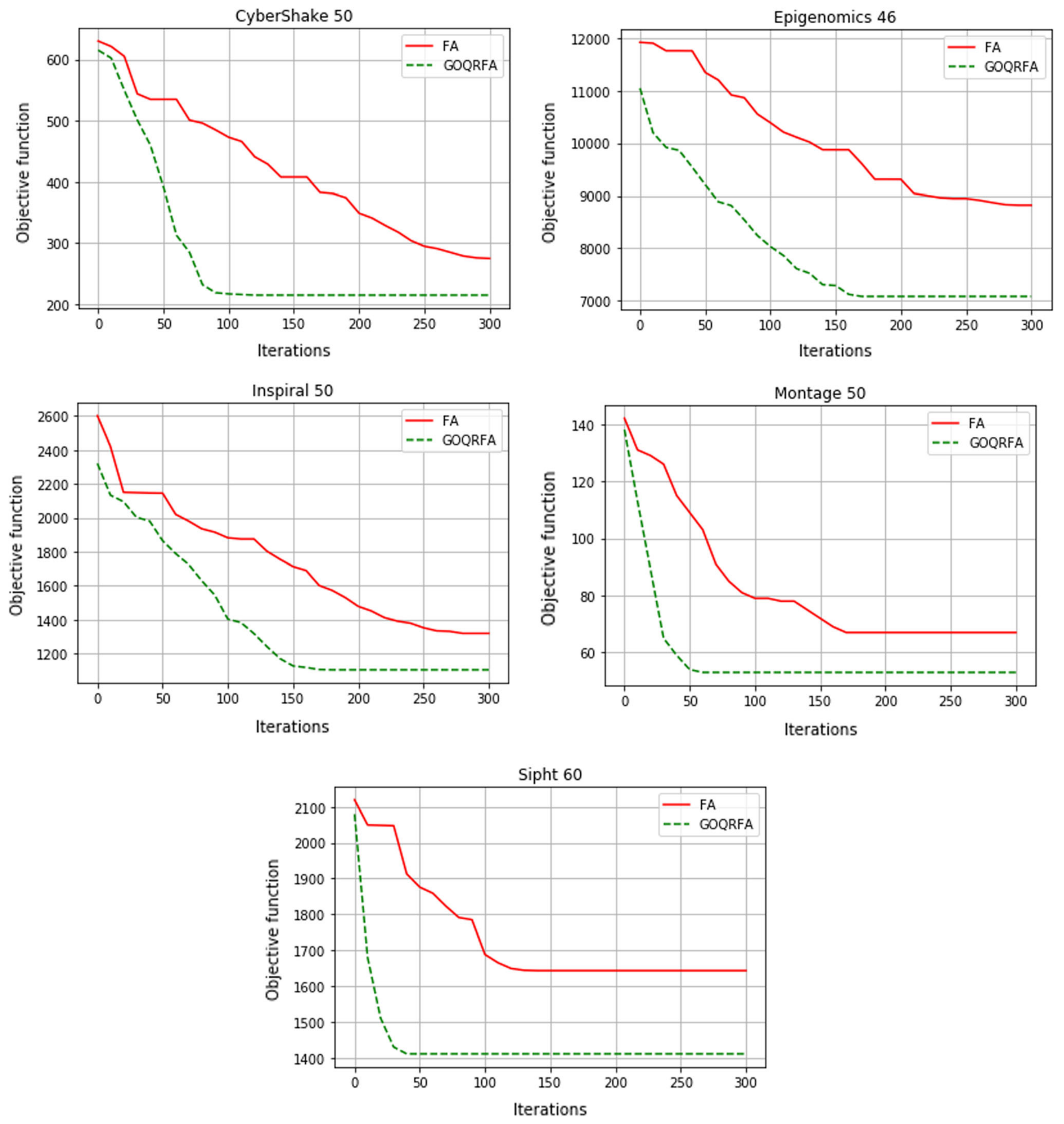

Fig. 8 Simulation 3 results—a convergence speed comparison between GOQRFA and the original FA for workflow models with medium nodes

over other state-of-the-art algorithms because it has proven to be an efficient optimizer in numerous practical domains. Furthermore, we have identified the deficiencies of the original FA, which can be summarized as the lack of explicit exploration and not a well-adjusted trade-off between intensification and diversification. Based on the observed deficiencies, we have devised and implemented an improved FA approach GOQRFA that overcomes observed drawbacks by incorporating genetic operators and quasi-reflective-based learning in the original implementation.

Following the usual practice from the modern computer science literature, the proposed GOQRFA was first tested on ten modern CEC 2019 benchmarks and compared with other outstanding methods. Afterward, practical experiments (simulations) along with comparative analysis with

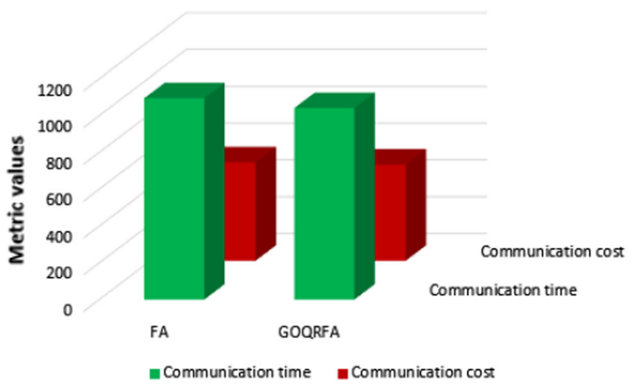

Fig. 9 Simulation 4 results—the communication time and cost comparison between GOQRFA and the original FA for random workflow model

other heuristics and metaheuristics for practical cloud-edge workflow scheduling challenge were performed. In both cases, GOQRFA outperformed the original FA and other state-of-the-art approaches used in the analysis.

The main contribution of this research is twofold. First, we have developed enhanced FA metaheuristics that establishes substantial improvements in terms of the solution quality and convergence than the basic FA and also obtains a better performance than other-state-of-the-art approaches. The proposed GOQRFA, such as any other hybridized or improved metaheuristic method, has the disadvantage of having more control parameters, which should be fine-tuned by the researcher. However, substantial performance improvements of GOQRFA over the original FA justify more control parameters. Second, we have managed to improve tackling workflow scheduling in cloud-edge environments by obtaining better results for the makespan, the cost, the combined objective, communication, and transfer costs than the results that are already published in the literature. The GOQRFA scheduler is capable of seeking the correlated balance between makespan and cost criteria, and at the same time, satisfying the real-time requirements.

There are many possible future research directions in this domain of computer sciences. We plan to continue our research with swarm intelligence by improving other outstanding algorithms either by hybridization with other heuristics/metaheuristics or by introducing minor changes (additional control parameters, search procedure modifications, etc.). Moreover, as part of future research, we will also try to simulate other scheduling scenarios in cloud-edge/cloud environments and establish improvements in generated scheduling plans.

Funding This research was supported by Ministry of Science of Republic of Serbia, Grant No. III-44006.

Declarations

Conflict of interest We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work.

Ethics approval The authors declare that they their work is compliant with ethical standards.

Consent to participate All authors have given their consent for this research.

Consent for publication All authors have given their consent for publication of this work.

References

1. Aggarwal A, Dimri P, Agarwal A, Bhatt A (2020) Self adaptive fruit fly algorithm for multiple workflow scheduling in cloud computing environment. *Kybernetes*
2. Bacanin N, Bezdan T, Tuba E, Strumberger I, Tuba M, Zivkovic M (2019a) Task scheduling in cloud computing environment by grey wolf optimizer. In *2019 27th Telecommunications Forum (TELFOR)* (pp. 1–4). IEEE
3. Bacanin N, Tuba E, Bezdan T, Strumberger I, Tuba M (2019) Artificial flora optimization algorithm for task scheduling in cloud computing environment. In: Yin H, Camacho D, Tino P, Tallón-Ballesteros AJ, Menezes R, Allmendinger R (eds) Intelligent Data Engineering and Automated Learning—IDEAL 2019. Springer International Publishing, Cham, pp 437–445. https://doi.org/10.1007/978-3-030-33607-3_47
4. Bacanin N, Tuba E, Zivkovic M, Strumberger I, Tuba M (2019c) Whale optimization algorithm with exploratory move for wireless sensor networks localization. In *International Conference on Hybrid Intelligent Systems* (pp. 328–338). Springer
5. Basha J, Bacanin N, Vukobrat N, Zivkovic M, Venkatachalam K, Hubálovskỳ S, Trojovskỳ P (2021) Chaotic harris hawks optimization with quasi-reflection-based learning: an application to enhance cnn design. Sensors 21:6654
6. Bäck T, Schwefel H (1993) An overview of evolutionary algorithms for parameter optimization. Evol Comput 1:1–23
7. Bezdan T, Cvetnic D, Gajic L, Zivkovic M, Strumberger I, Bacanin N (2021) Feature selection by firefly algorithm with improved initialization strategy. In *7th Conference on the Engineering of Computer Based Systems* (pp. 1–8)
8. Bezdan T, Zivkovic M, Antonijevic M, Zivkovic T, Bacanin N (2020a) Enhanced flower pollination algorithm for task scheduling in cloud computing environment. In *Machine Learning for Predictive Analysis* (pp. 163–171). Springer
9. Bezdan T, Zivkovic M, Tuba E, Strumberger I, Bacanin N, Tuba M (2020b) Glioma brain tumor grade classification from mri using convolutional neural networks designed by modified fa. In *International Conference on Intelligent and Fuzzy Systems* (pp. 955–963). Springer
10. Bezdan T, Zivkovic M, Tuba E, Strumberger I, Bacanin N, Tuba M (2020c) Multi-objective task scheduling in cloud computing environment by hybridized bat algorithm. In *International Conference on Intelligent and Fuzzy Systems* (pp. 718–725). Springer
11. Bittencourt LF, Sakellariou R, Madeira ER (2010) Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing* (pp. 27–34). IEEE

12. Boveiri HR (2015) List-scheduling techniques in homogeneous multiprocessor environments: a survey. Int J Softw Eng Its Appl 9:123–132
13. Cazacu R (2017) Comparative study between the improved implementation of 3 classic mutation operators for genetic algorithms. Procedia Engineering, 181, 634–640. http://www.sciencedirect.com/science/article/pii/S1877705817310287. https://doi.org/10.1016/j.proeng.2017.02.444.10th International Conference Interdisciplinarity in Engineering, INTER-ENG (2016) 6–7 October 2016. Tirgu Mures, Romania
14. Chen W, Deelman E (2012) Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *2012 IEEE 8th international conference on E-science* (pp. 1–8). IEEE
15. Ewees AA, Abd Elaziz M, Houssein EH (2018) Improved grasshopper optimization algorithm using opposition-based learning. *Expert Systems with Applications*, *112*, 156–172. http://www.sciencedirect.com/science/article/pii/S0957417418303701. https://doi.org/10.1016/j.eswa.2018.06.023
16. Fan Q, Chen Z, Xia Z (2020) A novel quasi-reflected harris hawks optimization algorithm for global optimization problems. *Soft Computing*, (pp. 1–19)
17. Forestiero A, Mastroianni C, Meo M, Papuzzo G, Sheikhalishahi M (2014) Hierarchical approach for green workload management in distributed data centers. In *European Conference on Parallel Processing* (pp. 323–334). Springer
18. Forestiero A, Mastroianni C, Papuzzo G, Spezzano G (2010) A proximity-based self-organizing framework for service composition and discovery. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* (pp. 428–437). IEEE
19. Forestiero A, Mastroianni C, Spezzano G (2008) Reorganization and discovery of grid information with epidemic tuning. Future Gener Comput Syst 24:788–797
20. Gajic L, Cvetnic D, Zivkovic M, Bezdan T, Bacanin N, Milosevic S (2021) Multi-layer perceptron training using hybridized bat algorithm. In *Computational Vision and Bio-Inspired Computing* (pp. 689–705). Springer
21. Hollander M, Wolfe DA, Chicken E (2013) Nonparametric statistical methods, vol 751. Wiley, Hoboken
22. Hyytiä E, Aalto S (2016) On round-robin routing with fcfs and lcfs scheduling. Perform Eval 97:83–103. https://doi.org/10.1016/j.peva.2016.01.002
23. Liu J, Mao Y, Liu X, Li Y (2020) A dynamic adaptive firefly algorithm with globally orientation. *Mathematics and Computers in Simulation*, *174*, 76–101. http://www.sciencedirect.com/science/article/pii/S0378475420300598. https://doi.org/10.1016/j.matcom.2020.02.020
24. Ma K, Hu S, Yang J, Xu X, Guan X (2018) Appliances scheduling via cooperative multi-swarm pso under day-ahead prices and photovoltaic generation. Appl Soft Comput 62:504–513
25. Manasrah AM, Ba Ali H (2018) Workflow scheduling using hybrid ga-pso algorithm in cloud computing. *Wireless Communications and Mobile Computing*, *2018*
26. Milan ST, Rajabion L, Darwesh A, Hosseinzadeh M, Navimipour NJ (2019) Priority-based task scheduling method over cloudlet using a swarm intelligence algorithm. *Cluster Computing*, (pp. 1–9)
27. Milosevic S, Bezdan T, Zivkovic M, Bacanin N, Strumberger I, Tuba M (2021) Feed-forward neural network training by hybrid bat algorithm. In *Modelling and Development of Intelligent Systems: 7th International Conference, MDIS 2020, Sibiu, Romania, October 22–24, 2020, Revised Selected Papers 7* (pp. 52–66). Springer International Publishing
28. Mohammadzadeh A, Masdari M, Gharehchopogh FS, Jafarian A (2020) Improved chaotic binary grey wolf optimization algorithm for workflow scheduling in green cloud computing. *Evolutionary Intelligence*, (pp. 1–29)
29. Muthusamy H, Ravindran S, Yaacob S, Polat K (2021) An improved elephant herding optimization using sine–cosine mechanism and opposition based learning for global optimization problems. Expert Syst Appl 172:114607
30. Pang L-P, Ng S-C (2018) Improved efficiency of mopso with adaptive inertia weight and dynamic search space. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (pp. 1910–1913)
31. Price K, Awad N, Ali M, Suganthan P (2018) Problem definitions and evaluation criteria for the 100-digit challenge special session and competition on single objective numerical optimization. In *Technical Report*. Nanyang Technological University
32. Rahnamayan S, Tizhoosh HR, Salama MMA (2007) Quasi-oppositional differential evolution. In *2007 IEEE Congress on Evolutionary Computation* (pp. 2229–2236)
33. Singh MR, Mahapatra S (2016) A quantum behaved particle swarm optimization for flexible job shop scheduling. Comput Ind Eng 93:36–44. https://doi.org/10.1016/j.cie.2015.12.004
34. Strumberger I, Bacanin N, Tuba M, Tuba E (2019) Resource scheduling in cloud computing based on a hybridized whale optimization algorithm. Appl Sci 9:4893
35. Strumberger I, Tuba E, Bacanin N, Tuba M (2020) Hybrid elephant herding optimization approach for cloud computing load scheduling. In: Zamuda A, Das S, Suganthan PN, Panigrahi BK (eds) Swarm, Evolutionary, and Memetic Computing and Fuzzy and Neural Computing. Springer International Publishing, Cham, pp 201–212
36. Strumberger I, Tuba E, Bacanin N, Zivkovic M, Beko M, Tuba M (2019b) Designing convolutional neural network architecture by the firefly algorithm. In *Proceedings of the 2019 International Young Engineers Forum (YEF-ECE), Costa da Caparica, Portugal* (pp. 59–65)
37. Thennarasu SR, Selvam M, Srihari K (2021) A new whale optimizer for workflow scheduling in cloud computing environment. J Ambient Intell Humanized Comput 12:3807–3814
38. Tizhoosh HR (2005) Opposition-based learning: A new scheme for machine intelligence. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)* (pp. 695–701). vol. 1
39. Tuba M, Bacanin N (2014) Improved seeker optimization algorithm hybridized with firefly algorithm for constrained optimization problems. Neurocomputing 143:197–207. https://doi.org/10.1016/j.neucom.2014.06.006
40. Wang H, Wang Y (2018) Maximizing reliability and performance with reliability-driven task scheduling in heterogeneous distributed computing systems. Journal of Ambient Intelligence and Humanized Computing. https://doi.org/10.1007/s12652-018-0926-9
41. Wang H, Zhou X, Sun H, Yu X, Zhao J, Zhang H, Cui L (2017) Firefly algorithm with adaptive control parameters. Soft Comput 3:5091–5102
42. Wang T, Liu Z, Chen Y, Xu Y, Dai X (2014) Load balancing task scheduling based on genetic algorithm in cloud computing. In *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing* (pp. 146–152). https://doi.org/10.1109/DASC.2014.35
43. Xu R, Wang Y, Huang W, Yuan D, Xie Y, Yang Y (2017) Near-optimal dynamic priority scheduling strategy for instance-intensive business workflows in cloud computing. Concurr Comput Pract Exp 29:e4167
44. Yang X-S (2009) Firefly algorithms for multimodal optimization. In: Watanabe O, Zeugmann T (eds) Stochastic Algorithms:

Foundations and Applications. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 169–178

45. Yang X-S, Xingshi H (2013) Firefly algorithm: recent advances and applications. Int J Swarm Intell 1:36–50
46. Ying X, Yuanwei Z, Yeguo W, Yongliang C, Rongbin X, Abubakar Sadiq S, Dong Y, Yun Y (2019) A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment. Future Gener Comput Syst 97:361–378. https://doi.org/10.1016/j.future.2019.03.005
47. Zhu Z, Zhang G, Li M, Liu X (2016) Evolutionary multi-objective workflow scheduling in cloud. IEEE Trans Parallel Distrib Syst 27:1344–1357
48. Zivkovic M, Bacanin N, Tuba E, Strumberger I, Bezdan T, Tuba M (2020a) Wireless sensor networks life time optimization based on the improved firefly algorithm. In *2020 International Wireless Communications and Mobile Computing (IWCMC)* (pp. 1176–1181). IEEE
49. Zivkovic M, Bacanin N, Venkatachalam K, Nayyar A, Djordjevic A, Strumberger I, Al-Turjman F (2021) Covid-19 cases prediction by using hybrid machine learning and beetle antennae search approach. Sustain Cities Soc 66:102669
50. Zivkovic M, Bacanin N, Zivkovic T, Strumberger I, Tuba E, Tuba M (2020b) Enhanced grey wolf algorithm for energy efficient wireless sensor networks. In *2020 Zooming Innovation in Consumer Technologies Conference (ZINC)* (pp. 87–92). IEEE
51. Zivkovic M, Bezdan T, Strumberger I, Bacanin N, Venkatachalam K (2021b) Improved harris hawks optimization algorithm for workflow scheduling challenge in cloud—edge environment. In *Computer Networks, Big Data and IoT* (pp. 87–102). Springer
52. Zivkovic M, Venkatachalam K, Bacanin N, Djordjevic A, Antonijevic M, Strumberger I, Rashid TA (2021c) Hybrid genetic algorithm and machine learning method for covid-19 cases prediction. In *Proceedings of International Conference on Sustainable Expert Systems: ICSES 2020* (p. 169). Springer Nature volume 176