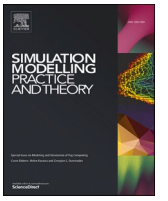
Simulation Modelling Practice and Theory 130 (2024) 102864

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat



HEPGA: A new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment

Hind Mikram [a], Said El Kafhali [a,*], Youssef Saadi [b]

[a] Hassan First University of Settat, Faculty of Sciences and Techniques, Computer, Networks, Modeling, and Mobility Laboratory (IR2M), B.P. 539, 26000 Settat, Morocco

[b] Data Science for Sustainable Earth Laboratory, Faculty of Sciences and Techniques, Sultan Moulay Slimane University, B.P. 523, 23000 Beni Mellal, Morocco

ARTICLE INFO

Keywords:
Metaheuristic algorithms
Workflow
Task scheduling
Resource utilization
Flowtime
Makespan
Cost
Multi-objective

ABSTRACT

Cloud data center comprises various physical and virtual machines, alongside storage datacenter services provided by cloud providers. Effectively mapping tasks to optimize resource utilization and load balance is essential for efficient task scheduling. This process, referred to as scheduling constraints, can significantly enhance overall efficiency. However, to harness the benefits of this scheduling, one must address the challenges arising during task execution. The interdependencies between tasks and the diverse resources available in the datacenter pose significant hurdles to efficient resource allocation. To address these challenges, this paper introduces the Hybrid HEFT-PSO-GA algorithm (HEPGA), aiming to efficiently allocate tasks to available resources across the datacenter. The HEPGA algorithm builds upon prior research by integrating the strengths of PSO (Particle Swarm Optimization) and GA (Genetic Algorithm) to optimize task scheduling in cloud computing environments. Through the fusion of PSO, GA, and HEFT-based Initialization, the algorithm strives to efficiently allocate tasks to processors, thereby minimizing the makespan. This approach capitalizes on parallel processing capabilities to further enhance resource utilization in the cloud environment. By varying the weights of the fitness function and considering the resources within the datacenter, we meticulously analyze the algorithm's performance concerning both makespan and Resource Utilization (RU). The results of these tests underscore the algorithm's consistent and robust resource utilization across diverse weight configurations, highlighting its adaptability to varying priorities. Moreover, the observed variations in makespan performance based on different weights emphasize the algorithm's potential for excellence when tailored to specific optimization goals.

1. Introduction

In the context of modern cloud computing, efficient task scheduling plays a pivotal role in optimizing the performance of cloud services [1]. The task scheduling process is critical across various distributed systems, aiming to enhance resource utilization, user experience, and energy efficiency. Researchers have approached the task mapping challenge using multi-objective frameworks, designing task allocation workflows through the utilization of metaheuristic techniques [2]. These methods, inspired by natural phenomena and societal behavior, harness advanced search strategies combined with randomization, empowering them with global

* Corresponding author.
E-mail address: said.elkafhali@uhp.ac.ma (S. El Kafhali).

https://doi.org/10.1016/j.simpat.2023.102864
Received 24 August 2023; Received in revised form 30 October 2023; Accepted 18 November 2023
Available online 19 November 2023

search capabilities and superior decision-making outcomes in diverse scenarios [3].

Managing large volumes of data has become a significant challenge due to the increased complexity and execution platforms and diversity of scientific applications. Scientists rely on WMS (Workflow Management Systems) to efficiently organize and manage remote data for large-scale studies [4]. Workflow applications, comprising sets of tasks and dependencies between them, find applications in various domains such as bioinformatics, astrophysics, and disaster modeling. The combination of different approaches and methods within a single solution has enabled the realization of complex scientific applications. Allocating tasks to VM processors within a multiprocessor setting requires consideration of the execution order of dependent tasks, a challenge associated with NP-complete complexity [5]. Multi-objective optimization becomes essential in scheduling workflow applications, enabling efficient load balancing across VMs in a cloud environment while simultaneously minimizing financial costs and execution duration. These three objectives necessitate a trade-off to achieve the optimal decision [6].

Considering the pay-per-use model of cloud environments, it is essential to address both the total financial cost and execution makespan while efficiently utilizing resources [7]. To optimize workflow scheduling, various efforts in heterogeneous environments focus on minimizing finish time and monetary cost. The reduction of scheduling algorithm execution time continues to be a critical challenge, particularly with the growing volume of data in cloud environments. To address this, various methods have been suggested to decrease task completion time by parallelizing sub-tasks and respecting their dependencies [8]. Precedence connections among tasks are typically depicted using directed acyclic graphs (DAGs), where vertices represent individual tasks and directed edges indicate task dependencies. Utilizing directed acyclic graphs allows for the representation of complex and diverse programs efficiently, making them a suitable and expressive approach for handling large-scale and diverse workloads in cloud computing [9].

Metaheuristic methods offer valuable solutions, each with its advantages and limitations. Hybridizing two or more methods can provide complementary benefits and superior performance. Among these methods, integrating two or more updating strategies proves promising in proposing a hybrid task scheduling algorithm for distributed computing, combining evolutionary strategy and swarm cognition [10].

The primary objective of this paper is to present an algorithm that effectively addresses the workflow-scheduling problem with a dual focus: first, to minimize the total makespan execution time, thereby reducing the overall time required for workflow completion; and second, to optimize resource utilization by efficiently allocating resources to tasks. By doing so, we aim to achieve a balanced distribution of tasks across available resources, which not only contributes to makespan reduction but also enhances the overall efficiency and cost-effectiveness of resource allocation. To achieve this, we have designed a task scheduling method that combines the GA with PSO. Furthermore, we have incorporated the widely adopted Heterogeneous Earliest Finish Time (HEFT) method, known for its effectiveness in handling heterogeneity, into our proposed approach.

While GA offers a powerful global search ability, it suffers from slow convergence. On the other hand, PSO converges quickly but can be trapped in local optimal positions. To overcome these limitations, the random initialization of both GA and PSO has been improved to enhance their performance. The proposed method (HEPGA) integrates the updating concept of PSO into the evolution process of GA, allowing HEPGA to leverage the benefits of both PSO and GA effectively, along with the advantages offered by the HEFT method in handling tasks with varying computational requirements on heterogeneous resources. HEFT is particularly advantageous in scenarios where the computational demands of tasks differ significantly, as it intelligently schedules tasks on heterogeneous resources to minimize the makespan. However, its effectiveness may be limited in certain cases with complex dependencies and dynamic task arrivals, leading to suboptimal solutions. By combining the strengths of GA, PSO, and HEFT, the proposed hybrid approach aims to enhance the performance of task scheduling in distributed computing systems, offering improved efficiency and effectiveness.

The initial phase, we employ PSO to establish a population and enhance particle velocities during each generation. PSO efficiently explores the search space by using a Levy distribution to generate diverse particles, replacing the random gene generation. This is followed by a GA stage where we refine solutions through selection, crossover, and mutation operations, working on the population of task-to-processor mappings derived from PSO. The GA improves the quality of these mappings to minimize the makespan. After the fusion step, we apply a parallelism procedure to the chromosomes. Our algorithm starts with HEFT-based Initialization, which maps tasks to processors based on their upward ranks and interdependencies, providing a solid foundation for both PSO and GA to converge towards optimal solutions.

In the upcoming sections, we will delve into the detailed steps of our proposed algorithm, providing accompanying pseudocodes to illustrate the process thoroughly.

The contributions of our work can be summarized as follows:

- We introduce HEPGA, a hybrid heuristic method that combines the strengths of GA, PSO, and HEFT to optimize resource utilization and minimize makespan in cloud computing environments.
- We develop a novel approach to task scheduling that leverages probability distribution to ensure the random initialization of metaheuristic methods is done in a way that maximizes the chances of finding the best solution.
- We test different weights for each objective in the fitness function to evaluate the performance of HEPGA through comprehensive simulations and comparisons with existing approaches.
- Our finding demonstrate that HEPGA can effectively handle large-scale data-intensive workflows and provide better solutions compared to traditional methods.

These contributions aim to advance the field of task scheduling in cloud computing environments, offering a comprehensive and robust solution to address the challenges of optimizing resource utilization, minimizing execution time, and ensuring cost-effectiveness while handling large-scale data-intensive workflows.

The forthcoming sections of this paper are organized as follows: In Section 2, we explore related research and methods applied to similar challenges. Section 3 provides a foundation by explaining the essential aspects of the problem. Moving forward, Section 4 presents our approach to problem-solving. The method's details are outlined in Section 5. Section 6 highlights the experimental results of the proposed approach and outlines the limitations of our work. Finally, Section 7 provides a summary of the findings and conclusions.

2. Related work

In recent years, the task-scheduling problem in cloud computing environments has garnered significant attention due to the escalating quantity of data and the need for reduced execution times. To address this challenge, various techniques have been proposed, aiming to minimize task completion times through the parallelization of sub-tasks while maintaining their precedence relationships. One common approach is to represent task dependencies using DAGs (directed acyclic graphs), where vertices represent tasks and directed edges depict the task dependencies. Such graphical representations allow for expressive and efficient scheduling of programs with diverse workloads.

In the context of the proposed methods, the PGA (Parallel Genetic Algorithm) was introduced as a hybrid heuristic approach [6]. PGA utilizes chromosomes to represent solution mappings of tasks onto computing units, employing a fitness function formulated with BNLP (Binary Non-Linear Programming). The strength of PGA lies in combining the advantages of PSO and GA through two crossover operations inspired by PSO. These crossovers involve the individual chromosome crossing with the global best and its historical best chromosomes, respectively. Simulated experiments of PGA demonstrated its superior performance in terms of user satisfaction, processing efficiency, and resource efficiency. However, PGA does have some weaknesses, particularly regarding its requirement for significant computational resources and time to converge to optimal solutions, especially for large-scale scheduling problems. On the other hand, the proposed method in [11] addresses the drawback of execution time in hybrid algorithms. Instead of randomly assigning the initial population for the genetic algorithm, this method leverages the best result obtained from the PSO algorithm. By utilizing the final result of PSO to reduce execution time, the researchers successfully achieve their main focus, which is to reduce power consumption and minimize load balancing costs.

As workflow applications grow more complex, researchers are increasingly exploring hybrid techniques to tackle the challenges of workflow scheduling. In this context, the proposed Hybrid Genetic Algorithm (HGA) takes center stage, seeking to enhance the performance of genetic algorithms through the incorporation of efficient heuristics and modifications in genetic operators [3]. By incorporating a heuristic-derived solution into the initial population, the Hybrid Genetic Algorithm (HGA) offers a substantial advantage and contributes to achieving an optimal makespan solution. Moreover, the HGA capitalizes on pre-existing knowledge, assuming that key information such as workflow size, task precedence, and available resources is known a priori, thus enabling it to leverage its advantages more effectively. The research in [5] introduces a novel approach to address the task-scheduling problem in cloud computing systems using a parallel GA within a MapReduce framework. By merging genetic algorithms with the HEFT algorithm, this method efficiently allocates sub-tasks to processors. This approach had a drawback in which the communication and computational costs of graphs were randomly selected from a predetermined range. Additionally, utilizing the MapReduce framework requires substantial low-level programming effort to implement mapping and reduction functions, which may increase the complexity of the scheduling process. Further research and optimization might be needed to address these challenges and enhance the scalability and robustness of the proposed method. The main goal of the paper [4] is to suggest an algorithm that deals with the workflow scheduling issue. The suggested approach distributes the load among the VMs and decreases the overall execution time while incurring the least amount of overall financial cost. To solve the workflow-scheduling problem, this research suggests a hybrid GA-PSO method that combines the advantages of both techniques and evaluates its performance using the workflow processes.

The scheduling problem involved multiple parameters, some of which may have been conflicting with each other. While enhancing the objectives' features, the generated solutions also needed to consider the implications of other factors. Additionally, security and privacy concerns need to be addressed alongside other considerations. Challenges associated with implementing the simulated method in real-world scenarios might have included administrative expenses, energy consumption by non-CPU-related components, data backups, and hardware issues. The proposed approach in [7] involved two stages of pre-processing and optimization using PSO. In the first phase, workflow tasks were categorized into appropriate queues based on their duration and complexity levels. To balance the workload on resources, thresholds were employed to allocate resources to tasks. PSO was then utilized to improve scheduling and identify better solutions in the subsequent stage. The proposed approach in [12] introduced a hybrid genetic algorithm that capitalized on both GA and the HEFT heuristic. Unlike previous methods, the algorithm optimized convergence by initializing the population with optimal solutions, leveraging HEFT's heuristic schedule. The authors also emphasized task independence by partitioning workflows and enabling parallel processing. Moreover, the algorithm's rigorous search mechanism, facilitated by multi-fold crossover and mutation operators, enhanced performance by covering a broad problem space. While the algorithm had limitations related to scalability and assumptions of task independence, it represented a significant advancement in addressing complex scheduling challenges in cloud environments. To attain a balance between task efficiency and energy efficiency, the authors in [13] introduced a two-step hybrid method known as the Genetic Algorithm and Energy-Conscious Scheduling Heuristic (GAECS). The primary objective of their approach was to strike a balance between task efficiency and energy efficiency in the context of multi-objective task scheduling within cloud computing. GAECS was designed to offer an effective and optimal solution that prioritizes the reduction of makespan and minimization of energy consumption. This research contribution addresses the key challenges of task scheduling in cloud environments, aligning with the broader objectives of enhancing both efficiency and sustainability in cloud computing.

Cloud data centers, which were massive and resource-intensive, resulted in significant energy consumption and longer task

execution times. Consequently, data transfer became frequent, and data centers needed to employ VM scaling to enhance resource efficiency. In [14], Utilizing CloudSim as the underlying simulator, a novel technique based on multi-objective optimization was employed to assess the performance of VM allocation. This study concentrated on evaluating the RAM and CPU requirements while addressing various challenges related to VM host placement. It tackled issues concerning VM selection and host allocation in the context of VMP (Virtual Machine Placement). The primary objective of this study was to evaluate performance using the Multi-Objective Particle Swarm Optimization (MO-PSO) approach to minimize execution time and energy consumption. However, implementing multi-objective optimization algorithms like MOPSO could increase computational complexity, especially in large-scale cloud environments. The search for optimal solutions that satisfy multiple objectives might have required significant computational resources and time, leading to suboptimal VM allocations. To mitigate communication costs, the authors in [9] proposed a task allocation approach that employs traditional PSO to form clusters of highly interconnected activities. Furthermore, they devised a strategy to minimize execution costs within a static environment of a Distributed Computing System (DCS) by assigning the identified task clusters to capable processors. The objective was to identify the optimal task-to-processor allocation that maximizes both the system's Response Time (RT) and Flowtime. This task allocation challenge in a heterogeneous system was closely linked to critical performance metrics such as System Cost (SC).

In the ever-expanding domain of IoT and fog-cloud computing, efficient resource allocation and scheduling are vital for timely data processing. The power of two choices load balancing approach, widely effective in traditional distributed systems, is now being explored in multi-tier environments. The authors in [15] introduced three resource allocation and scheduling heuristics for real-time workflow jobs in IoT-fog-cloud settings. One strategy employed exhaustive searches for optimal resource allocation, while the other two leveraged the power of two choices. However, these approaches brought complexities, sensitivity to network dynamics, and challenges in handling real-time constraints within dynamic IoT-fog-cloud environments.

In the context of workflow scheduling and task offloading, the research in [16] focused on Cloud and Fog environments, particularly for extreme data workflows with stringent response time requirements. It introduced a novel approach called Multi-objective Workflow Offloading (MOWO) that optimized response time, reliability, and cost. The study formulated workflow scheduling as a multi-objective optimization problem and employed the NSGA-II metaheuristic to develop the MOWO algorithm. However, the algorithm proved to be sensitive to initial configurations and did not fully account for real-world variability and resource heterogeneity, particularly in dynamic workflows. The proposed workflow scheduling algorithm in [17] combined real-world pricing and the IaaS model while employed an EMO (Evolutionary Multi-objective Optimization) approach and a specialized encoding scheme to address multi-objective cloud scheduling challenges. It also acknowledged that considering real-world pricing may impact computational efficiency, emphasizing the importance of accurate cloud computing system modeling.

To enhance the security of VMs, various security mechanisms were made available. However, due to the constraints of limited resources and budgetary considerations within the cloud environment, ensuring the security of all VMs became an impractical endeavor. To tackle this challenge, the author of [18] proposed a novel scheduling method, known as MOWS, which optimally assigned tasks to Virtual Machines (VMs) in a way that minimized security threats while still ensuring efficient workflow execution. By considering both task security demands and interactions, the method sought to strike a balance between security and performance, which was particularly vital in shared cloud environments, focusing on minimizing the potential threats posed by malicious tasks without excessively compromising the cloud's performance. However, a weakness of this approach was that it introduced additional computational overhead due to security and performance optimization, potentially impacting the overall workflow response time. Additionally, the feasibility of real-time threat response in a large-scale cloud environment could be complex, and the rescheduling strategies might face scalability issues when addressing multiple simultaneous attacks.

After analyzing the existing related work, it became apparent that a combining GA and PSO was a popular approach for solving Multi-Objective Task Scheduling Problems (MOTSPs) in cloud computing environments. However, both GA and PSO algorithms suffer from a common limitation – their reliance on randomly generated initial solutions. To address this shortcoming, we propose a novel approach that leverages the Levy distribution to generate high-quality initial solutions. The Levy distribution is a probability distribution characterized by a long tail, which means it is more likely to produce larger jumps in the search space. This property makes the Levy distribution particularly useful for exploring vast solution spaces, as it can efficiently explore the entire range of possibilities. Moreover, we employ the HEFT to generate an initial priority list of tasks for scheduling. By iteratively adjusting the weights in our fitness function, we aim to find the optimal weight combinations that allow us to simultaneously minimize the makespan and achieve the desired resource utilization in the data center. Additionally, dealing with the NP-Hard nature of MOTSPs in the Cloud, especially concerning cost and time optimization, had remained a significant challenge. In other words, finding the optimal weight values could be a daunting task and might have required some trial and error. Thus, performing sensitivity analysis and exploring various weight combinations became necessary to identify a set of weights that yielded favorable solutions. As a result, we undertook this research to address this problem and aimed to achieve superior results compared to existing algorithms. Thus, our proposed method aspires to overcome these challenges by integrating the strengths of HEFT, GA, PSO and levy distribution to provide superior solutions for multi-objective task scheduling in cloud computing environments.

3. Background

3.1. Methods' hybridization

There are several meta-heuristic techniques, each with specific benefits and limitations. Hybridization appears as a potential strategy to leverage the full potential of these approaches and achieve improved performance. Fig. 1 illustrates three typical

hybridization approaches. In the first technique (Fig. 1a), during each iterative or evolutionary phase, the population updating methods of two or more algorithms are progressively applied. The second approach involves executing population strategies from one algorithm sequentially and then utilizing another one (Fig. 1b). However, these two techniques only combine the advantages of several meta-heuristic algorithms, which can sometimes lead to constrained or even suboptimal performance [10].

Parallel hybridization (Fig. 1c), in the context of optimization algorithms, refers to the simultaneous application of multiple optimization techniques.

The selection of the hybridization type is a pivotal and intricate step in designing an effective optimization algorithm. The hybridization process involves synergizing multiple meta-heuristic techniques to harness their collective strengths and improve the overall performance of the algorithm. However, identifying the most suitable hybridization type can be challenging, especially when considering the diverse array of possibilities. Often, a single algorithm may integrate multiple hybridization strategies to achieve superior results. This complexity highlights the significance of making well-informed choices to strike a balance between different techniques. The hybridization type chosen must align seamlessly with the problem's nature and characteristics to capitalize on the unique advantages each technique brings. It is through careful evaluation, experimentation, and a deep understanding of the algorithm's goals that the optimal hybridization type can be determined, ultimately leading to enhanced optimization outcomes.

3.2. Scientific workflow

A workflow is a widespread model used to describe scientific applications, where a collection of tasks represents the nodes of the workflow, and the communication between these tasks forms the links connecting the nodes [19]. In this paper, we focus on Directed Acyclic Graphs (DAGs) as one of the types of workflow.

3.2.1. Directed acyclic graph (DAG)

A DAG is a directed graph that does not contain any cycles, meaning there are no closed loops in the graph. In the context of task scheduling, the nodes of the graph represent individual sub-tasks, and the edges represent the dependencies between these sub-tasks. Each edge in the graph indicates that one sub-task must be completed before another can start [9]. Fig. 2 provides an illustrative example of DGA.

In other words, the graph G (V, E) represents the task dependencies in the system, where V is the set of nodes representing the sub-tasks, and E is the set of directed edges representing the dependencies between the sub-tasks. Communication costs are considered between sub-tasks, with the restriction that each sub-task can only be executed on a single processor. Consequently, the graph is designed to be acyclic to prevent circular dependencies, ensuring that tasks can be scheduled efficiently without encountering deadlocks or infinite loops. If two interconnected subtasks are assigned to the same processor, no communication cost is incurred. The graph's initial and terminal nodes represent the workflow's start and end points [20].

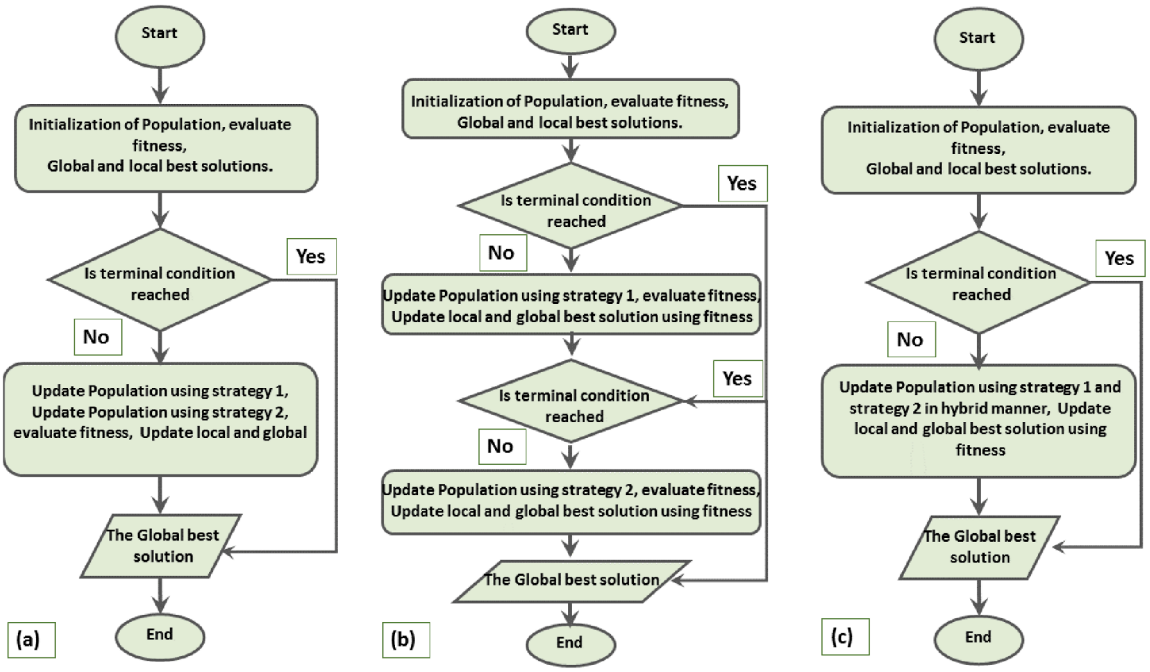


Fig. 1. The type of methods hybridization. (a) Sequential updating hybridization. (b) Sequential execution hybridization. (c) Parallel hybridization.

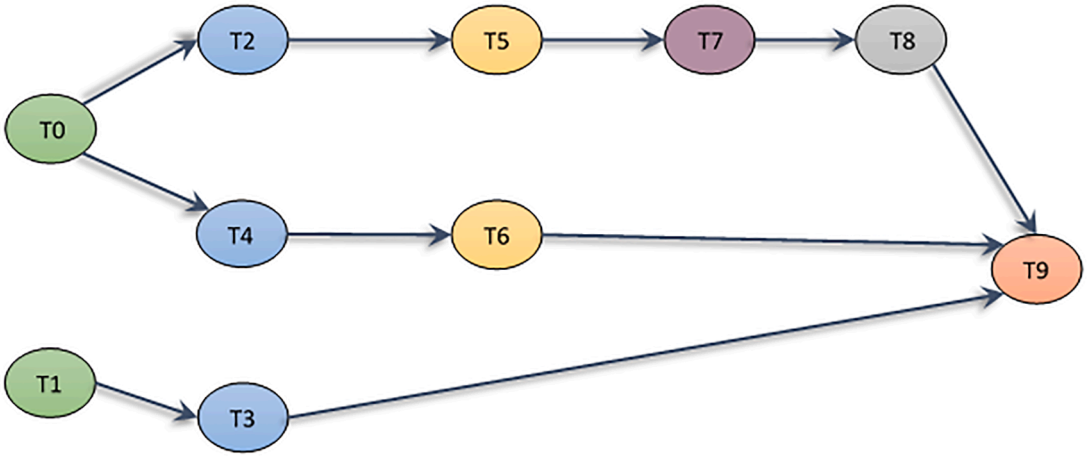


Fig. 2. Example of DAG.

3.3. Workflow scheduler

A scheduler holds a pivotal role in allocating tasks to available resources while maximizing a set of objectives. To cater effectively to users' requirements, the scheduler provides a visual representation of the workflow and establishes multiple queues for efficient task distribution. Workflows can be directed towards resources using either straightforward queue-based methods or more intricate strategies. The significance of an efficient scheduler lies in its ability to fulfill users' objectives, ensuring the optimal utilization of cloud resources. Process schedulers, such as GA, ACO, and PSO, discussed here, are designed to ensure satisfactory Quality of Service (QoS) for end-users [21].

3.3.1. GA (Genetic algorithm)

GA takes inspiration from genetic and evolutionary mechanisms observed in natural systems. As a population-based meta-heuristic, GA employs a range of operators to create optimized individuals from an initially randomized population. These operators play a pivotal role in guiding the convergence of individuals within the population. The core GA algorithm comprises three fundamental genetic operations: selection, crossover, and mutation. In the selection operation, specific solutions are chosen as parents, and then the crossover operator combines these parents to generate offspring. The mutation operator modifies the offspring based on predefined mutation rules. In the context of GA, solutions are referred to as individuals or chromosomes, and each iteration of the algorithm is termed a generation [12].

3.3.2. PSO (Particle swarm optimization)

PSO is an optimization method commonly used to address complex multimodal optimization problems. Drawing inspiration from the collective behavior of flocks of birds or schools of fish, PSO employs a population of particles, where each particle represents a potential solution with its unique position and velocity. Initially, all particles are randomly placed within the search space. As the algorithm progresses, these particles explore the search space to find the best possible solutions. During each iteration, particles update their positions based on their current velocity and are influenced by two crucial variables: pbest (personal best), representing the best position a particle has discovered so far, and gbest (global best), which signifies the best position among all particles in the swarm. These variables guide the particles' movements, enabling them to converge toward promising regions in the search space [22]. This technique finds applications in diverse fields, including artificial neural networks and various optimization problems.

3.3.3. HEFT (Heterogeneous earliest finish time)

The Heterogeneous Earliest Finish Time (HEFT) algorithm, first introduced in [23], stands out as a widely acknowledged list-based scheduling heuristic. With a proven track record of exceptional performance when compared to numerous other scheduling algorithms. HEFT is designed to handle a bounded number of fully connected heterogeneous processors. At its core, HEFT employs a descending priority order for the task list, determined by upward rank or b-level. Task-to-processor allocation is accomplished by considering the earliest finish time [4]. By relying on the concept of upward ranks to prioritize tasks in a non-rotating directed graph, the HEFT algorithm takes into account task dependencies and mean execution times across various processors. In the proposed hybrid approach, the HEFT schedule plays a pivotal role in forming the initial population. List-based scheduling comprises two distinct phases: the initial generation of the task-priority list, succeeded by processor allocation. Overall, the HEFT algorithm's ability to efficiently schedule tasks in complex environments makes it an ideal choice for many applications in cloud computing.

3.4. The optimization model

Finding the optimal solution to task workflow problems can be particularly challenging, especially when dealing with extensive search spaces inherent in cloud computing environments. The complexity of these issues can exponentially escalate with the number of tasks, processors, and other contributing factors. In such scenarios, a single-objective optimization approach may prove insufficient to comprehensively address the diverse and often conflicting objectives of a cloud-based system. This is precisely where multi-objective optimization steps in. It is a powerful technique that aims to address multiple objectives simultaneously, providing a comprehensive understanding of the problem landscape and allowing decision-makers to consider trade-offs between different goals.

A common multi-objective optimization model revolves around the minimization or maximization of multiple components within a function $\boldsymbol{fi}$, formulated as:

$$\textbf{Minimize or Maximize } \boldsymbol{fi}(\boldsymbol{x}1, \boldsymbol{x}2, \ldots, \boldsymbol{xn}) \tag{1}$$

In this equation, $\boldsymbol{fi}$ represents the $\boldsymbol{ith}$ component of the objective function, and $\boldsymbol{x}1, \boldsymbol{x}2, \ldots, \boldsymbol{xn}$ denote the decision variables influencing the function performance. The objective is to determine the optimal values for these decision variables that concurrently minimize or maximize each component $\boldsymbol{fi}$. This task involves reconciling multiple conflicting objectives [24].

The symbols used in the equations and algorithms in this research are listed in Table 1.

4. Problem formulation

The objective of a scheduling algorithm is to efficiently allocate resources $\boldsymbol{RC_i}$ to workflow applications $\boldsymbol{WA_i}$ in a cloud computing environment. The resources are represented by a pool of virtual machines $(\boldsymbol{V}1, \boldsymbol{V}2, \ldots, \boldsymbol{Vn})$, while the workflows consist of various tasks $(\boldsymbol{WA}1, \boldsymbol{WA}2, \ldots, \boldsymbol{WAm})$. The primary goal is to optimize the overall performance by minimizing execution time and cost while ensuring an even distribution of the workload among the available resources.

To achieve this, **equation (2)** is employed to computes the total execution cost of the task $\boldsymbol{WA_i}$ on all virtual machines $\boldsymbol{V_j}$**.** In other words, the total execution cost across all virtual machines is the sum of the processing costs of all tasks on their respective virtual machines.

$$\boldsymbol{TotalCost} = \sum_{i \in m} \sum_{j \in n} \boldsymbol{processingCost_{ij}} \;\forall\; \boldsymbol{i} \in \boldsymbol{m},\; \boldsymbol{j} \in \boldsymbol{n} \tag{2}$$

The term $\boldsymbol{processingCost_{ij}}$ represents the cost of executing a task $\boldsymbol{WA_i}$ in the $\boldsymbol{j^{th}}$ VM. It quantifies the expense incurred when task $\boldsymbol{i}$ is run on the virtual machine $\boldsymbol{j}$.

Furthermore, Eq. (3) defines the **makespan (M),** which is the maximum end time of all tasks across all processors. It can be represented as:

$$\boldsymbol{M} = \mathbf{Max}_{p \in P}\{\mathbf{Max}_{t \in WA}\, \boldsymbol{endExecutionTime}(\boldsymbol{t}, \boldsymbol{p})\} \tag{3}$$

Table 1
Symbols used in this paper.

Symbol	Description
$\boldsymbol{fi}$	Objective function
$\boldsymbol{WA_i}$	Workflow applications
$\boldsymbol{RC_i}$	Resources
$\boldsymbol{V_i}$	Virtual machines
$\boldsymbol{processingCost_{ij}}$	The cost of executing a task
$\boldsymbol{endExecutionTime}(\boldsymbol{t}, \boldsymbol{p})$	The end time of task $\boldsymbol{t}$ on processor $\boldsymbol{p}$.
$\boldsymbol{Ft_i}$	The flow time
$\boldsymbol{ArrivalTime}(\boldsymbol{WA_i})$	The time when the task arrives for execution.
$\mathbf{M}(\mathbf{x})$	The makespan
$\boldsymbol{TotalCost}(\boldsymbol{x})$	The total cost
DAG	The directed acyclic graph
$(\boldsymbol{\alpha},\ \boldsymbol{\beta},\ \boldsymbol{\lambda})$	The weights assigned to Total Cost, Makespan, and Flowtime, respectively
$\mathbf{w}$	The inertia weight
$\mathbf{c}_1$ and $\mathbf{c}_2$	The cognitive and social coefficients.
$\mathbf{P}^{\mathbf{t}}_{\text{best (ij)}}$	The personal best position
$\mathbf{P}^{\mathbf{t}}_{\text{bestglobal(ij)}}$	The global best position among all particles.
$\mathbf{P}^{\mathbf{t}}_{\mathbf{ij}}$	The current processor allocation of task j for particle i.
processor_schedule	Set of the scheduling details for each processor.
NbrOfProcessor	The Maximum number of processor
NbrOfCloudlet	The Maximum number of cloudlet
$\boldsymbol{mW}(\overline{\boldsymbol{t_i}})$	The average computational cost (mean weight) of task $\boldsymbol{WA_i}$.
$\mathbf{SUCC}(\boldsymbol{WA_i})$	The successor tasks $\boldsymbol{WA_i}$
$\boldsymbol{Cost}(\ \boldsymbol{WA_i},\ \boldsymbol{WA_j})$	The communication cost between tasks $\boldsymbol{WA_i}$and $\boldsymbol{WA_j}$
$\mathbf{UpWardRank}_{\boldsymbol{b}}(\ \boldsymbol{WA_j})$	The upward rank of each successor task $\boldsymbol{WA_i}$ of $\boldsymbol{WA_j}$.

Where $\boldsymbol{P}$ is the set of all processors in the schedule, $\boldsymbol{WA}$ is the set of all tasks in the schedule, and $\boldsymbol{endExecutionTime}(\boldsymbol{t},\boldsymbol{p})$ refers to the end time of task $\boldsymbol{t}$ on processor $\boldsymbol{p}$.

Additionally, Eq. (4) calculates the **flowtime**, denoted as $\boldsymbol{Ft_i}$, which represents the entire completion times of tasks that have already been scheduled. It includes both the processing time required for task execution and any waiting periods that may occur. F**lowtime** is a crucial metric in real-life applications, particularly within industries, as it promotes equitable resource utilization [25]. Calculating $\boldsymbol{Ft_i}$ involves determining the total time taken for a task to finish, considering both its processing time and any potential waiting time. It can be represented as follows:

$$\boldsymbol{Ft_i} = \boldsymbol{endExecutionTime}(\boldsymbol{WA_i}) - \boldsymbol{ArrivalTime}(\boldsymbol{WA_i}) \tag{4}$$

Where $\boldsymbol{endExecutionTime}(\boldsymbol{WA_i})$ refers to the end time of the task $\boldsymbol{WA_i}$, and $\boldsymbol{ArrivalTime}(\boldsymbol{WA_i})$ represents the time when the same task arrives for execution.

The multi-objective fitness function aims to achieve an optimal solution by simultaneously minimizing three key objectives: makespan, flowtime, and cost. The fitness function, formulated as Eq. (5), drives the optimization process towards this goal:

$$\textbf{Minimize}\{\mathbf{M}(\mathbf{x}), \boldsymbol{Ft}(\boldsymbol{x}), \ \boldsymbol{TotalCost}(\boldsymbol{x})\} \tag{5}$$

In this equation, M(x) signifies the makespan, Ft(x) represents the flowtime, and TotalCost(x) corresponds to the total cost associated with the scheduling solution x.

To balance the trade-off between execution time, flowtime, and cost, the multi-objective model incorporates weights for each objective. Eq. (6) illustrates how the fitness function is formulated to achieve this trade-off:

$$\boldsymbol{Fitness} = \boldsymbol{\alpha} * \boldsymbol{TotalCost} + \boldsymbol{\beta} * \boldsymbol{M} + \boldsymbol{\lambda} * \boldsymbol{Ft} \tag{6}$$

Where $(\boldsymbol{\alpha}, \ \boldsymbol{\beta}, \ \boldsymbol{\lambda})$ are the weights assigned to Total Cost, Makespan, and Flowtime, respectively. It's important to note that the sum of these weights must be equal to 1 $(\boldsymbol{\alpha} + \boldsymbol{\beta} + \boldsymbol{\lambda} = 1)$. By adjusting these weights, one can control the emphasis given to each objective during the optimization process. For instance, increasing the weight of the Total Cost will prioritize minimizing the cost, while reducing the weight of the Makespan may lead to shorter execution times at the expense of higher costs. This flexibility enables the model to accommodate different preferences and priorities based on customer choices and specific optimization requirements.

In designing our task scheduling approach, we employ a fitness function that encompasses multiple objectives, including makespan, flowtime, and cost, to ensure a comprehensive evaluation of potential solutions. It is important to note that while our fitness function considers a range of objectives, our primary research focus is twofold: first, to minimize the makespan, thereby reducing the overall time required for workflow completion; and second, to optimize resource utilization by efficiently allocating resources to tasks. These specific objectives align with the core challenges in task scheduling efficiency and cloud resource allocation that we aim to address.

5. Methodology

The proposed approach employs a multi-stage strategy for task scheduling, aiming to leverage parallel hybridization techniques. In the initial stage, the DAG is harnessed to facilitate efficient task allocation to various processors based on their dependencies. This initial step focuses on optimizing the scheduling process and enhancing task completion times. Subsequently, a parallel hybridization strategy is employed by simultaneously integrating the GA and PSO algorithms. This parallel hybridization aims to ensure both the feasibility of the scheduled tasks and efficient exploration of the solution space. The collaboration between GA and PSO involves evaluating task assignments and processor mappings, ensuring their alignment with constraints and dependencies specified in the DAG. Additionally, the HEFT technique is applied to validate task assignment feasibility. Fig. 3 provides an illustrative depiction of the

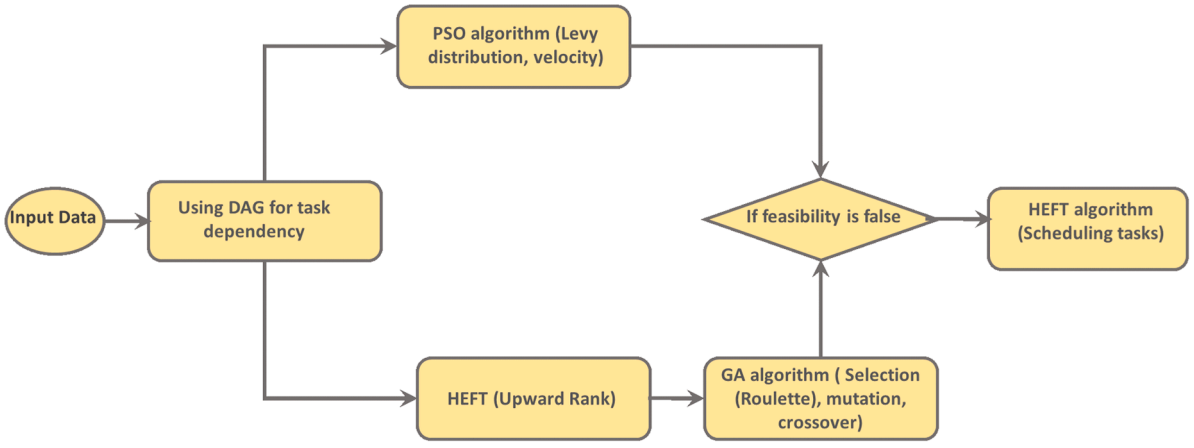


Fig. 3. Execution Sequence of Proposed Algorithm.

algorithm's execution flow.

5.1. Hybridization type

Among the three hybridization methods outlined in Section 3.1—sequential updating, sequential execution, and parallel hybridization—we have opted for the parallel hybridization approach due to its potential to yield enhanced performance and reduced execution times. This selection is grounded in solid reasoning, given that the parallel hybridization method capitalizes on the collective strengths of PSO, HEFT, and GA in a simultaneous and parallel manner. While both the sequential updating and sequential execution hybridization methods offer benefits in leveraging individual optimization techniques, they may fall short of fully exploiting the potential synergies between them. As a result, we have embraced the parallel hybridization approach to unlock the true optimization power of these techniques. This approach ensures that PSO's proficiency in global exploration, HEFT's utilization of the DAG structure, and GA's diversity and adaptability are all actively contributing to the optimization process simultaneously. Additionally, the incorporation of Levy distribution for random initialization in the PSO algorithm provides an innovative touch that augments exploration capacity. Notably, what sets the parallel hybridization method apart is its ability to blend these techniques' advantages and mechanisms cohesively. Furthermore, our algorithm benefits from the fact that HEFT validates the feasibility of both PSO and GA solutions, contributing an extra layer of robustness to the final outcomes.

5.2. Maintaining dependencies

In our proposed algorithm, a DAG workflow is utilized to model the dependencies between tasks. In other words, it plays a role in defining the order of task execution and ensuring that tasks are executed in a way that respects their dependencies. This DAG structure guides the entire scheduling process, ensuring that tasks are scheduled and executed in a way that respects their dependencies. As tasks are scheduled, their dependencies are considered to ensure that no task starts before its dependencies are completed. Both the PSO and GA components of our approach leverage this DAG to generate scheduling solutions. PSO schedules tasks and assigns them to processors based on the DAG's structure, and then GA refines the solutions while maintaining the dependencies.

5.3. Operating the mapping using PSO

This work aims to replace the random generation of position and velocity with a Levy distribution. The key idea behind using a Levy distribution for updating the particle's position and velocity is to introduce a mechanism that enables efficient exploration of the search space. Notably, particle positions were updated using a Gaussian distribution. However, inspired by the observation that many foragers and wandering animals follow a Levy distribution of steps, this distribution was found to be useful for optimization algorithms [26].

The Levy distribution is a probability distribution that describes the behavior of random walks with heavy-tailed steps. It is known for its ability to promote long jumps in the search space, allowing particles to quickly explore new areas and potentially find better solutions. By incorporating the Levy distribution into the PSO algorithm, we aim to enhance the algorithm's ability to efficiently navigate the solution space, leading to more effective exploration of diverse regions and an increased likelihood of discovering high-quality solutions for the multi-objective task scheduling problem in cloud computing.

The main steps of PSO implementation are described as follows:

Step 1: Initialization

- Initialize the swarm of particles using levy distribution, personal best (pbest), and global best (gbest)

Step 2: Velocity Update

- For each particle i in the swarm and each task j:
  - Update the velocity of particle i for task j at time $(t+1)$ using the following equation (7):

$$\mathbf{V}_{\mathbf{ij}}^{\mathbf{t+1}} = \mathbf{w}\mathbf{V}_{\mathbf{ij}}^{\mathbf{t}} + \mathbf{c}_1\mathbf{r}_1\left(\mathbf{P}_{\mathbf{best\ (ij)}}^{\mathbf{t}} - \mathbf{P}_{\mathbf{ij}}^{\mathbf{t}}\right) + \mathbf{c}_2\mathbf{r}_2\left(\mathbf{P}_{\mathbf{bestglobal(ij)}}^{\mathbf{t}} - \mathbf{P}_{\mathbf{ij}}^{\mathbf{t}}\right) \tag{7}$$

Where: $\mathbf{V}_{\mathbf{ij}}^{\mathbf{t+1}}$ is the velocity of particle i for task j at time $(t+1)$. $\mathbf{w}$ is a weight that controls the impact of the current velocity on the update. $\mathbf{c}_1$ and c_2 are acceleration constants controlling the impact of **pbest** and **gbes**t on the update. $\mathbf{P}_{\mathbf{best\ (ij)}}^{\mathbf{t}}$ is the processor allocation of task j for particle i 's personal best position. $\mathbf{P}_{\mathbf{bestglobal(ij)}}^{\mathbf{t}}$ is the processor allocation of task j for the global best position among all particles. $\mathbf{P}_{\mathbf{ij}}^{\mathbf{t}}$ is the current processor allocation of task j for particle i.

Step 3: Position Update

- For each particle i in the swarm:
  - Swap the processor allocations of tasks with the two highest velocities for particle i.
  - Recalculate the fitness of the particle's new position using Eq. (6).

Step 4: Update pbest and gbest

- The algorithm above starts by initializing the swarm with particles, and each particle's fitness is evaluated by Eq. (6). The particle with the best fitness is selected as the global best (gbest). Additionally, each particle maintains its best personal solution (pbest) based on its own fitness.

During each generation, the swarm particles update their velocities and positions (representing scheduling decisions) based on their pbest and gbest values. The velocities are updated using inertia weight to balance global and local exploration. The PSO algorithm uses two constants (C1 and C2) to control the influence of pbest and gbest, respectively, on the particle's velocity. After updating the velocities, the particles swap virtual machines (processors) based on their velocities' top two values. This swap aims to enhance the exploitation and exploration of the search space. The process of updating velocities, swapping virtual machines, and evaluating new solutions continues for multiple generations.

5.4. Task scheduling to VMs by HEFT

5.4.1. Scheduling Cloudlet-Processor

The HEFT method is responsible for generating an initial chromosome. This initial chromosome serves as a starting point for the GA's population. Algorithm 1 shows the procedure for forming the initial population.

This algorithm schedules cloudlets onto processors, considering their dependencies and available resources. It ensures that cloudlets are only scheduled when their dependencies are met and updates the schedule accordingly. The initial phase involves quantifying the initial population of potential solutions, represented by chromosomes. The chromosomes are encoded with task scheduling details, where each gene corresponds to a task and its assigned processor. To ensure diversity, the population is generated with Levy distribution with consideration of task dependencies. Feasibility checks are performed to validate the generated chromosomes, and only feasible solutions are retained. The algorithm iteratively continues this process until all cloudlets are scheduled.

After generating the initial population, the suitable value, which represents the overall execution time of each chromosome, is assessed and employed to rank the chromosomes. The ranking is determined using the HEFT processor allocation technique. Subsequently, a fusion process is initiated by selecting and combining multiple chromosomes.

5.4.2. Upward rank

The concept of **upward rank** is utilized to quantify the priority of the first three chromosomes in a population. This quantification is essential for generating a well-seeded initial population in the GA.

The process of up-ranking is depicted by Eq. (8), which calculates the upward rank value for each chromosome. This rank is determined by considering the dependencies and communication costs between tasks represented by the chromosome [23].

$$\mathbf{upWardRank}_b(WA_i) = mW(\bar{t}_i) + \max_{WA_j \in \mathrm{SUCC}(WA_i)} \left(Cost(WA_i, WA_j) + \mathbf{UpWardRank}_b(WA_j) \right) \tag{8}$$

Where $mW(\bar{t}_i)$ represents the average computational cost (mean weight) of the task WA_i .

$\max_{WA_j \in \mathrm{SUCC}(WA_i)}$ is the maximum operator applied to the summation over all successor tasks WA_j of WA_i. $Cost(WA_i, WA_j)$ represents the communication cost between tasks $WA_i$and WA_j . $\mathbf{UpWardRank}_b(WA_j)$: Denotes the upward rank of each successor task WA_i of WA_j .

In summary, the HEFT algorithm prepares the initial solution for the Genetic Algorithm by applying algorithm 2 to prioritize and schedule tasks based on their upward ranks. In other words, it is employed to rank the chromosomes based on total execution time, contributing to finding better solutions. The algorithm leverages the upward ranks, dependencies, and communication costs to efficiently allocate tasks and optimize the makespan, flowtime, and cost.

Generate Chromosome algorithm plays a crucial role in establishing an effective task-to-processor mapping and defining relationships among tasks in DAG workflow, forming the foundation for scheduling decisions. To begin, the chromosome and data dependencies in the DAG workflow are initialized (lines 1). This chromosome represents a task-to-processor mapping, while the data dependencies outline the relationships among tasks, which are essential for making informed scheduling choices. Additionally, the average processing cost for each task is calculated based on its processing costs on different processors. Upward ranks array,

Algorithm 1
Scheduling Cloudlet to Processor.

```
1.  Initialize schedule with empty values
2.  For each processor_schedule in schedule:
3.      g=Creation of empty gene (cloudlet = 0,processor = 0)
4.      tsd= Creation of this gene cloudlet scheduling details (g)
5.      processor_schedule.add(tsd)
6.  End for
7.  Add genes to processor queues and cloudlet to processor mapping
8.  Check for Completed cloudlets.
9.      For i from 1 to NbrOfProcessor:
10.       Map cloudlet to processor
11.       Verify the satisfaction of dependencies between cloudlets
12.       If dependencies satisfied
13.         Schedule cloudlets
14.       End if
15.       Update the schedule of cloudlet to processor
16.     End for
17. Check the processor queue is empty
```

Algorithm 2
Generate chromosome.

```
Output: Chromosome
1.  Initialization of chromosome schedule
2.  For i from 0 to NbrOfCloudlet:
3.      Calculate upward ranks using Eq. (8)
4.  endFor
5.  Sort Cloudlet based on the value of upward ranks
6.  For i from 0 to NbrOfCloudlet
7.      Add completed cloudlet
8.  End for
9.  For i = 0 to nbrOfProcessor:
10.     Add scheduled chromosome
11. End for
12. For each cloudlet in upward ranks array:
13.         For each dependency d in dependencies of cloudlet:
14.             comm= calculate communication cost (d, cloudlet)
15.         For each processor p from 1 to nbrOfProcessor:
16.             Determine finish time on each processor
17.         End for
18. Schedule cloudlet on the processor that yields the minimum end time.
19. Update Chromosome schedule
20. Update list of finished cloudlet
21. return Chromosome
```

representing task dependencies and computation needs, providing valuable information for the scheduling process. For each task in the workflow, a gene is created with the task and processor set to 0, serving as a container to store the task-to-processor assignment. The upward rank for each task is then computed based on its dependencies and computation requirements (line 3). To ensure optimal scheduling and minimize total execution time, the tasks are sorted based on their upward ranks (line 5), giving priority to those with higher dependencies.

In the subsequent steps (lines 6–11), the algorithm iterates through the tasks and takes into account communication costs and dependencies to schedule them on suitable processors. For each task, the method evaluates the execution times on different processors while considering the communication delay with dependent tasks (14–16). To find the best processor for each task, the algorithm selects the one with the minimum end time (lines 19–20). It then updates the chromosome's schedule and the list of completed tasks accordingly, ensuring that an efficient task mapping to processors is achieved.

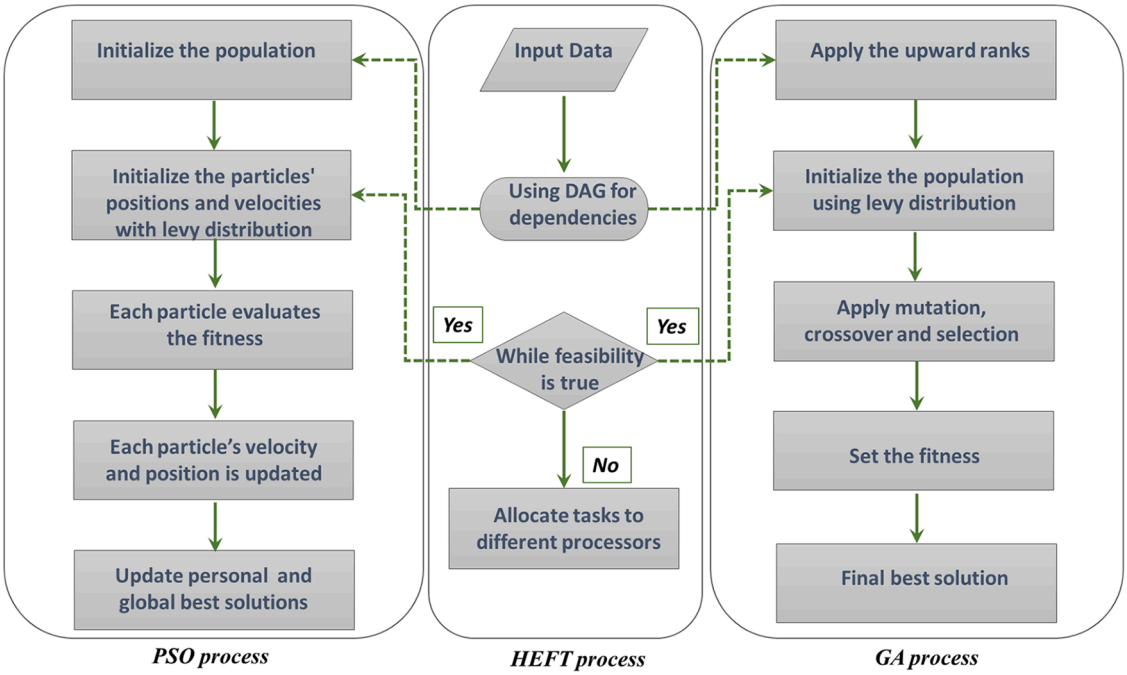


Fig. 4. The flowchart of the proposed method.

5.5. Refining the mapping using GA

In the GA section, every chromosome is assessed and assigned a rating based on its total execution time. This evaluation occurs immediately after generating the initial population of chromosomes. This ranking is referred to as the fitness value of each chromosome and is determined by employing the HEFT processor allocation algorithm. Subsequently, a fusion process is initiated by combining multiple chromosomes. Once the fusion step is finished, the parallelism approach is applied to the chromosomes. This entire process is iterated until the termination condition is satisfied.

The GA then advances to the evolution phase, where crossover and mutation operations are implemented to enhance the chromosomes' fitness. During the crossover process, two parent chromosomes are combined, giving rise to offspring that inherit genetic information from both parents. Mutation introduces minor random alterations in the offspring, promoting exploration within the search space. Selection employs roulette wheel selection, favoring chromosomes with higher fitness values for inclusion in the next generation. This iterative process improves the convergence of the GA towards superior solutions. Over numerous generations, the GA refines the population, gradually converging toward an optimal solution. After multiple iterations, the algorithm ultimately converges to a final chromosome that represents the best solution to the task scheduling predicament within the cloud environment.

The key steps in the genetic algorithm are as follows:

The flowchart of the proposed method is depicted in Fig. 4, illustrating the multi-stage process that integrates the collaborative strengths of PSO, HEFT, and GA. Notably, it's important to emphasize that feasibility checks are performed throughout the algorithm's execution, ensuring that scheduling solutions maintain their validity at each step.

5.6. Complexity analysis

This section presents a thorough complexity analysis of the suggested algorithm, illustrating the complexity of its individual steps. The fundamental organization of the algorithm is shown, highlighting its key elements and the connections between them. Detailed complexity analyses of each algorithmic technique are expounded upon in the subsequent section, accompanied by insights into computational requirements and other factors influencing the efficiency of each method.

The complexity analysis of the Chromosome step is determined to be $O(nbrOfProcessors * (nbrOfCloudlets + dependencies\ per\ Cloudlet))$. Moving forward, the Swarm step's complexity remains at $O(nbrOfSwarm * nbrOfCloudlets)$, remaining unaffected by variations in input data or dependencies between cloudlets and processors. The Particle complexity is calculated as $O(nbrOfCloudlet * (number\ of\ dependencies\ for\ each\ cloudlet))$. The Heft stage, which oversees hierarchical scheduling, exhibits an overall complexity of $O(nbrOfCloudlets * nbrOfProcessors + nbrOfCloudlets * (number\ of\ dependents\ for\ each\ job) + nbrOfCloudlets * log(nbrOfCloudlets) + nbrOfCloudlets * (number\ of\ data\ sizes\ for\ each\ cloudlet))$. This sophisticated estimation represents the complex interplay between task processing costs, dependencies, and cloudlet setups.

The Population phase, a pivotal aspect of the algorithm, exerts a significant influence on its complexity. Responsible for establishing the framework to generate and maintain candidate solutions, this phase carries an estimated complexity of $O(200 * (n * limit))$, where n represents the population's number of chromosomes and limit is the lower value between populations. The interaction between population size, chromosomal evaluation, and convergence criteria, which illuminates the algorithm's search and optimization

Table 2
Cloudsim Configuration.

Data Center Configuration	
Name of Attribute	**Value**
Architecture	×86
OS	Linux
Time-Zone	10.0
Cost per process	3.0
Cost per Memory	0.05
Cost per Storage	0.001
Costper Bandwidth	0.1
Host Configuration	
Storage	100,000 MB
Host_mips	1000
Host RAM	4096 MB
Host Bandwidth	100,000 Gbps
VM Configuration	
VM Image Size	10,000 MB
VM_RAM	512 MB
VM_MIPS	1000
VM_Bandwidth	1000 Gbps
VM_PES	5
VMM_NAME	Xen
Cloudlet Configuration	
File size/ Output size	1000
Length	2000–8000
PES	3

process, collectively contributes to the intricate complexity of the problem.

6. Results and discussion

6.1. Simulation environment

For the evaluating our suggested algorithm, we implemented the HEPGA algorithm using CloudSim. CloudSim is widely recognized and extensively utilized in the cloud computing research community. Its extensibility and the presence of a supportive community made it an attractive and practical choice for our research, offering the flexibility needed to adapt the simulator to our specific requirements. Notably, we included a Directed Acyclic Graph (DAG) to model complex task dependencies, making CloudSim ideal for our research focused on multi-objective task scheduling in cloud computing [27].

To assess the performance of the proposed HEPGA algorithm, we compared it with existing work scheduling algorithms, such as the GA used in [6], Hybrid Genetic Algorithm (HGA) proposed in [4], and the PSO.

The initial configuration of CloudSim involved several parameters related to the data center setup, as illustrated in Table 2. This configuration includes diverse attributes along with their respective values, which play a crucial role in effectively configuring and executing the simulation process.

To accurately model communication and computation costs within our simulation, two fundamental graphs, G1n $\times$ n and G2n $\times$ m, are constructed:

6.1.1. Graph G1n $\times$ n (Communication cost graph)

For each task within the DAG representing the workflow, communication costs are generated. These costs emulate the time required for data transfer between tasks. The communication costs are randomly generated within a range of 1 to 10-time units, reflecting the realistic variability in data transfer times. Each communication cost corresponds to an edge in the DAG, symbolizing the interaction between tasks.

6.1.2. Graph G2n $\times$ m (Computing cost graph)

To capture the execution times of tasks on different available processors, computing costs are assigned. These costs mirror the execution durations of tasks based on the processing power of each processor and the complexity of the tasks. The computing costs are drawn from a predefined range of 5 to 20 time units. This range reflects the diversity in execution times arising from factors such as resource capacities and task intricacies. In our simulation, computing costs are derived from the processing cost values defined in the cloudlet information. The processing cost values are tailored to the specific simulation scenario, taking into account the number of processors available (3 in this case). These values are utilized to shape the execution durations of tasks on different processors, ultimately guiding the scheduling decisions.

The selected range of 5 to 20-time units for computing costs aligns with the nature of our simulation, which involves tasks executed on 3 processors. This range ensures a credible representation of varying execution durations, contributing to the realism and accuracy of our simulation environment.

The algorithm commenced with a population of 100 genes. In GA, we set the crossover rate to 0.9 and the mutation rate to 0.5. A higher crossover rate promotes exploration by creating diverse offspring, while a higher mutation rate enhances exploration by introducing randomness. The chosen values provide a reasonable trade-off between these two aspects. In the PSO algorithm phase, the initial weight in PSO was set to 0.9. This value is known to balance exploration and exploitation. A higher weight encourages exploration, allowing particles to search a broader solution space early on. We set the acceleration coefficients (C1), (C2), and the inertia weight, as specified in Table 3.

We chose 100 iterations for PSO and 200 iterations for GA based on empirical testing. Increasing the number of iterations may enhance GA's performance, but it should be balanced with computational resources.

Table 3
PSO/ GA algorithm parameters.

PSO parameters	
Description	**Values**
Maximum number of iteration	100
Initial weight	0.9
C1 Personal coefficient	1
C2 Global coefficient	1.1
Swarm size	100
GA parameters	
Maximum number of iteration	200
Population size	100
Crossover	0.9
Mutation	0.5

6.2. Simulation results and discussion

This sub-section provides insights into the performance of the proposed scheduling algorithm by analyzing the results obtained from different parameter combinations. The evaluation is conducted using a fitness function that incorporates three key objectives: minimizing the Total Cost, reducing the Makespan (M), and optimizing the Flow Time (FT). The parameter weights for the objectives are denoted as α, β, and λ respectively, where their values determine the relative importance of each objective within the fitness function. The evaluated parameter combinations demonstrate the trade-offs between objectives. For instance, a higher β value may prioritize minimizing the Makespan, while a higher α or λ value may emphasize reducing the Total Cost or optimizing Flow Time. The evaluation was conducted by varying the weight values for each objective in our fitness function, as shown in Table 4.

Table 5 provides scenarios for conducting a comprehensive evaluation of our algorithm's performance across different resources.

Although our fitness function is multiobjective, we emphasize that the objectives evaluated in our extensive simulations are centered on makespan and resource utilization. This choice is driven by the need to prioritize the performance metrics most relevant to our study while maintaining the flexibility to accommodate various scheduling aspects in our fitness function. However, it is important to note that, apart from the primary objectives of makespan reduction and resource utilization optimization, we are open to assessing other metrics to improve the efficiency of our algorithm under different criteria. In the following sections, we will outline our experimental methodology, with a consistent focus on these primary objectives and a readiness to explore additional metrics for a comprehensive evaluation.

6.2.1. Makespan

The makespan is the total time taken for the execution of all tasks in the schedule. A lower makespan indicates that the tasks are completed faster, which is generally desirable.

The performance of the scheduling algorithm is significantly influenced by the selection of weight parameters within the fitness function. Fig. 5 provides a visual representation of the results obtained from various weight settings, shedding light on the sensitivity of each scenario to different weights and their effects on Makespan.

Analyzing the experiment results, we've identified noteworthy trends regarding the relationship between varying resources and the algorithm's performance. Specifically, our observations indicate that with an increase in the number of resources, the Makespan tends to increase as well, except for a unique trend observed in scenario 4.

Scenario 1, illustrated in Fig. 5, entails a restricted resource setting with only 2 hosts and 5 VMs. In this context, the most favorable outcomes were realized when assigning a higher weight to the cost objective. This observation suggests that in situations of resource scarcity, prioritizing cost optimization leads to advantageous results.

Moving on to Scenario 2 and Scenario 3 (Fig. 5), in both of these scenarios (10 hosts and 50 VMs). Optimal results were obtained when the cost and flow time objectives were approximately balanced. However, a noteworthy distinction emerged between the two scenarios. In Scenario 2, increasing the weight for flow time led to a remarkable improvement in makespan achieved by the HEPGA algorithm. This observation is intriguing as a heightened focus on flow time prompted the algorithm to allocate tasks effectively, consequently reducing makespan. Conversely, in Scenario 3, marked by a larger quantity of cloudlets (1000 cloudlets), assigning a lower weight to flow time contributed to reduced makespan. This finding highlights that in scenarios with fewer cloudlets, prioritizing a diminished flow time becomes more advantageous, directly influencing makespan.

Scenario 4, depicted in Fig. 5, showcased distinctive behaviors in the context of ample resources (20 hosts and 150 VMs). The best makespan was achieved with a weight of 0.6 assigned to makespan and equal weights for cost and flow time. This implies that in scenarios with abundant resources, a balanced approach across the objectives tends to yield optimal makespan outcomes.

Fig. 6 presents the Makespan achieved by the four scheduling algorithms, HEPGA, GA, HGA, and PSO, across different test scenarios. In scenario 1, as illustrated in Fig. 6, with a relatively small number of hosts and VMs, HEPGA consistently demonstrated superior performance in terms of Makespan optimization compared to both PSO and GA across varying weight settings. Notably, GA consistently achieved lower Makespan values than those obtained with PSO. This disparity can be attributed to GA's inherent robust exploration capability, facilitated by its utilization of crossover and mutation operations. These mechanisms enable GA to effectively explore different regions of the search space, which becomes particularly advantageous in scenarios with limited resources. In contrast, PSO might encounter challenges in achieving the same degree of exploration due to its reliance on particle positions and velocities.

Table 4
Varying the fitness weights.

Scenario	α (Total Cost)	β (Makespan)	λ (Flowtime)	Description
Scenario 1	0.6	0.2	0.2	Balanced weights reflecting equal importance to makespan, and flowtime.
Scenario 2	0.5	0.3	0.2	Moderately emphasizing total cost while considering both makespan and flowtime.
Scenario 3	0.4	0.4	0.2	Giving equal weight to makespan and total cost, with a slight consideration for flowtime.
Scenario 4	0.7	0.1	0.2	Significantly prioritizing total cost while accounting for flowtime.
Scenario 5	0.3	0.2	0.5	Prioritizing flowtime for applications with critical time-sensitive requirements.
Scenario 6	0.4	0.1	0.5	Focusing on flowtime while maintaining some attention to the total cost.
Scenario 7	0.2	0.6	0.2	Emphasizing makespan for situations where rapid task completion is crucial.
Scenario 8	0.1	0.2	0.7	Prioritizing flowtime significantly for real-time applications.
Scenario 9	0.3	0.7	0.0	Highlighting makespan in scenarios where minimizing task completion time is paramount.
Scenario 10	0.2	0.3	0.5	Striking a balance between makespan and flowtime with moderate consideration for total cost.

Table 5
Varying the number of resources.

Experiment	Number of Hosts	Number of VMs	Number of Cloudlets
Scenario 1	2	5	100
Scenario 2	10	50	600
Scenario 3	10	50	1000
Scenario 4	20	150	1000

Fig. 5. Evaluation of Makespan by varying the weights in the fitness function.

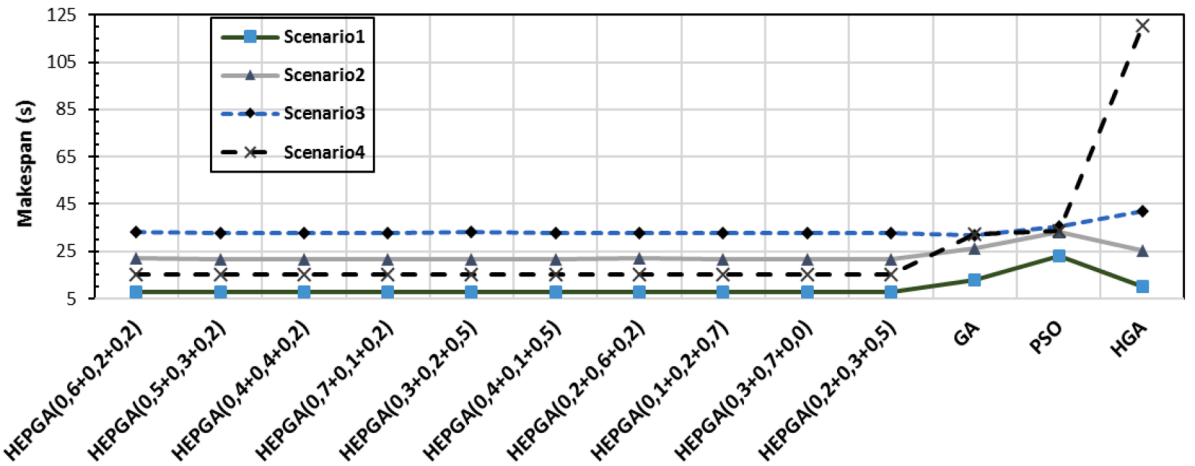


Fig. 6. Comparative Analysis of Scheduling Algorithms Using Makespan Performance.

While PSO is proficient in local search and exploitation, its global exploration capability might be comparatively constrained in resource-limited scenarios. This could be a contributing factor to the higher Makespan values observed with PSO.

The scenario 2, where an increased number of hosts and VMs are considered, the Makespan values for HEPGA, GA, and PSO are higher compared to those observed in Scenario 1. Intriguingly, when assigning a relatively higher weight to flow time, the HEPGA algorithm demonstrates improved Makespan values. This underlines the potential benefits of simultaneously leveraging GA, HEFT, and PSO techniques, especially when optimizing for flow time is of significant importance.

Moving on to scenario 3, as visualized in Fig. 6, the Makespan values for both GA and PSO remain relatively high. Surprisingly, GA outperformed both HEPGA and PSO, achieving the lowest Makespan value. The increased complexity introduced by the greater number of cloudlets may have led to a more intricate optimization landscape, where GA's exploration and optimization mechanisms proved to be particularly effective. While HEPGA showcased competitive performance across most scenarios, this outcome underscores the sensitivity of algorithm performance to the specific characteristics of the problem instance and the interplay between various parameters. It reinforces the understanding that the effectiveness of different optimization algorithms can vary based on the size and complexity of the problem.

Finally, considering scenario 4, with even more resources available, it was observed that the Makespan values are generally lower. Interestingly, the optimal Makespan for HEPGA is achieved when the Makespan weight is set to 0.6, with equal weights for cost and flow time. This suggests that in environments with abundant resources, a balance among the three objectives yields favorable outcomes. The improvement in Makespan achieved by HEPGA in comparison to GA and PSO further highlights the potential of hybridization in enhancing task scheduling optimization.

Nevertheless, we observed a notable increase in Makespan values for HGA in scenario 4, where resources are more abundant, in contrast to scenario 1 where it outperformed PSO and GA. This observation suggests that HGA might be better suited for scenarios with fewer resources, where its exploration and optimization capabilities can shine.

In summary, the presented results emphasize the efficacy of the HEPGA algorithm in generating schedules that require less time for task completion. This indicates the algorithm's capability to efficiently allocate tasks to available resources, leading to minimized execution times. The optimal weight combinations vary between algorithms due to their distinct optimization strategies. The results highlight the importance of tuning weight parameters to achieve optimal performance for each objective. This underscores the efficiency of HEPGA, especially in resource-constrained scenarios, while also indicating potential areas for further refinement and adaptation in resource-rich environments.

6.2.2. *Resource utilization (RU)*

RU represents how efficiently the available resources are utilized. A higher RU value indicates better resource usage. It suggests that resources are efficiently utilized without excessive idle time.

In the context of our Hybrid HEPGA, the Resource Utilization metric provides valuable insights into the efficiency and utilization of resources across different weight settings. RU serves as a performance indicator, reflecting how effectively the algorithm allocates

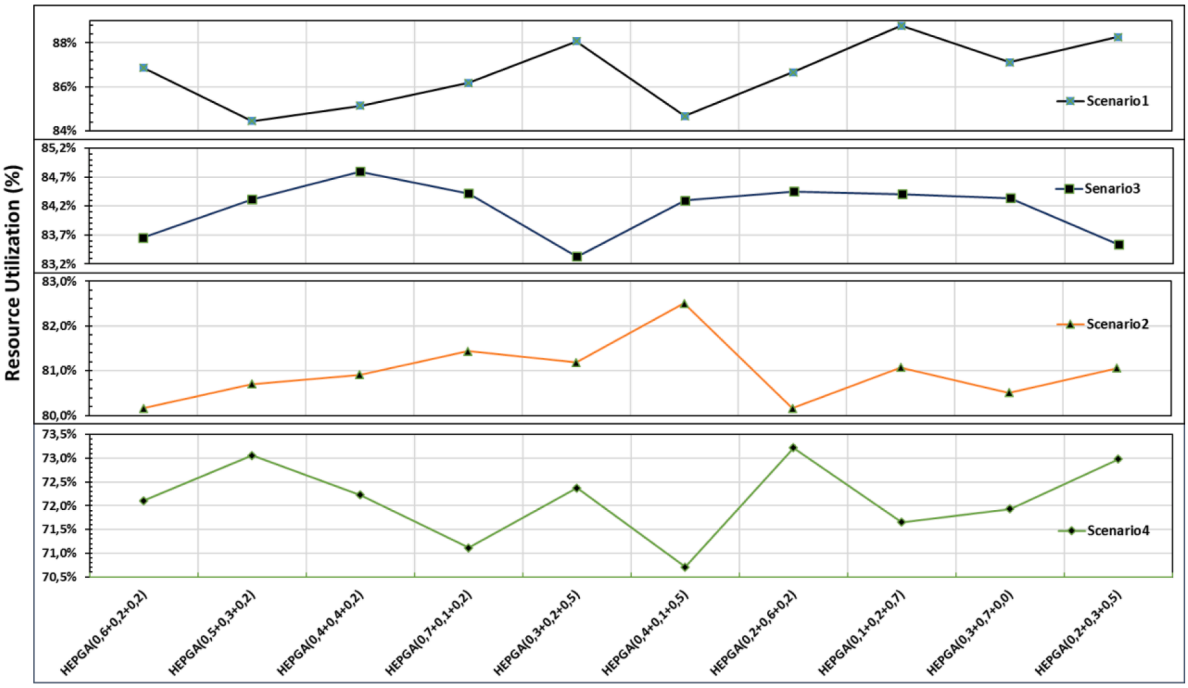


Fig. 7. Evaluation of RU by varying the weights in the fitness function.

resources while minimizing Makespan, Flow Time, and Cost.

In scenario 1, depicted in Fig. 7, characterized by a relatively small number of hosts and VMs, the RU values for HEPGA exhibit a consistent trend. Higher weights assigned to Cost, Flow Time, and Makespan objectives correspond to higher RU values. This suggests that when resources are constrained, prioritizing any single objective leads to better resource utilization, indicating a more balanced allocation of computational resources.

Scenario 2, as illustrated in Fig. 7, with an increased quantity of hosts and VMs, shows an interesting trend that emerges in HEPGA's RU values. Here, a harmonious approach to optimizing both Cost and Flow Time objectives yields notably improved RU values. In contrast to scenario 1, where distinct objective prioritization led to greater RU efficiency, scenario 2′s findings point to the benefits of striking a balance between Cost and Flow Time optimization. This intriguing outcome can be attributed to the larger pool of available resources. As resource capacity increases, HEPGA effectively capitalizes on this abundance to distribute tasks optimally among processors, leading to enhanced resource utilization and more efficient task scheduling. Moreover, the algorithm's adaptability to various weight settings further underlines its capacity to adapt to varying resource scales while optimizing objectives. This insight emphasizes the algorithm's potential to harness the synergy between Cost and Flow Time optimization to achieve superior RU values in resource-rich environments.

Shifting to scenario 3, depicted in Fig. 7, marked by an amplified number of cloudlets, RU values for HEPGA exhibit a distinct behavior as the Flow Time weight is increased. In contrast to scenario 2, where higher Flow Time weight correlated with improved RU values, scenario 3 showcases more varied tendencies. RU values fluctuate, suggesting a more intricate relationship between Flow Time weight and resource utilization. This observation could be attributed to the complex interplay between multiple factors, such as task dependencies, resource availability, and scheduling constraints. The results highlight the intricate nature of resource utilization optimization, particularly in scenarios with a larger number of cloudlets. Further analysis may reveal additional insights into the algorithm's behavior and its adaptability to different objectives and resource configurations.

Moving to scenario 4, as illustrated in Fig. 7, with a substantial augmentation in resources (hosts, VMs, and cloudlets), we observe RU values being influenced by the assigned weights within the fitness function. RU values exhibit minor fluctuations as the weights for Cost, Makespan, and Flow Time objectives are adjusted. Notably, HEPGA demonstrates consistent RU values across various weight settings, highlighting its prowess in efficient resource allocation. The RU values, although not following a linear trend, remain within a relatively close range, suggesting that HEPGA maintains a balanced utilization of resources across various weight configurations. This adaptability to varying weight distributions underscores the algorithm's robustness and versatility in addressing scheduling complexities associated with diverse priorities and resource requirements.

Overall, the RU values for HEPGA illustrate its capacity to adapt resource allocation based on the relative importance of objectives. Higher RU values indicate that the algorithm allocates resources effectively, resulting in improved task scheduling and optimization of objectives. As the weights shift, HEPGA demonstrates its versatility in achieving optimal resource utilization across different scenarios, contributing to more efficient cloud computing environments.

In scenario 1, as depicted in Fig. 8, it is observed that HEPGA consistently outperforms both GA and PSO in terms of Resource Utilization (RU) optimization across varying weight configurations. While HGA and GA also attains higher RU values than PSO, its RU values are lower than those of HEPGA. This underscores HEPGA's greater effectiveness in resource utilization, resulting in more significant RU enhancements compared to the other two algorithms in this resource-constrained scenario. The exploration and exploitation mechanisms of HEPGA appear well-suited for effectively maximizing limited resources.

With an increased number of hosts, VMs, and cloudlets in scenario 2, HEPGA continues to exhibit competitive RU values, as illustrated in Fig. 8. Notably, the RU values of PSO experience a significant improvement, making it a more effective resource utilization option in this setting. This could be attributed to PSO's ability to explore and exploit the search space more efficiently as the complexity of the problem increases. It's worth mentioning that HGA consistently achieved higher RU values compared to both PSO

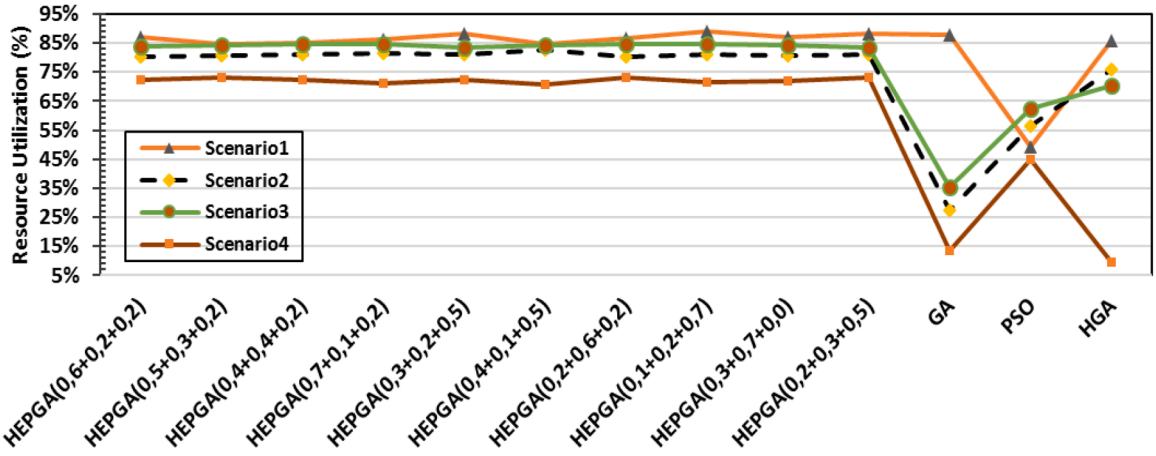


Fig. 8. Comparative Analysis of Scheduling Algorithms Using RU Performance.

and GA, demonstrating its strength in resource utilization.

Moving on to scenario 3, characterized by even larger numbers of hosts and VMs, we observe further developments in the results. HEPGA continues to demonstrate competitive RU values, suggesting its capability to effectively utilize resources in resource-rich environments. Both GA and PSO achieve higher RU values compared to other scenarios, they also exhibit an increase in their RU values, indicating that they are becoming more efficient at resource utilization. This trend can be attributed to the algorithms' adaptation to the greater complexity of the problem and the availability of more resources. However, it is noteworthy that HGA exhibits relatively lower RU values compared to Scenario 2. This suggests that HGA might not perform as effectively in scenarios with a large number of hosts and VMs. Nevertheless, it still outperforms PSO and GA in terms of RU, highlighting its suitability for scenarios of moderate complexity. This nuanced performance variation across scenarios emphasizes the importance of selecting an algorithm that aligns with the specific characteristics of the problem and available resources

Finally, analyzing RU values across various weight combinations for Cost, Flow Time, and Makespan objectives, HEPGA consistently exhibits RU values ranging from approximately 71.92% to 73.00%. These RU values suggest efficient resource utilization with a balanced consideration of multiple objectives. HEPGA's intelligent allocation of tasks optimizes resource utilization, contributing to the overall enhancement of the scheduling algorithm's performance. In contrast, both PSO and GA display higher RU values, specifically 0.448678679 for PSO and 0.135973591 for GA. This indicates that while PSO and GA can still achieve reasonable resource utilization, their efficiency in utilizing resources is notably lower compared to HEPGA. It is worth noting that HGA exhibits the lowest RU values, making it the least suitable choice for scenarios involving large data centers. In this context, HGA lags behind PSO and GA in terms of RU performance.

Table 6 presents the average percentage improvement of HEPGA over GA, HGA, and PSO for both Resource Utilization (RU) and Makespan metrics. These values represent the average improvements across different scenarios with varying weight combinations.

HEPGA consistently outperforms the other algorithms, demonstrating significant average improvements. In terms of task completion time (Makespan), HEPGA is 26% better than the GA algorithm, 41% better than the PSO algorithm, and 36% better than HGA. In the context of Resource Utilization (RU) optimization, the HEPGA algorithm excels, boasting a remarkable 51% improvement over the GA algorithm, 34% better than the PSO algorithm, and 28% better than HGA.

The findings of our study underscore the significance of the Hybrid HEPGA algorithm in achieving superior Resource Utilization outcomes. Notably, across all scenarios, HEPGA consistently demonstrates RU values that outperform GA, HGA and PSO. The adaptability of HEPGA to varying weight distributions further highlights its capacity to address scheduling challenges with differing priorities and resource demands.

Our study has identified key tendencies that guide the effective utilization of resources in cloud computing environments:

- In scenarios with limited resources, prioritizing any single objective (Cost, Flow Time, or Makespan) leads to improved RU values. This emphasizes the importance of maintaining a balanced allocation of computational resources to achieve better overall efficiency.
- In scenarios with larger resource pools, a balanced approach toward optimizing objectives, particularly Cost and Flow Time, leads to notably higher RU values. HEPGA's adaptability to leverage abundant resources for more effective task distribution results in enhanced resource utilization.
- In scenarios with an increased number of cloudlets, the relationship between Flow Time weight and RU values becomes more intricate. Task dependencies, resource availability, and scheduling constraints play a crucial role in influencing RU outcomes. Additionally, the effect of Makespan weight on RU values is less pronounced, with relatively consistent RU values observed across different Makespan weights.
- HEPGA's consistent RU values across different weight configurations in various scenarios highlight its robustness in efficiently managing resource allocation. The algorithm's ability to adapt to changing weight distributions underscores its versatility in addressing diverse scheduling challenges.

6.3. Limitations

The primary limitation of our approach lies in the potential trade-off between optimization objectives. While our method effectively minimizes makespan and optimizes resource utilization, there may be situations where the optimization of one objective slightly impacts the other. For instance, optimizing for makespan may lead to a marginal increase in resource utilization in certain scenarios. We recognize that this trade-off is inherent to multi-objective optimization and is not unique to our approach. To further clarify, we elaborate on specific scenarios or conditions where this trade-offs may be more pronounced and provide a comprehensive discussion of the challenges involved. Additionally, we present practical strategies to mitigate these limitations and strike a balance between our dual optimization objectives. We believe that by addressing these aspects in detail, we can offer a more complete understanding of the proposed methods' limitations and how they relate to real-world cloud computing scenarios.

7. Conclusion and future work

This study presented a novel Hybrid HEFT-PSO-GA (HEPGA) for task scheduling in cloud computing environments. The research sheds light on the intricate interplay between optimization objectives, resource allocation, and algorithmic behavior. HEPGA consistently demonstrates robust Resource Utilization (RU) values across diverse weight configurations, showcasing its efficiency in allocating resources and accommodating varying priorities. The study emphasizes the significance of parameter selection within the

Table 6
Percentage improvement of HEPGA compared to GA, HGA and PSO.

Algorithms	Metrics	HEPGA (0,6 + 0,2 + 0,2)	HEPGA (0,5 + 0,3 + 0,2)	HEPGA (0,4 + 0,4 + 0,2)	HEPGA (0,7 + 0,1 + 0,2)	HEPGA (0,3 + 0,2 + 0,5)	HEPGA (0,4 + 0,1 + 0,5)	HEPGA (0,2 + 0,6 + 0,2)	HEPGA (0,1 + 0,2 + 0,7)	HEPGA (0,3 + 0,7 + 0,0)	HEPGA (0,2 + 0,3 + 0,5)
GA	**Makespan**	25,84%	26,80%	26,51%	26,86%	26,49%	26,81%	26,63%	26,51%	26,54%	26,87%
	RU	50,97%	50,45%	50,93%	50,93%	51,40%	50,54%	46,19%	51,68%	51,15%	51,51%
PSO	**Makespan**	40,18%	41,00%	40,80%	40,92%	40,70%	40,97%	40,83%	40,72%	40,77%	41,03%
	RU	34,14%	34,20%	34,29%	34,25%	34,54%	34,10%	34,52%	34,71%	34,37%	34,72%
HGA	**Makespan**	35,67%	36,44%	36,17%	36,79%	36,26%	36,66%	36,30%	36,39%	36,25%	36,56%
	RU	27,42%	27,08%	27,43%	27,74%	27,99%	27,55%	27,62%	28,39%	27,76%	28,09%

fitness function, revealing the nuanced relationships between weight combinations and algorithmic performance. To comprehensively evaluate the algorithm's effectiveness, we conducted tests across varied scenarios, manipulating weight configurations within the fitness function, and varying the number of resources. The observed variations in Makespan performance based on different weights underscore HEPGA's potential for excellence when aligned with specific optimization goals. The analysis of RU values provides valuable insights into the effective balancing of competing objectives, such as Flow Time, and Cost, while efficiently utilizing available computational resources. As future work, the suggested avenues for exploration include advanced optimization techniques, dynamic weight adaptation, and altering the working environment. However, further investigation is recommended to delve into the nuanced interactions between different weight combinations and their specific influence on resource utilization and Makespan trends.

Data availability

No data was used for the research described in the article.

Acknowledgements

The authors thank the anonymous reviewers for their valuable comments, which have helped us to considerably improve the content, quality, and presentation of this article.

References

[1] S. El Kafhali, I. El Mir, K. Salah, M. Hanini, Dynamic scalability model for containerized cloud services, Arab. J. Sci. Eng. 45 (2020) 10693–10708.
[2] H. Mikram, S. El Kafhali, Y. Saadi, Metaheuristic Algorithms Based Server Consolidation for Tasks Scheduling in Cloud Computing Environment, in: The International Conference on Artificial Intelligence and Computer Vision, Cham: Springer Nature, Switzerland, 2023, pp. 477–486.
[3] H. Mikram, S. El Kafhali, Y. Saadi, Performance Analysis of Scheduling Algorithms for Virtual Machines and Tasks in Cloud Computing, in: The International Conference of Advanced Computing and Informatics, Springer International Publishing, Cham, 2022, pp. 278–289.
[4] S.G. Ahmad, C.S. Liew, E.U. Munir, T.F. Ang, S.U. Khan, A hybrid genetic algorithm for optimization of scheduling workflow applications in heterogeneous computing systems, J. Parallel Distrib. Comput. 87 (2016) 80–90.
[5] H. Mikram, S. El Kafhali, Y. Saadi, Processing Time Performance Analysis of Scheduling Algorithms for Virtual Machines Placement in Cloud Computing Environment, in: International Conference On Big Data and Internet of Things, Springer International Publishing, Cham, 2022, pp. 200–211.
[6] A.M. Manasrah, H. Ba Ali, Workflow scheduling using hybrid GA-PSO algorithm in cloud computing, Wireless Commun. Mob. Comput. 2018 (2018) 1–16.
[7] H. Mikram, S. El Kafhali, Y. Saadi, Server consolidation algorithms for cloud computing: taxonomies and systematic analysis of literature, Int. J. Cloud Appl. Comput. (IJCAC) 12 (1) (2022) 1–24.
[8] J.E.N. Mboula, V.C. Kamla, C.T. Djamegni, Cost-time trade-off efficient workflow scheduling in cloud, Simul. Modell. Pract. Theory 103 (2020), 102107.
[9] Z. Peng, P. Pirozmand, M. Motevalli, A. Esmaeili, Genetic Algorithm-Based Task Scheduling in Cloud Computing Using MapReduce Framework, Math. Probl. Eng. (2022) 2022.
[10] K. Shao, Y. Song, B. Wang, PGA: A New Hybrid PSO and GA Method for Task Scheduling with Deadline Constraints in Distributed Computing, Mathematics 11 (6) (2023) 1548.
[11] S.M. Mirmohseni, A. Javadpour, C. Tang, LBPSGORA: create load balancing with particle swarm genetic optimization algorithm to improve resource allocation and energy consumption in clouds networks, Math. Probl. Eng. 2021 (2021) 1–15.
[12] A. Iranmanesh, H.R. Naji, DCHG-TS: a deadline-constrained and cost-effective hybrid genetic algorithm for scientific workflow scheduling in cloud computing, Clust. Comput. 24 (2021) 667–681.
[13] P. Pirozmand, A.A.R. Hosseinabadi, M. Farrokhzad, M. Sadeghilalimi, S. Mirkamali, A. Slowik, Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing, Neural. Comput. Appl. 33 (2021) 13075–13088.
[14] R. Choudhary, S. Perinpanayagam, Applications of Virtual Machine Using Multi-Objective Optimization Scheduling Algorithm for Improving CPU Utilization and Energy Efficiency in Cloud Computing, Energies 15 (23) (2022) 9164.
[15] G.L. Stavrinides, H.D. Karatza, Resource Allocation and Scheduling of Real-Time Workflow Applications in an IoT-Fog-Cloud Environment, in: 2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC), IEEE, 2022, pp. 1–8.
[16] V. De Maio, D. Kimovski, Multi-objective scheduling of extreme data scientific workflows in fog, Fut. Gen. Comp. Syst. 106 (2020) 171–184.
[17] Z. Zhu, G. Zhang, M. Li, X. Liu, Evolutionary multi-objective workflow scheduling in cloud, in: IEEE Transactions on parallel and distributed Systems 27, 2015, pp. 1344–1357.
[18] F. Abazari, M. Analoui, H. Takabi, S. Fu, MOWS: multi-objective workflow scheduling in cloud computing based on heuristic algorithm, Simul. Modell. Pract. Theory 93 (2019) 119–132.
[19] H.R. Faragardi, M.R.S. Sedghpour, S. Fazliahmadi, T. Fahringer, N. Rasouli, GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds, IEEE Trans. Parallel Distrib. Syst. 31 (6) (2019) 1239–1254.
[20] Y. Saadi, S. Jounaidi, S. El Kafhali, H. Zougagh, Reducing energy footprint in cloud computing: a study on the impact of clustering techniques and scheduling algorithms for scientific workflows, Computing (2023) 1–31.

[21] M. Farid, R. Latip, M. Hussin, N.A.W. Abdul Hamid, A survey on QoS requirements based on particle swarm optimization scheduling techniques for workflow scheduling in cloud computing, Symmetry 12 (4) (2020) 551.
[22] N. Malik, M. Sardaraz, M. Tahir, B. Shah, G. Ali, F. Moreira, Energy-efficient load balancing algorithm for workflow scheduling in cloud data centers using queuing and thresholds, Appl. Sci. 11 (13) (2021) 5849.
[23] H. Topcuoglu, S. Hariri, M.Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[24] D. Gabi, A.S. Ismail, A. Zainal, Z. Zakaria, Quality of service task scheduling algorithm for time-cost trade off scheduling problem in cloud computing environment, Int. J. Intellig. Syst. Tech. Appl. 18 (5) (2019) 448–469.
[25] H. Kumar, A new hybrid particle swarm optimization algorithm for optimal tasks scheduling in distributed computing system, Intellig. Syst. Appl. 18 (2023), 200219.
[26] D. Wang, D. Tan, L. Liu, Particle swarm optimization algorithm: an overview, Soft Computing 22 (2018) 387–408.
[27] A. Verma, S. Kaushal, A hybrid multi-objective particle swarm optimization for scientific workflow scheduling, Parallel Comput. 62 (2017) 1–19.