Computer Standards & Interfaces 97 (2026) 104106

Contents lists available at ScienceDirect

Computer Standards & Interfaces

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

A novel hybrid WOA–GWO algorithm for multi-objective optimization of energy efficiency and reliability in heterogeneous computing

Karishma, Harendra Kumar *

Department of Mathematics and Statistics, Gurukula Kangri (Deemed to be University) Haridwar, Uttarakhand 249404, India

ARTICLE INFO

Keywords:
Energy-efficient scheduling
Heterogeneous computing
Hybrid WOA–GWO
Metaheuristics
Reliability optimization
Sensitivity analysis

ABSTRACT

Heterogeneous computing systems are widely adopted for their capacity to optimize performance and energy efficiency across diverse computational environments. However, most existing task scheduling techniques address either energy reduction or reliability enhancement, rarely achieving both simultaneously. This study proposes a novel hybrid whale optimization algorithm–grey wolf optimizer (WOA–GWO) integrated with dynamic voltage and frequency scaling (DVFS) and an insert-reversed block operation to overcome this dual challenge. The proposed Hybrid WOA–GWO (HWWO) framework enhances task prioritization using the dynamic variant rank heterogeneous earliest-finish-time (DVR-HEFT) approach to ensure efficient processor allocation and reduced computation time. The algorithm's performance was evaluated on real-world constrained optimization problems from CEC 2020, as well as Fast Fourier Transform (FFT) and Gaussian Elimination (GE) applications. Experimental results demonstrate that HWWO achieves substantial gains in both energy efficiency and reliability, reducing total energy consumption by 55% (from 170.52 to 75.67 units) while increasing system reliability from 0.8804 to 0.9785 compared to state-of-the-art methods such as SASS, EnMODE, sCMAgES, and COLSHADE. The experimental results, implemented on varying tasks and processor counts, further demonstrate that the proposed algorithmic approach outperforms existing state-of-the-art and metaheuristic algorithms by delivering superior energy efficiency, maximizing reliability, minimizing computation time, reducing schedule length ratio (SLR), optimizing the communication-to-computation ratio (CCR), enhancing resource utilization, and minimizing sensitivity analysis. These findings confirm that the proposed model effectively bridges the existing research gap by providing a robust, energy-aware, and reliability-optimized scheduling framework for heterogeneous computing environments.

1. Introduction

1.1. Motivation

In recent years, the exponential growth in data volume and computational demands has propelled the development of heterogeneous computing systems. Numerous computing resources are required for various heterogeneous computing models, such as utility computing, peer-to-peer, and grid computing. These resources can be allocated through the network in order to fulfill the needs of carrying out high performing tasks [1]. Resource scheduling is a fundamental challenge in heterogeneous computing, especially as the number of tasks and resources increases. Inefficient task allocation can lead to processor overutilization or underutilization, complicating the scheduling process [2]. Heterogeneous distributed computing has proven highly effective in handling diverse and complex end-user tasks, driven by advancements in network technologies and infrastructure [2,3]. However, as multiprocessor task scheduling is an NP-hard optimization problem, delivering a valid solution within a predefined deadline remains a significant challenge for real-time applications in heterogeneous systems [4]. Alternatively, the delicate balance between performance and power consumption stands as a pivotal factor in the design of multiprocessor systems [5]. To attain optimal performance, it is imperative to implement efficient scheduling of applications across the diverse resources within heterogeneous computing systems, complemented by efficient runtime support mechanisms [6]. In multiprocessor systems, scheduling of tasks involves arranging the sequence of tasks and facilitating their execution across selected processors to achieve a predetermined goal, such as meeting deadlines, minimizing overall execution time (makespan), conserving energy, enhancing system reliability, among other objectives [7].

Efficiently managing energy consumption is pivotal in the design of heterogeneous distributed systems. This is essential as the dissipation of energy directly influences not only the development and

* Corresponding author.
E-mail addresses: maths.karishma97@gmail.com (Karishma), balyan.kumar@gmail.com (H. Kumar).

https://doi.org/10.1016/j.csi.2025.104106
Received 14 February 2025; Received in revised form 14 November 2025; Accepted 28 November 2025
Available online 7 December 2025

operation of the system but also profoundly impacts the individuals within the living environment [8]. The rise in energy consumption has emerged as a significant concern, which has a direct impact on the costs associated with computing services. This consumption typically comprises dynamic energy resulting from switching activities and static energy arising from leakage currents [9]. Recognizing the importance of energy conservation, researchers have explored and developed several techniques to address this issue. These include memory optimization, DVFS, and resource hibernation. DVFS, also known as dynamic speed scaling (DSS), dynamic power scaling (DPS), and dynamic frequency scaling (DFS), is particularly noteworthy for its potential to save energy [10,11]. This technique facilitates energy-efficient scheduling by dynamically adjusting the supply voltage and frequency of a processor while tasks are running, thereby optimizing energy usage [12–14]. The implementation of dynamic voltage scaling for energy-efficient optimization presents a noteworthy advancement. Nevertheless, it is essential to acknowledge a potential drawback: an elevated risk of transient failures in processors, which could undermine the reliability of systems [9,15]. Reliability pertains to the probability that a schedule will successfully complete its execution within the defined parameters [16,17]. Higher frequencies typically correspond to both high energy consumption and enhanced reliability, whereas lower frequencies are associated with decreased energy consumption and reduced reliability [18]. When an application meets its designated reliability objective – referred to as a reliability goal, requirement, assurance, or constraint in various studies – it is deemed reliable in accordance with functional safety standards. These standards include DO-178B for avionics systems, ISO 26262 for automotive systems, and IEC 61508 for a broad spectrum of industrial software systems [8,19].

1.2. Our contributions

Task scheduling, an NP-hard problem, increases the complexity of voltage adjustment choices in heterogeneous computing systems [20,21]. Balancing energy efficiency and reliability presents a major challenge, as prioritizing one often complicates optimizing the other [18]. Scheduling algorithms are broadly classified into heuristics and metaheuristics. Heuristic methods, which use greedy strategies for optimal selection [22], are computationally efficient but often fail to perform well in complex or large-scale scheduling problems [21,23]. In contrast, metaheuristic algorithms, inspired by natural processes, offer more reliable results and greater flexibility [24]. Metaheuristics are popular due to their simplicity, adaptability, independence from derivative-based methods, and ability to avoid local optima. Authors [25] developed an optimized gravitational search algorithm (GSA) to enhance feature-level fusion in multimodal biometric systems. Their work demonstrated how metaheuristic optimization can effectively improve system performance through better parameter tuning and search-space exploration. Authors [26] developed a hybrid white shark optimizer–support vector machine (WSO–SVM) model for gender classification from video data, where the white shark optimizer was used to fine-tune SVM parameters, leading to improved accuracy and faster processing compared to traditional SVM methods. In [27] authors developed two novel task scheduling models based on the metaheuristic GWO technique to optimize energy consumption while minimizing computational time for parallel applications. In this article, a novel hybridization (HWWO) of the WOA [28] and GWO [29] is employed to tackle the task scheduling problem. Hybrid algorithms are designed to integrate the features of various metaheuristic approaches, exploiting their synergy to address complex optimization challenges. This fusion not only enhances the efficiency and flexibility of the algorithms but also augments their overall performance, often surpassing that of traditional metaheuristic algorithms. Furthermore, a wide range of such hybrid algorithms has been developed, driving the evolution of new-generation metaheuristics that effectively balance exploration and exploitation. The proposed HWWO leverages this approach to achieve superior scheduling performance by harnessing the complementary strengths of WOA and GWO while mitigating their individual drawbacks.

The WOA demonstrates superior exploration capabilities through its advanced updating mechanism, employing a randomized search approach to dynamically shift positions and navigate towards optimal solutions. As highlighted by [30], WOA strikes a good balance between exploration and exploitation, exhibiting notable convergence speed in solving optimization problems. However, despite its effectiveness relative to traditional algorithms, WOA can struggle to escape local optima due to its encircling mechanism [31] and may fail to effectively refine the best solutions. Conversely, GWO excels in exploitation through strong local search capabilities but suffers from limited diversity in the early stages, which can hinder global search. To address these limitations, we have proposed a hybrid approach that augments WOA with mutation operators and integrates it with GWO to enhance overall scheduling performance. Recognizing that excessive mutation can disrupt previously discovered good solutions and impede convergence, we have incorporated an insert-reversed block operation after mutation to preserve solution quality and improve the algorithm's efficiency.

This study endeavors to attain an optimized energy-efficient scheduling algorithm with a maximal systems' reliability, thus minimizing the aggregate energy consumption of precedence-constrained tasks in parallel applications executed on heterogeneous computing systems. The primary contributions of this research are succinctly outlined as follows:

- This article proposes innovative hybrid algorithmic approaches that combine the WOA and GWO to tackle the intricate problems related to energy-efficient tasks scheduling.
- This study meticulously designs energy-efficient scheduling algorithms that leverage the hybrid WOA–GWO to effectively optimize two key objectives; energy consumption and the reliability of the system. The algorithms ensure compliance with the deadline constraints of parallel applications.
- The proposed algorithm assigns tasks to suitable processors by synergistically integrating the HWWO technique with DVFS technology. Here, the proposed algorithm applies the mutation operator in conjunction with the insert-reversed block operation as part of the HWWO technique and helps to mitigate the static energy consumption. Additionally, the DVFS technique has been utilized to mitigate the dynamic energy consumption.
- Comprehensive experimental evaluations are carried out by comparing the proposed HWWO algorithm's performance against several well-known algorithms, including the FFT, GE, and four benchmark algorithms from the 'CEC2020 Competition' — SASS, EnMODE, sCMAgES, and COLSHADE. Furthermore, the algorithm is subjected to testing on a set of unimodal benchmark test functions.
- The experimental results demonstrate that the proposed algorithmic approach outperforms existing state-of-the-art and metaheuristic algorithms by delivering superior energy efficiency, maximizing reliability, minimizing computation time for tasks assignment, minimizing sensitivity analysis and SLR, CCR, and enhancing resource utilization. These promising results hold true across various scale conditions and deadline constraints.
- Conduct an evaluation of the proposed technique's computational complexity during execution and employ the Wilcoxon-signed rank statistical test to validate its performance.

The structure of the article is outlined as follows. Section 2 offers an extensive review of relevant literature and related works. Detailed explanations of the WOA and GWO techniques are covered in Section 3. In Section 4, pertinent models and problem formulations are explored, along with essential notations used throughout the study. Section 5 provides a thorough description of the proposed model. The development of simulation experiments to evaluate the proposed model is detailed in Section 6. Finally, Section 7 elaborates on the study's conclusion, discussing limitations and directions for future research.

2. Literature review

The substantial energy consumption associated with computing systems poses a significant impediment to their rapid advancement. Therefore, minimizing energy usage while ensuring system reliability has become a pressing concern for fostering sustainable computing methodologies. Researchers from multiple fields have devoted substantial efforts to investigating the intricate challenges associated with task scheduling techniques that strike a balance between minimizing energy consumption and maximizing system reliability.

In the realm of sustainable computing systems, DVFS technique has emerged as a prominent and widely employed method for efficiently curtailing energy consumption [32]. A study by [33] addressed energy-aware task assignment for deadline-constrained workflow applications in heterogeneous computing environments. Earlier, [34] explored theoretical models for DVFS and proposed an energy-aware scheduling strategy for single-processor platforms. Advanced algorithms like enhanced energy-efficient scheduling (EES) [35] and DVFS-enabled energy-efficient workflow task scheduling (DEWTS) [36] were later developed to reduce energy consumption in parallel applications. EES uses DVFS to slack non-critical tasks while meeting time constraints, while DEWTS enhances energy efficiency by selectively turning off processors to minimize static energy consumption. In [37], authors proposed the downward energy consumption minimization (DECM) algorithm that innovatively transfers application deadlines to task-level deadlines using deadline-slack and task level concepts, enabling low-complexity energy minimization. Authors [38] proposed a two-stage solution to enhance the reliability of automotive applications while satisfying energy and response time constraints. First, it solved response time reduction under energy constraint (RREC) via average energy pre-allocation. While, the second stage enhanced reliability within the remaining energy–time budgets from the RREC stage. Addressing the challenge of energy-efficient tasks scheduling in cloud environments, the authors of [39] proposed an algorithm based on DVFS that prioritizes tasks by deadline, categorizes physical machines, and assigns tasks to nearby machines in the same priority class. Researchers introduced the energy makespan multi-objective optimization algorithm for energy-efficient, low-latency workflow scheduling across fog–cloud resources [40]. The research work of [41,42] aimed to devise an approach that could reduce the overall execution time for parallel applications running on high-performance distributed computing environment, while concurrently enforcing adherence to predetermined energy consumption thresholds.

Reliability-aware design algorithms aimed at ensuring reliability typically try to reduce certain objectives while also satisfying reliability requirements. Improving the reliability of parallel applications frequently results in longer schedules or higher energy usage. Optimizing both schedule length (or energy consumption) and reliability concurrently poses a classic bi-criteria optimization problem requiring the identification of pareto-optimal solutions [43,44]. The researchers in [45] introduced an approach to reliably assign and schedule tasks on heterogeneous multiprocessor systems, tackling the complexities and potential failures associated with critical applications. Researchers in [46,47] tackled the intricate problem of workflow scheduling on heterogeneous computing systems, aiming to achieve high reliability while minimizing the unnecessary duplication of resources. Energy consumption and reliability are closely intertwined concepts. In [48], researchers explored this relationship and developed a model that linked energy consumption to reliability levels. Their work aimed to maximize the reliability of parallel applications executed on uniprocessor systems while adhering to strict deadlines and energy consumption constraints. Authors in [49], proposed power management schemes for homogeneous multiprocessors that targeted energy savings while upholding specified system reliability levels.

Metaheuristic techniques are versatile methods inspired by natural phenomena, such as evolutionary adaptation, biological swarms, and physical systems. Among these, swarm intelligence (SI) methods form a prominent category, drawing on the collective behavior of living organisms. These algorithms utilize population-based, stochastic, and iterative strategies. Numerous real-world challenges, including drone deployment, image processing, wireless sensor network localization, machine learning optimization, and others, have found effective solutions by employing SI techniques [2,50]. Beyond their diverse applications, SI algorithms have undergone continuous enhancements through modifications, hybridizations with other techniques [51], and parallel computing implementations [52]. These efforts aim to further optimize their performance and obtain superior solutions across diverse problem domains. SI algorithms are inspired by the collaborative behaviors observed in various animal and insect communities, where entities interact and respond to their environment collectively. This includes animal herds, ant colonies, fish schools, bacterial aggregations, and bird flocks etc. These algorithms exhibit notable advantages including adaptability, user-friendliness, and reliability [53]. Some of the recently developed SI algorithms are artificial bee colony (ABC) [54], ant colony optimization (ACO) [55], cuckoo optimization algorithm (COA) [56], particle swarm optimization (PSO) [57], horse herd optimization algorithm [58], krill herd (KH) [59], crow search algorithm (CSA) [60], GWO [29], sailfish optimizer (SFO) [61], and WOA [28] etc.

This study implements the WOA algorithm to address energy-efficient and reliable task scheduling in heterogeneous computing environments. The approach enhances WOA by hybridizing it with GWO, leveraging the strengths of both techniques. WOA, introduced by Mirjalili and Lewis in 2016 [28], is a notable method in swarm intelligence optimization. Inspired by the remarkable hunting strategies employed by humpback whales in the vast ocean, this algorithm demonstrated competitive or superior performance compared to several existing optimization methods [28]. Similar to WOA, GWO is a nature-inspired metaheuristic optimization algorithm based on the social behavior and hunting strategies of grey wolves [29]. It is widely used for solving complex optimization problems. WOA, recognized for its unique approach, has been effectively applied to scheduling tasks in cloud computing, aiming to enhance system performance within constrained computing capacities [62]. The researchers proposed an innovative scheduling approach based on WOA that combined multi-objective optimization with trust awareness. It mapped tasks to virtual resources based on priorities, evaluated trust via SLA parameters, and enforced deadline constraints for task execution on VMs [63]. The study tackled the challenge of scheduling tasks on heterogeneous multiprocessor systems equipped with DVFS capabilities. The objective was to optimize energy consumption while adhering to constraints related to makespan and system reliability. To achieve this, [64] proposed an enhanced variant of the WOA, incorporating opposition-based learning and an individual selection strategy. In the article [65], the authors introduced an improved whale algorithm (IWA) to optimize tasks allocation in multiprocessing systems (MPS), minimizing energy consumption and makespan. They utilized DVFS and addressed task scheduling's NP-hard nature. The article [66] addressed power consumption in cloud infrastructure, underscoring the necessity for energy-efficient algorithms and load balancing techniques. The authors employed various optimization algorithms, such as PSO, COA, and WOA, to achieve efficient resource scheduling and mitigate energy consumption.

The researchers proposed an innovative tasks scheduling algorithm leveraging the GWO technique for cloud computing environment [67]. This GWO-based tasks scheduling (GWOTS) approach aimed to minimize execution costs, reduce energy consumption, and shorten the overall makespan . Researchers developed an advanced multi-objective optimization technique inspired by the GWO to address the growing computational demands on cloud data centers [68]. Their primary goals were to maximize the efficient utilization of cloud resources, minimize energy consumption, and reduce the overall execution time, while

Table 1
Performance parameters of various WOA based metaheuristic tasks scheduling techniques.

S. No.	Reference	Core technique	Environment	Considered parameters					
				Energy	Reliability	CT_{TA}	Sensitivity	Makespan	Resource utilization
1	[72]	VWOA	Heterogeneous	✓	✗	✗	✗	✓	✓
2	[73]	WOA	Heterogeneous	✗	✗	✓	✗	✗	✓
3	[74]	M-WODE	Heterogeneous	✗	✗	✗	✗	✓	✗
4	[75]	WOA	Heterogeneous	✗	✗	✗	✗	✓	✗
5	[76]	h-DEWOA	Heterogeneous	✓	✗	✗	✗	✓	✗
6	[77]	IWC	Homogeneous	✗	✗	✓	✗	✗	✗
7	[24]	HGWWO	Heterogeneous	✓	✗	✗	✗	✓	✓
8	[78]	HWOA based MBA	Homogeneous	✗	✗	✓	✗	✓	✓
9	[79]	HPSWOA	Heterogeneous	✗	✗	✓	✗	✗	✓
10	[80]	CWOA	Heterogeneous	✓	✗	✗	✗	✓	✓
11	[81]	HWOA	Heterogeneous	✗	✗	✓	✗	✗	✗
12	[82]	WHOA	Heterogeneous	✓	✗	✓	✗	✓	✓
13	[83]	ANN-WOA	Homogeneous	✗	✗	✓	✗	✗	✓
14	[84]	WOA	Heterogeneous	✓	✗	✓	✗	✓	✗
15	Proposed model	HWWO	Heterogeneous	✓	✓	✓	✓	✓	✓

ensuring the requested services were delivered effectively. In [69], authors proposed a novel hybrid model that combined PSO and GWO for workflow scheduling in cloud computing environments. This integrated approach aimed to enhance the overall performance by optimizing total execution costs and reducing the time required for task completion . The article [70] addressed the challenges of tasks allocation and quality of service (QoS) optimization in cloud–fog computing environment. The authors proposed a multi-objectives GWO algorithm, implemented within the fog broker system, to minimize delay and energy consumption. In article [71], authors addressed tasks scheduling challenges in cloud computing by proposing a GWO-based algorithm. The approach aimed to efficiently allocate resources and minimize task completion times.

WOA employs simple yet powerful search mechanisms to efficiently identify optimal solutions. However, like other SI algorithms, WOA can face challenges such as getting trapped in local optima and maintaining population diversity. To address these limitations, numerous WOA variants have been proposed, enhancing the core algorithm through modifications or hybridization. These improved versions have been successfully applied to a variety of optimization problems, including task scheduling in distributed computing environment.

Table 1 provides a concise summary of recent studies that utilize WOA-based metaheuristic techniques for task scheduling in distributed systems. The table categorizes these approaches based on key performance parameters, such as energy consumption, reliability, CT_{TA}, sensitivity, makespan, and resource utilization. These parameters are chosen for their relevance in evaluating the proposed HWWO method in this article. As evident from Table 1, while many studies consider these parameters individually or in limited combinations, none evaluate them as comprehensively as we have done in this work. This underscores the uniqueness of our approach and its potential to provide a more well-rounded assessment of task scheduling optimization.

3. Preliminaries

The following discussion aims to provide a succinct yet comprehensive understanding of the core concepts driving the WOA and GWO metaheuristics. Additionally, it shall elucidate the distinctive features of these algorithms and their versatile applicability across a wide spectrum of problem spaces.

3.1. Whale optimization algorithm

Whales are majestic creatures that captivate the imagination. Among the animal kingdom, they hold the distinction of being the largest mammals on earth. An adult whale can reach staggering proportions, ranging from 30 m in length and weighting up to 180 tons. These giants of the sea are classified into various species, including the killer whale, the sei, the finback, the humpback, the minke, and the awe-inspiring blue whale. Beyond their sheer size, whales are remarkable for their intelligence and emotional depth, exhibiting complex social behaviors. They are often observed traveling and living in close-knit groups. One of the most fascinating species is the humpback whale, renowned for its intricate hunting technique known as bubble-net feeding [85]. When it comes to hunting, humpback whales exhibit a remarkable preference for preying on schools of krill or small fish that congregate near the ocean's surface. Their intricate hunting strategy involves diving to depths of around 12 m and then employing a fascinating maneuver. The whales release a spiral of bubbles, carefully encircling their prey, and then gracefully swim upwards towards the surface, trapping their quarry within the bubble net. It is noteworthy that this bubble-net feeding technique is a unique behavior that has been observed exclusively in humpback whales. The exceptional hunting prowess of these whales has served as a source of inspiration for the development of a swarm intelligence algorithm known as the WOA [28]. Proposed for solving continuous optimization problems, the WOA aims to mimic the humpback whales' remarkable hunting strategies. The WOA represents each potential solution as a whale searching for the optimal position, guided by the best solution found. It uses two mechanisms: encircling the prey (exploring promising areas) and creating bubble nets (exploiting by trapping targets). As depicted in Fig. 1, humpback whales exhibit a remarkable coordinated feeding strategy involving the creation of bubble nets to trap and capture their prey. The exploration phase searches for potential solution regions, while exploitation focuses on the most viable solutions within those areas, balancing exploration and exploitation for efficient optimization

3.1.1. A mathematical-based model

This section initially models the whale behaviors of encircling targets, prey searching, and spiral bubble-net feeding maneuvers mathematically.

3.1.1.1. Encircling prey. The WOA algorithm takes inspiration from the hunting behavior of humpback whales, which can effectively locate and encircle prey. Since the optimal solution's position is not known initially, the algorithm assumes the current best solution is near the global optimum. The other candidate solutions then attempt to update their positions towards this best solution identified so far. This encircling and

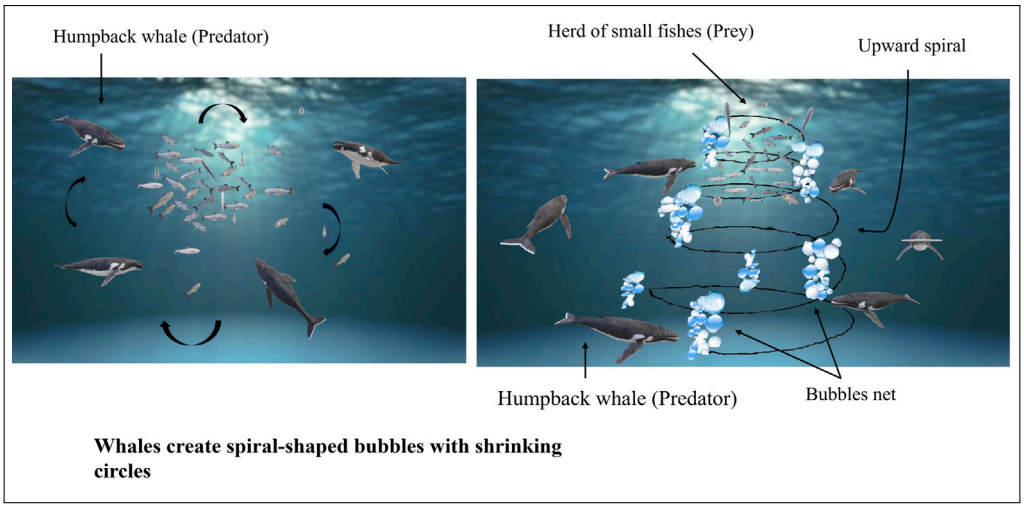

Fig. 1. Coordinated feeding strategy employed by humpback whales involving bubble nets.

localization process is mathematically modeled through Eqs. (1) and (2).

$$\vec{D} = |\vec{\xi} * \vec{Z}^*(r) - \vec{Z}(r)| \tag{1}$$

$$\vec{Z}(r+1) = |\vec{Z}^*(r) - \vec{\zeta} * \vec{D}| \tag{2}$$

The Eqs. (1) and (2) involve several variables and vectors. The variable r denotes the current iteration number. $\vec{\zeta}$ and $\vec{\xi}$ represent coefficient vector quantities. The vector $\vec{Z}^*$ representing the current best solution must be updated during any iteration where a superior solution is identified. The vector $\vec{Z}$ indicates another position vector being considered. The absolute value operation is represented by the || symbol. The calculation of the coefficient vectors $\vec{\zeta}$ and $\vec{\xi}$ proceeds in the following manner [85]:

$$\vec{\zeta} = 2 * \vec{\mu} * \vec{l}_1 - \vec{\mu} \tag{3}$$

$$\vec{\xi} = 2 * \vec{l}_2 \tag{4}$$

In Eq. (3), the parameter $\vec{\mu}$ linearly decreases from 2 to 0 across all iterations, encompassing the exploration and exploitation phases, while l_1, l_2 are random vectors in the range [0, 1]. The formulation of $\vec{\mu}$ can be expressed as [2]:

$$\vec{\mu} = 2 - r * \left\{ \frac{2}{max\_iteration} \right\} \tag{5}$$

3.1.1.2. Exploitation phase (bubble-net attacking method). Two distinct methodologies have been proposed aimed at constructing mathematical models to characterize the bubble-net feeding behavior exhibited by humpback whales.

i **Shrinking encircling mechanism:** This behavior is modeled by diminishing the convergence parameter $\vec{\mu}$ in Eq. (3). Moreover, the oscillation range of $\vec{\zeta}$ contracts linearly via $\vec{\mu}$, transitioning from 2 to 0 across iterations. Stated differently, $\vec{\zeta}$ represents a random value within the interval $[-\mu, \mu]$.

ii **Spiral updating position mechanism:** The approach commences by quantifying the distance between the vector $\vec{Z}^*$, representing the best solution identified thus far, and another whale position vector $\vec{Z}$, through Eq. (6). Subsequently, it defines the spiral motion pattern originating from the present location and progressing towards an enhanced solution, as expressed through Eq. (7). In these equations, the constant b governs the logarithmic spiral's geometric characteristics, while v signifies a randomly generated scalar quantity constrained within the bounds of [−1, 1].

$$\vec{D'} = |\vec{Z}^*(r) - \vec{Z}(r)| \tag{6}$$

$$\vec{Z}(r+1) = \vec{D'} * e^{bv} * cos(2\pi v) + \vec{Z}^*(r) \tag{7}$$

The humpback whale circles its prey in a tightening spiral pattern while hunting. The WOA models this by randomly choosing between the shrinking encirclement or spiral model, each with 50% probability, to update whale positions during optimization. This stochastic positional update process is given by Eq. (8), wherein p denotes a randomly generated scalar confined within the range [0,1].

$$\vec{Z}(r+1) = \begin{cases} \vec{Z}^*(r) - \vec{\zeta} * \vec{D} & \text{if } p < 0.5 \\ D' * e^{bv} * cos(2\pi v) + \vec{Z}^*(r) & \text{if } p \geq 0.5 \end{cases} \tag{8}$$

3.1.1.3. Exploration phase (searching for prey strategy). This strategy facilitates the whales in surveying the problem domain to uncover unexplored regions and augment the diversity within the population. A randomly selected search agent dictates the positional update for each individual whale. The parameter $\vec{\zeta}$ enables steering the search agent away from an arbitrarily chosen humpback whale. The exploration phase, governed by Eq. (10), prevents the premature convergence to local optima [28].

$$\vec{D} = |\vec{\xi} * \vec{Z}_{rand} - \vec{Z}(r)| \tag{9}$$

$$\vec{Z}(r+1) = \vec{Z}_{rand} - \zeta * \vec{D} \tag{10}$$

where, $\vec{Z}_{rand}$ denotes a randomly selected whale position vector from the current population within the search space.

3.2. Grey wolf optimization algorithm

The GWO is a SI technique that emulates the hierarchical leadership structure and cooperative hunting strategies exhibited by grey wolves in their natural habitats [29]. The GWO algorithm mathematically formalizes the search, encirclement, and attack behaviors observed in the predatory conduct of grey wolves. It incorporates the hierarchical social structure present within wolf packs as a core concept. Wolves are classified into four distinct hierarchical tiers based on their levels of dominance. The α, β, and δ ranks represent the leaders, presumed to possess superior capabilities that guide the pack. In contrast, the ω

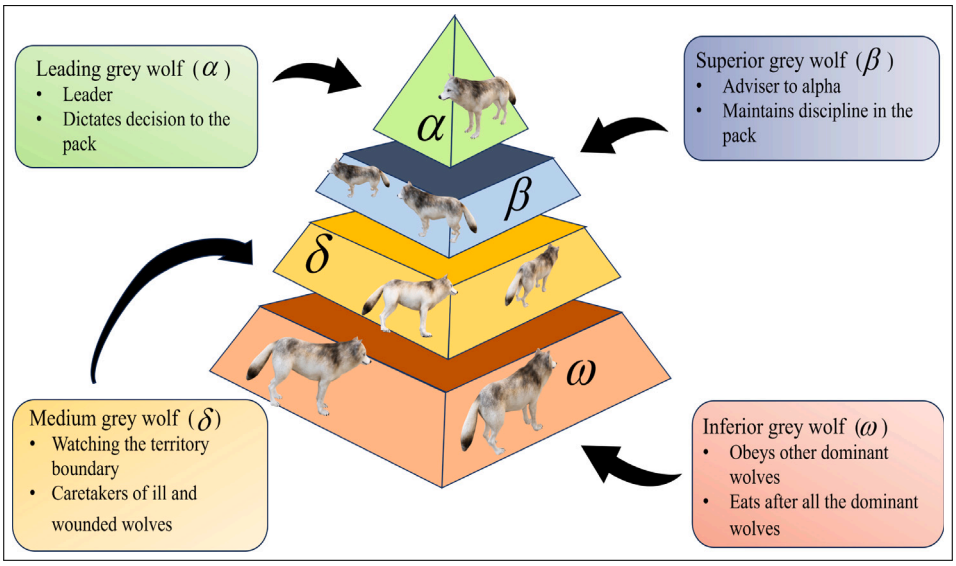

Fig. 2. Organizational structure and role hierarchy in grey wolf packs.

wolves assume a subordinate role, following the navigation directives provided by the dominant leaders (see Fig. 2).

GWO employs mathematical models that emulate the intricate hunting tactics exhibited by grey wolves, including their pursuit, encirclement, and eventual capture of prey, as a framework to guide the optimization process.

3.2.1. Social behavior

The mathematical formulation supposes α to be the preeminent solution, embodying the social behavior of the lead wolf. The subsequent solutions, β and δ, constitute the second and third-best outcomes, respectively. All remaining solutions are collectively classified as ω. The hunting process within the GWO algorithm is steered by the triumvirate of α, β, and δ, while the ω solutions are governed by adherence to this leading trio.

Encircling the prey: The predatory strategy of grey wolves involves meticulously encircling and confining their prey during the hunt. This critical stage of encirclement is mathematically modeled by the ensuing system of equations:

$$\vec{Y} = |\vec{\xi} * \vec{Z}_p(r) - \vec{Z}(r)| \tag{11}$$

$$\vec{Z}(r+1) = \vec{Z}_p(r) - \vec{\zeta} * \vec{Y} \tag{12}$$

In the given context, the variable $\vec{Y}$ symbolizes the vector distance separating the prey's location from the wolf's position. The variable r denotes the current iteration number, while $(r + 1)$ signifies the iteration number that follows. The variable $\vec{Z}_p(r)$ signifies the position of the prey within the optimization process, whereas $\vec{Z}(r)$ denotes the position of the wolf. These variables are employed to model the interaction between the prey and the wolf, which is a crucial aspect of the optimization algorithm being discussed.

The optimization algorithm iteratively calculates and refines the coefficient vectors $\vec{\zeta}$ and $\vec{\xi}$ through the use of the mathematical expressions represented by Eqs. (3) and (4), which are provided as follows:

$$\vec{\zeta} = 2 * \vec{\mu} * \vec{l}_1 - \vec{\mu} \text{ and } \vec{\xi} = 2 * \vec{l}_2$$

where, the parameter $\vec{\mu}$ linearly decreases from 2 to 0 across all iterations, encompassing the exploration and exploitation phases, while l_1, l_2 are random vectors in the range [0, 1].

3.2.2. Hunting

The grey wolf algorithm mimics the intricate hunting strategies employed by wolf packs in nature. Central to this optimization process is the α wolf, acting as the lead entity guiding the search for the optimal solution. Through iterative refinement, the α continuously updates and stores the best solution encountered, replacing it with an improved one if found in subsequent iterations. This iterative refinement allows convergence towards the optimal result. While the α takes the lead, the β and δ wolves contribute their prowess to the hunt. By mathematically simulating the hunting behavior, the algorithm assumes that the α, β, and δ solutions possess superior knowledge of the potential optimal position. Consequently, the top three candidate solutions are retained. The remaining search agents, including ω, must update their positions based on the α location. In essence, the α, β, and δ predict the optimal location, while other wolves randomly explore the surrounding areas, driven by the overarching goal of locating the prey — the global optimum. The positions of the wolves are iteratively updated through the subsequent mathematical equations that emulate the hunting behavior.

$$\left.\begin{aligned} \vec{Y}_\alpha &= |\vec{\xi}_1 * \vec{Z}_\alpha - \vec{Z}| \\ \vec{Y}_\beta &= |\vec{\xi}_2 * \vec{Z}_\beta - \vec{Z}| \\ \vec{Y}_\delta &= |\vec{\xi}_3 * \vec{Z}_\delta - \vec{Z}| \end{aligned}\right\} \tag{13}$$

$$\vec{Z}_1 = \vec{Z}_\alpha - \vec{\zeta}_1 * \vec{Y}_\alpha \tag{14}$$

$$\vec{Z}_2 = \vec{Z}_\beta - \vec{\zeta}_2 * \vec{Y}_\beta \tag{15}$$

$$\vec{Z}_3 = \vec{Z}_\delta - \vec{\zeta}_3 * \vec{Y}_\delta \tag{16}$$

$$\vec{Z}(r+1) = \frac{\vec{Z}_1 + \vec{Z}_2 + \vec{Z}_3}{3} \tag{17}$$

3.2.3. Exploitation (attacking prey)

The hunting process involves a strategy employed by the grey wolves to restrict the prey's mobility, rendering it vulnerable to an attack. This approach is implemented by gradually decreasing the value of a parameter $\vec{\mu}$(decrease from 2 to 0). Concurrently, the value of another parameter, $\vec{\zeta}$, is also reduced in accordance with the value of $\vec{\mu}$, ensuring it remains within the range of [−1, 1]. The grey wolves initiate an attack on the prey when the value of $\vec{\zeta}$ falls between −1 and 1.

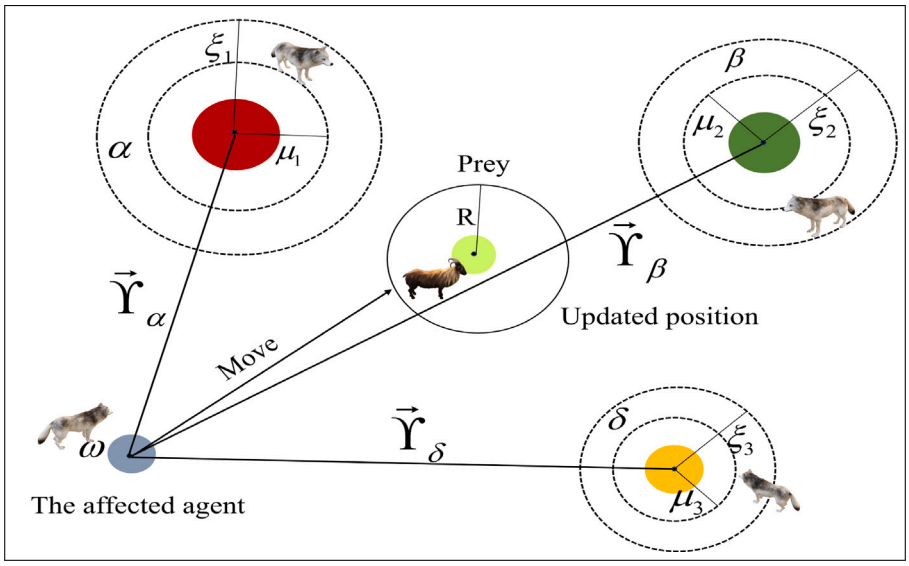

Fig. 3. Dynamic positioning of search agents based on parameter interactions.

3.2.4. Exploration (search for prey)

The lead wolves, α, β, and δ, strategically position themselves in a manner that balances the pursuit of the prey with the readiness to strike. This dual approach is modeled through a parameter $\vec{\zeta}$, where values exceeding 1 represent a diversion from the prey's immediate vicinity, yet still within striking range. Another influential factor governing the exploration process is denoted by $\vec{\xi}$, which plays a crucial role, particularly in scenarios where the algorithm encounters local optima. The range of $\vec{\xi}$ lies between 0 and 2, and its value is determined through a mathematical expression, labeled as Eq. (4).

Fig. 3 illustrates a search agent dynamically positioning itself within a search space using α, β, and δ parameters. The agent's final position, representing an estimated prey location, is depicted within a circle defined by these parameters. Surrounding agents adapt their positions around this estimated location, introducing randomness and coordinated behavior akin to predators.

4. Notations and mathematical modeling

4.1. Notations

The key symbols and their meanings, as employed in the present work, are summarized in Table 2.

Table 2
Key symbols for the present work.

Notation	Description
G	Direct acyclic graph (DAG) representing the distributed parallel application
$X = \{\tau_1, \tau_2, \ldots, \tau_{\|X\|}\}$	Set of $\|X\|$ tasks
$Y = \{Y_1, Y_2, \ldots, Y_{\|Y\|}\}$	Set of $\|Y\|$ processors
$\hat{c}_{i,k}$	Worst-case response time between the tasks τ_1 and τ_k
$\hat{w}_{i,l}$	Worst-case execution time of the task τ_i on the processor Y_l
LB(G)	Lower bound of G
DL(G)	Deadline of application G
MS(G)	Makespan of application G
$\hat{E}_s(G)$	Static energy consumption of G
$\hat{E}_d(\tau_i, Y_l, f_{l,h})$	The dynamic energy consumption of taskτ_i on processor Y_l with frequency $f_{l,h}$
$\hat{E}_d(G)$	Dynamic energy consumption of G
$\hat{E}_{total}(G)$	The aggregate energy consumption of G
$R_e(G)$	Reliability of the application G
$R_e(\tau_i, Y_l, f_{l,h})$	Reliability of task τ_i executed on the Y_l with frequency $f_{l,h}$
$R_{e(min)}(G)$	Minimum reliability value of G
$R_{e(max)}(G)$	Maximum reliability value of G
$R_{e(goal)}(G)$	Reliability goal of the application G
$\lambda_{l,h}$	Failure rate of processor Y_l at frequency $f_{l,h}$
$R_{e(min)}(\tau_i)$	Minimum reliability value of τ_i
$R_{e(max)}(\tau_i)$	Maximum reliability value of τ_i
$R_e(\tau_i)$	Reliability of task τ_i
$EST(\tau_k, Y_l)$	Earliest start time of kth task on processor Y_l
$EFT(\tau_k, Y_l)$	Earliest finish time of kth task on processor Y_l
$AFT(\tau_k)$	Actual finish time of kth task

4.2. Application model

The directed acyclic graph (DAG) serves as a versatile representation widely adopted in academic research for modeling distributed parallel applications. In this study, the application is effectively modeled as a DAG, denoted as $G = (X, \hat{E}, \hat{W}, \hat{C})$. This model encompasses a set X, comprising various computational tasks with distinct worst-case execution times ($\hat{W}CETs$) on different processors. Furthermore, the model incorporates a set $\hat{E}$, representing communication edges between these tasks. Each element in $\hat{E}$, represented as $\hat{e}_{i,k}$, signifies a communication link from task τ_i to τ_k, accompanied by a precedence constraint that mandates task τ_k to commence only upon the completion of task τ_i. Consequently, every $\hat{c}_{i,k}$ represents the worst-case response time ($\hat{W}CRT$) of $\hat{e}_{i,k}$. The set $succ(\tau_i)$ represents the immediate successor tasks of τ_i, and $pred(\tau_i)$ represents the immediate predecessor tasks of τ_i. Tasks lacking predecessors are designated as τ_{entry}, while those lacking successors are designated as τ_{exit}. In instances where a function includes multiple τ_{entry} or τ_{exit} tasks, dummy tasks with zero-weight dependencies are introduced into the graph to maintain consistency [8]. For $|Y|$ no. of processors, $\hat{w}_{i,l} \in \hat{W}_{|X|\times|Y|}$ gives the $\hat{W}CET$ of task τ_i on processor Y_l.

The expressions LB(G) and DL(G) represent the lower bound and deadline of G, respectively. It is imperative to maintain a condition wherein the lower bound of an application remains below its associated deadline, i.e., $LB(G) \leq DL(G)$ In the course of this work, the concept of lower bound pertains to the minimal achievable makespan by an application developed through a conventional scheduling algorithm, where each processor is singularly dedicated to the application, operating at its maximum frequency [36,86]. $MS(G)$ denotes the actual makespan achieved by application G, which signifies the precise conclusion time of τ_{exit} within the corresponding DAG.

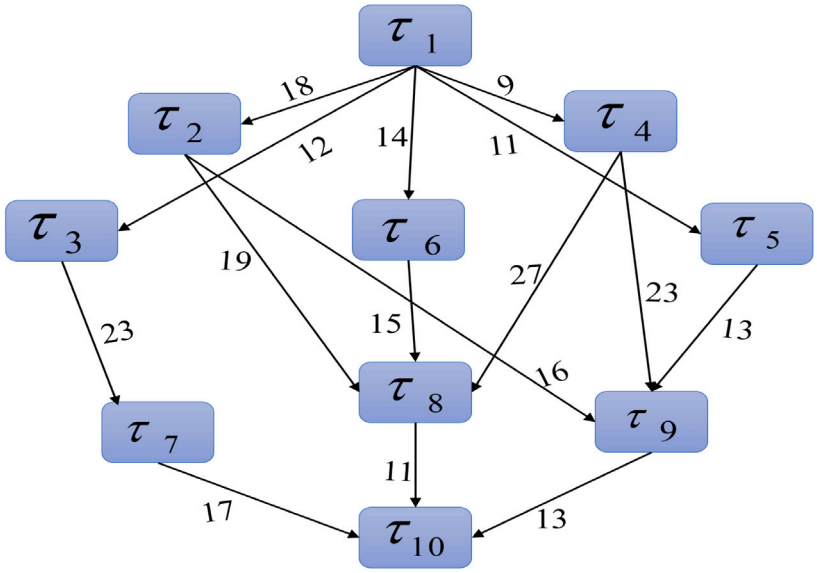

Fig. 4. Example of a DAG featuring 10 tasks.

Table 3
($\hat{W}CETs$) of tasks in Fig. 4.

Tasks \Processors	Y_1	Y_2	Y_3
τ_1	14	16	9
τ_2	13	19	18
τ_3	11	13	19
τ_4	13	8	17
τ_5	12	13	10
τ_6	13	16	9
τ_7	7	15	11
τ_8	5	11	14
τ_9	18	12	20
τ_{10}	21	7	16

Fig. 4 illustrates a DAG-based parallel application as an example [8,86]. This demonstration comprises 10 tasks processed across three designated processors $\{Y_1, Y_2, Y_3\}$. In the illustration, the weight of 18 on the edge $\hat{e}_{1,2}$ connecting τ_1 and τ_2 symbolizes the response time, denoted as $\hat{c}_{1,2}$, if τ_1 and τ_2 are not allocated to the same processor.

The data in Table 3 represents the worst-case execution times ($\hat{W}CETs$) corresponding to the maximum frequency illustrated in Fig. 4. The value of 14 assigned to the intersection of τ_1 and Y_1 in Table 3 denotes the ($\hat{W}CETs$), symbolized as $\hat{w}_{1,1} = 14$. The variations in ($\hat{W}CETs$) for an identical task across different processors arise from the intrinsic diversity of the processors.

4.3. Power and energy models

Considering the nearly linear correlation between voltage and frequency, DVFS techniques are employed to scale down these parameters, thereby achieving energy conservation. Consistent with the approaches adopted in [8,87], the term frequency change is utilized to denote the simultaneous alteration of both voltage and frequency. For DVFS-capable systems, a widely adopted system-level power model, as exemplified in [8,87], is leveraged. This model expresses the power consumption at a given frequency (f) as follows:

$$P(f) = P_s + \phi(P_{ind} + P_d) = P_s + \phi(P_{ind} + C_{ef} f^m) \tag{18}$$

Within this power model, P_s symbolizes the static power component, which can be mitigated solely by deactivating the entire system. The frequency-independent dynamic power is represented by P_{ind}, and this component can be eliminated by transitioning the system into a low-power sleep state. P_d denotes the dynamic power component that exhibits a dependency on frequency. The parameter ϕ signifies the system states and serves as an indicator of whether dynamic power is presently being consumed within the system, where $\phi = 1$ signifies an active state, and $\phi = 0$ represents an inactive condition. The term C_{ef} symbolizes the effective switching capacitance. The exponent m, known as the dynamic power exponent with a minimum value of 2. Both C_{ef} and m are processor-specific constants.

Strategically reducing the operating frequency presents an avenue to curb frequency-dependent power dissipation. However, prolonged execution times may ensue, augmenting static power consumption and frequency-independent power expenditure. Several studies, including those conducted by [8,87], and [88], have established the existence of an optimal energy-efficient frequency, denoted as f_{ee}, at which the system achieves minimal power consumption. This optimal frequency can be formulated as follows:

$$f_{ee} = m\sqrt{\frac{P_{ind}}{(m-1)C_{ef}}} \tag{19}$$

Under the premise that a processor's operating frequency can vary between a minimum available frequency, f_{min}, and a maximum frequency, f_{max}, the optimal energy-efficient frequency for executing a given task should adhere to the following formulation:

$$f_{low} = max(f_{ee}, f_{min}) \tag{20}$$

As a result, any of the actual effective frequencies, denoted as f_h, should reside within the range delineated by $f_{low} \le f_h \le f_{max}$ [8].

For a system with $|Y|$ heterogeneous processors, each processor requires individual power parameters. Here, the static power set is defined as:

$$\{P_{1,s}, P_{2,s}, \ldots, P_{|Y|,s}\}$$

frequency-independent and dependent dynamic power sets are represented as:

$$\{P_{1,ind}, P_{2,ind}, \ldots, P_{|Y|,ind}\} \text{ and } \{P_{1,d}, P_{2,d}, \ldots, P_{|Y|,d}\}$$

the effective switching capacitance is defined as:

$$\{C_{1,ef}, C_{2,ef}, \ldots, C_{|Y|,ef}\}$$

lowest energy-efficient frequency set is represented as:

$$\{f_{1,low}, f_{2,low}, \ldots, f_{|Y|,low}\}$$

and actual effective frequency set is:

$$\{\{f_{1,low}, f_{1,c}, f_{1,d}, \ldots, f_{1,max}\}, \{f_{2,low}, f_{2,c}, f_{2,d}, \ldots, f_{2,max}\}, \ldots \{f_{|Y|,low}, f_{|Y|,c}, f_{|Y|,d}, \ldots, f_{|Y|,max}\}\}$$

Let $\hat{E}_s(G)$ denote the static energy consumed by active processors executing application G. Since inactive processors do not consume energy, $\hat{E}_s(G)$ is the sum of static energy consumption across all active processors, calculated as:

$$\hat{E}_s(G) = \sum_{l=1, Y_l \text{ is on}}^{|Y|} \left(P_{l,s} * MS(G)\right) \tag{21}$$

The dynamic power consumption of task τ_i executing on Y_l at frequency $f_{l,h}$ is represented by $P_d(\tau_i, Y_l, f_{l,h})$. This can be formulated as:

$$P_d(\tau_i, Y_l, f_{l,h}) = (P_{l,ind} + C_{l,ef} * f_{l,h}^{m_l}) \tag{22}$$

And the dynamic energy consumption of τ_i is calculated as:

$$\hat{E}_d(\tau_i, Y_l, f_{l,h}) = (P_{l,ind} + C_{l,ef} * f_{l,h}^{m_l}) * \frac{f_{l,max}}{f_{l,h}} * \hat{w}_{i,l} \tag{23}$$

Let $\hat{E}_d(G)$ represent the total dynamic energy consumption of application G, calculated as the sum of dynamic energy consumed by each task.

$$\hat{E}_d(G) = \sum_{i=1}^{|X|} \hat{E}_d(\tau_i, Y_{ac(i)}, f_{ac(i),hz(i)}) \tag{24}$$

where $Y_{ac(i)}$ signifies the processor and $f_{ac(i),hz(i)}$ represents the frequency at which task τ_i is actively executing.

Hence, the aggregated energy consumption of G can be deduced using the ensuing formulation:

$$\hat{E}_{total}(G) = \hat{E}_s(G) + \hat{E}_d(G) \tag{25}$$

4.4. Reliability model and reliability goal

The concept of processor reliability probability adhering to a Poisson distribution has been extensively studied and widely accepted within the relevant literature [8]. The variable λ_l denotes the failure rate per unit time of processor Y_l and the reliability of a task τ_i executed on processor Y_l within its $\hat{W}CET$ can be quantified using the following mathematical expression:

$$R_e(\tau_i, Y_l) = e^{-\lambda_l * \hat{w}_{i,l}} \tag{26}$$

For DVFS-enabled processors, research shows varying failure rates across frequencies. Let $\lambda_{l,max}$ denote the failure rate of processor Y_l at maximum frequency. Then, the failure rate $\lambda_{l,h}$ of Y_l at frequency $f_{l,h}$ is calculated as:

$$\lambda_{l,h} = \lambda_{l,max} * 10^{\frac{d(f_{l,max} - f_{l,h})}{f_{l,max} - f_{l,min}}} \tag{27}$$

Where the constant d indicates the sensitivity of failure rates to voltage scaling.

Subsequently, a correlation is established between task reliability and frequency, as outlined by Eqs. (26) and (27) i.e., the reliability of the task τ_i executed on the processor Y_l with the frequency $f_{l,h}$ is calculated as follows:

$$\begin{aligned} R_e(\tau_i, Y_l, f_{l,h}) &= e^{-\lambda_{l,h} * \frac{\hat{w}_{i,l} * f_{l,max}}{f_{l,h}}} \\ &= e^{-\lambda_{l,max} * 10^{\frac{d(f_{l,max} - f_{l,h})}{f_{l,max} - f_{l,min}}} * \frac{\hat{w}_{i,l} * f_{l,max}}{f_{l,h}}} \end{aligned} \tag{28}$$

The overall reliability of an application can be expressed as the product of the reliability values associated with each of its constituent tasks. The reliability value of the application G is denoted as:

$$R_e(G) = \prod_{\tau_i \in X} R_e(\tau_i) \tag{29}$$

To determine the application's reliability bounds, an evaluation across all available processors is conducted. The minimum and maximum reliability values are then derived using the respective equations:

$$R_{e(min)}(\tau_i) = \min_{Y_l \in Y} R_e(\tau_i, Y_l, f_{l,low}) \tag{30}$$

$$R_{e(max)}(\tau_i) = 1 - \prod_{Y_l \in Y} (1 - R_e(\tau_i, Y_l, f_{l,max})) \tag{31}$$

As per Eq. (29), the reliability of application G is the product of task reliabilities. Hence the minimum and maximum reliability values of G can be computed as:

$$R_{e(min)}(G) = \prod_{e(min)} (\tau_i) \tag{32}$$

$$R_{e(max)}(G) = \prod_{e(max)} (\tau_i) \tag{33}$$

The application is deemed reliable if its reliability metric satisfies the specified reliability goal, denoted as $R_{e(goal)}(G)$ i.e.,

$$R_{e(min)}(G) \leq R_{e(goal)}(G) \leq R_{e(max)}(G) \tag{34}$$

4.5. Description of scheduling problem

The focus of this subsection is on the tasks scheduling problem in distributed computing environments consisting of parallel applications G and heterogeneous processor set Y. Specifically, it addresses a scenario where tasks must be executed on a set Y of processors that support varying frequency levels. This assignment must simultaneously optimize two critical objectives: minimizing overall energy consumption and ensuring the application's reliability metric $R_e(G)$ meets or surpasses a predefined reliability goal, $R_{e(goal)}(G)$. Ultimately, the goal is to judiciously map tasks to processor-frequency combinations that strike an optimal balance between reducing energy usage and boosting system reliability for the given application.

$$\hat{E}_{total}(G) = \hat{E}_s(G) + \hat{E}_d(G)$$
$$\text{subject to: } R_e(G) = \prod_{\tau_i \in X} R_e(\tau_i) \geq R_{e(goal)}(G)$$

5. Proposed hybrid approach

This study presents an inventive methodology that seamlessly integrates the HWWO algorithm and the DVFS technique, addressing the crucial need for energy-efficient tasks scheduling strategies that can adeptly address both static and dynamic energy considerations. The dynamic adjustment of processor voltage and frequency, facilitated by the DVFS mechanism, enables energy consumption optimization during task execution. This integrated methodology possesses the potential to revolutionize tasks scheduling in computational environments by achieving substantial energy savings, enhancing system reliability, and concurrently optimizing computational time. The HWWO algorithm employs a mutation operator in conjunction with an insert-reversed block operation, contributing to the minimization of static energy consumption. Complementing this, the DVFS technique is strategically employed to tackle dynamic energy consumption, further augmenting the energy-efficiency of the proposed solution. The incorporation of the WOA and the GWO into the task assignment methodology is substantiated by their exceptional versatility. These techniques have consistently exhibited superior performance in addressing assignment issues, demonstrating remarkable convergence characteristics [89].

In any SI algorithm, achieving a balance between exploitation and exploration is crucial for its effectiveness in terms of convergence speed and solution quality. As underscored in [30], the WOA algorithm showcases remarkable performance regarding convergence speed while adeptly balancing exploration and exploitation in addressing optimization issues. Although the WOA demonstrates effectiveness compared

to conventional algorithms, it may encounter challenges such as struggling to evade local optima because of its encircling search mechanism and insufficient solution enhancement after each iteration. These limitations of WOA prompted the proposal of a hybrid approach with GWO. GWO excels in exploitation, offering a solution to WOA's challenges through two strategies: preserving the best solution per iteration and evaluating new solutions with the best one during exploration. If the outcome improves upon the best solution, agents' positions change; otherwise, they remain unchanged. Additionally, to enhance the HWWO algorithm, authors proposed integrating mutation operators into WOA and combining it with GWO.

Mapping tasks to multiprocessors poses a significant NP-hard challenge. Consequently, the hybrid metaheuristic HWWO scheduling technique is employed as a strategic solution. This approach unfolds across two distinct phases: initial task prioritization, wherein tasks are sequenced in descending order of priorities, and subsequent task allocation to suitable processors.

Phase 1

5.1. Prioritizing tasks and deadline determination

To calculate the lower bound for tasks scheduling, this study utilizes the heterogeneous earliest-finish-time (HEFT) algorithm proposed by [86]. The HEFT algorithm is chosen due to its proven effectiveness in generating high-quality schedules for heterogeneous computing systems, making it a reliable choice for this purpose. Obtaining an exact lower bound is a challenging endeavor, so the scheduling length estimated by the HEFT method is adopted as the standard. It is further assumed that the application's deadline constraint is not known until after this lower bound reference has been determined.

The strategic allocation of tasks stands as a pivotal challenge in the context of (DAG) list scheduling within heterogeneous distributed systems. In an endeavor to tackle this challenge, the present article adopts the dynamic variant rank HEFT algorithm (DVR HEFT), as introduced by [90]. As per the findings elucidated by [90], the DVR HEFT algorithm demonstrates enhanced performance over its conventional counterpart, HEFT, by yielding superior outcomes while concurrently mitigating time complexity. By utilizing this improved task prioritization approach, the DVR HEFT algorithm aims to enhance the efficiency of scheduling interdependent tasks across heterogeneous computing resources within a distributed system. In its initial phase, akin to other static algorithms, the DVR HEFT algorithm undertakes the computation of task priorities. Within the DVR HEFT framework, this entails meticulously establishing task priorities through a comprehensive evaluation of their upward rank values, following a procedure similar to that of the HEFT algorithm. The upward rank, denoted as $Rank_U(\tau_i)$, for a given task τ_i, is recursively determined through the following equation:

$$Rank_U(\tau_i) = \left\{ f(\hat{w}_i) + \max_{\tau_k \in succ(\tau_i)} [avr(\hat{c}_{i,k}) + Rank_U(\tau_k)] \right\} \tag{35}$$

where, the symbol $\hat{w}_i$ represents the $\hat{W}CET$ of task τ_i. The task weight value, produced by the function $f(\hat{w}_i)$, is contingent upon the task's execution duration on each processor whereas $\hat{c}_{i,k}$ signifies the $\hat{W}CRT$ between the tasks τ_i and τ_k.

In a heterogeneous computing environment, the time required to execute a task can fluctuate based on the capabilities and performance characteristics of the specific machine handling that task. As a result, in such heterogeneous settings, there exist multiple distinct methodologies to calculate the computational weight associated with each node or task. The approach chosen to determine a node's weight $\hat{w}_i$ could potentially optimize computation time in certain scenarios, however, it does not guarantee improvements across all cases. As a result, the researchers in [90] proposed three distinct methods for calculating the upward rank values assigned to tasks. These alternative schemes for determining task priorities are described as follows:

1. Determine task weights through the calculation of the average execution time across all processors, mirroring the HEFT algorithm's methodology. In the HEFT algorithm, the function $f(\hat{w}_i)$ is computed by averaging the execution time of τ_i across each machine, depicted as:

$$f(\hat{w}_i) = avr(\hat{w}_{Y_1}, \hat{w}_{Y_2}, \ldots, \hat{w}_{Y_{|Y|}}) \tag{36}$$

2. Alternatively, it can be derived from the best-case scenario.

$$f(\hat{w}_i) = \min(\hat{w}_{Y_1}, \hat{w}_{Y_2}, \ldots, \hat{w}_{Y_{|Y|}}) \tag{37}$$

3. Task weighting based on the worst-case scenario.

$$f(\hat{w}_i) = \max(\hat{w}_{Y_1}, \hat{w}_{Y_2}, \ldots, \hat{w}_{Y_{|Y|}}) \tag{38}$$

Each of these schemes results in a unique task ordering. For instance, when applied to the DAG example shown in Fig. 4, they produce different upward rank lists for tasks. The most optimal ranking is achieved by the first scheme, the HEFT algorithm, with the corresponding values for the tasks depicted in Fig. 4 summarized in Table 4.

Upon employing the DVR HEFT technique to generate an optimized upward rank list for the tasks, the next step involved calculating the computation time required to allocate each task to the available processors. This allocation process followed the methodology outlined in the HEFT algorithm. The task with the highest rank value is identified and then assigned to the processor that could complete its execution at the earliest possible finish time (EFT). To determine the earliest feasible execution time for a given task τ_k on processor Y_l, the values for the $EST(\tau_k, Y_l)$ (earliest start time) and $EFT(\tau_k, Y_l)$ are calculated recursively as follows:

$$\begin{cases} EST(\tau_{entry,Y_l}) = 0; \\ EST(\tau_k, Y_l) = max \begin{cases} avail[l], \\ \max\limits_{\tau_i \in pred(\tau_k)} \{AFT(\tau_i) + \hat{c}_{i,k}\} \end{cases} ; \end{cases} \tag{39}$$

and $EFT(\tau_k, Y_l) = EST(\tau_k, Y_l) + \hat{w}_{k,l}$ (40)

The term $avail[l]$ denotes the earliest moment when processor Y_l is ready to execute a task, while $AFT(\tau_i)$ refers to the actual finish time of task τ_i. If tasks τ_i and τ_k are allocated to the same processor, the variable $\hat{c}_{i,k}$ is assigned a value of zero. The makespan of the application defines the exact completion time of task τ_{exit}, accounting for the scheduling of all tasks within a DAG. As previously described, the process to compute the lower bound of application G unfolds as follows:

$$LB(G) = LB(\tau_{exit}) \tag{41}$$

Therefore, the relative deadline can be fulfilled. For the illustrated example in Fig. 4, the application's deadline is considered as the sum of its lower bound and 20 [86].

The comparative scheduling results for the specified DAG illustrated in Fig. 4 have been determined using a variety of algorithms, including HEFT [86], DECM [37], energy-aware processor merging (EPM) [91], reliability enhancement under energy and response time constraints (REREC) [38], and energy-efficient scheduling with a reliability goal (ESRG) [8]. Assessing energy consumption and reliability involves referencing the power parameters listed in Table 5 for all processors. Each processor's energy-efficient frequency, denoted as f_{ee}, is calculated based on Eq. (19), while the maximum frequency, f_{max}, for each processor is considered to be 1.0, as indicated in previous studies [8,91]. The schedule generated by employing the HEFT algorithm at its maximum frequency level is graphically presented in Fig. 5. The aggregate energy, denoted as $\hat{E}_{total}(G)$, and the overall reliability, represented as $R_e(G)$, resulting from the execution of the HEFT algorithm, are quantified as 170.52 and 0.880426, respectively [91]. The visual depictions of the scheduling outcomes for the other algorithms are illustrated in Figs. 6–9. It is worth highlighting that the processors displayed with shading

Table 4
$Rank_U$ values for tasks illustrated in Fig. 4.

Tasks	τ_1	τ_3	τ_2	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9	τ_{10}
$Rank_U$	133.19	118.19	115.86	114.19	101.53	87.27	70.86	59.86	44.36	14.7

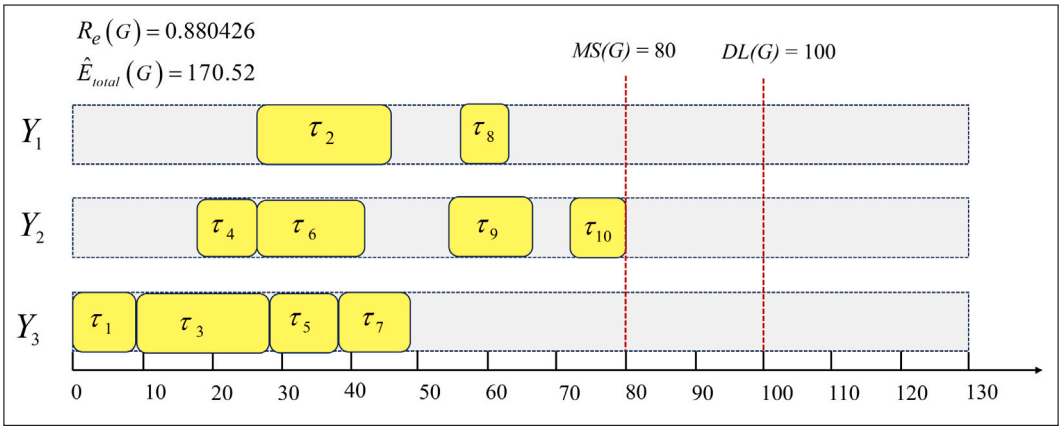

Fig. 5. Scheduling result of the DAG shown in Fig. 4 using the HEFT technique.

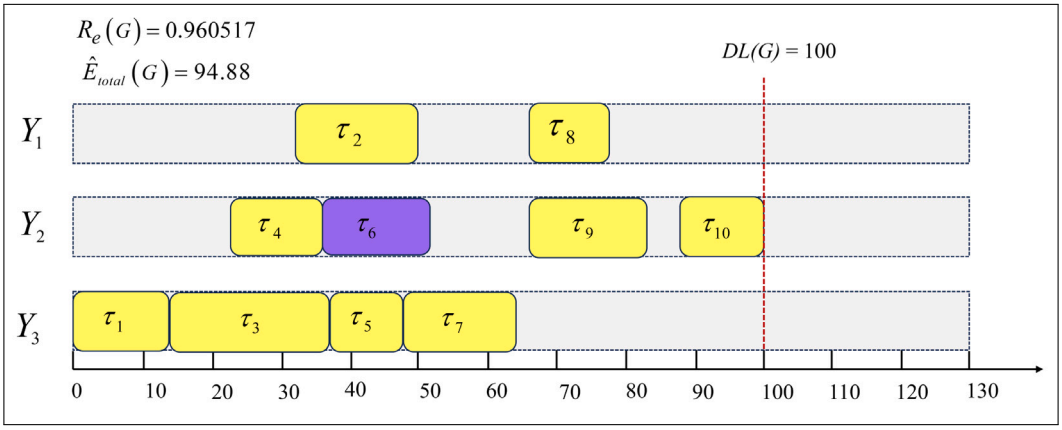

Fig. 6. Scheduling result of the DAG shown in Fig. 4 using the DECM technique.

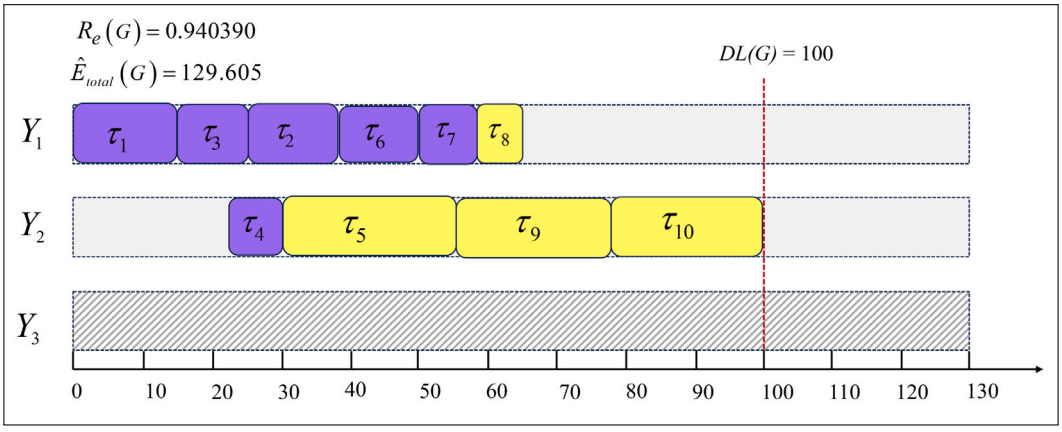

Fig. 7. Scheduling result of the DAG shown in Fig. 4 using the EPM technique.

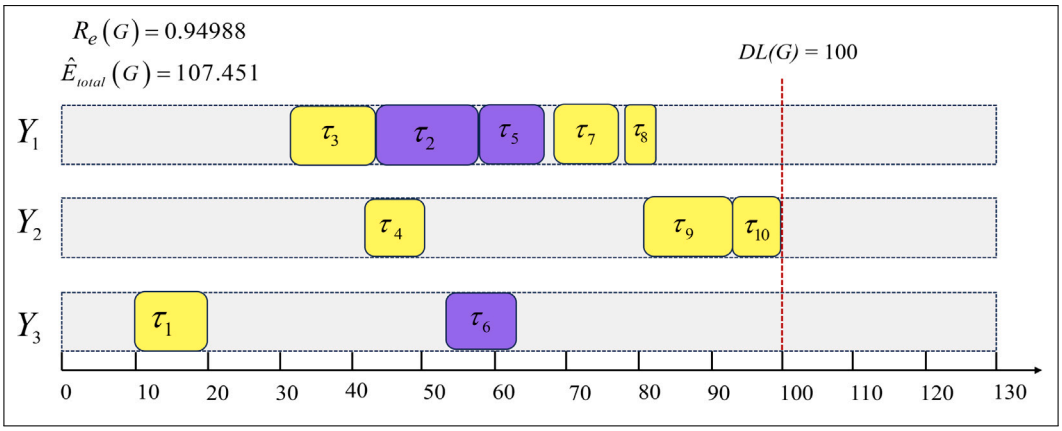

Fig. 8. Scheduling result of the DAG shown in Fig. 4 using the REREC technique.

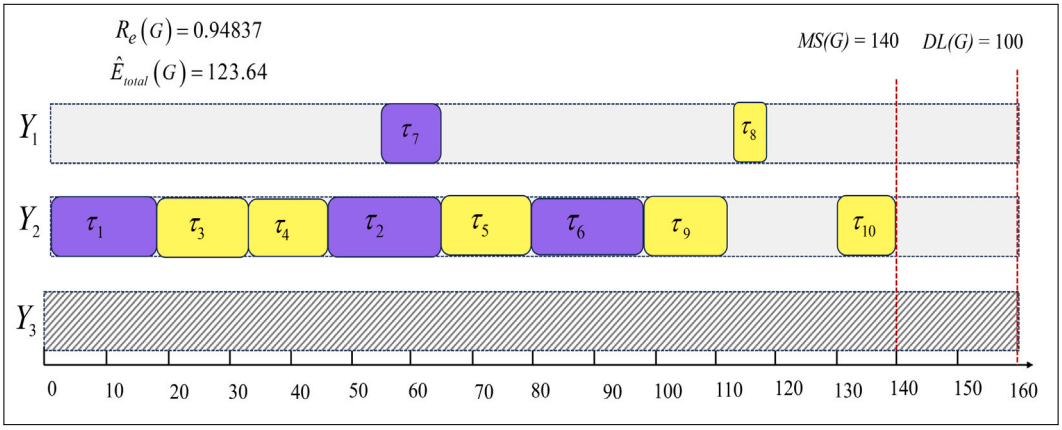

Fig. 9. Scheduling result of the DAG shown in Fig. 4 using the ESRG technique.

Table 5
Power parameters of processors ($Y_1, Y_2, and Y_3$).

Y_l	$P_{l,s}$	$C_{l,ef}$	$P_{l,ind}$	m_l	$f_{l,low}$	$f_{l,max}$	$\lambda_{l,max}$
Y_1	0.3	0.8	0.06	2.9	0.33	1.0	0.0005
Y_2	0.2	0.7	0.07	2.7	0.29	1.0	0.0002
Y_3	0.1	1.0	0.07	2.4	0.29	1.0	0.0009

in the figures signify their inactive or idle status during the execution of the respective schedules.

The tasks displayed in blue indicate a higher execution frequency. Initially, all processors shown in Fig. 6 and processors Y_1 and Y_2 illustrated in Fig. 7 become active. Following that, every processor depicted in Fig. 8 is activated. Afterward, Fig. 9 demonstrates the activation of processors Y_1 and Y_2 specifically.

The total energy consumption when employing the DECM algorithm (depicted in Fig. 6) is lower than other algorithms, followed by REREC and ESRG algorithms. Crucially, this technique also enhances the system's reliability compared to these other methods. By analyzing Figs. 5–9, it becomes evident that the DECM technique delivers superior performance concerning both energy efficiency and reliability, surpassing the EPM, HEFT, REREC, and ESRG algorithms.

Phase 2

5.2. DVFS-integrated hybrid WOA-GWO scheduling model

The section before this one explains the task prioritization process, where tasks are arranged in descending order of priorities by employing the DVR HEFT technique. This method aids in reducing the static energy consumed. This section proposes a novel algorithmic approach that leverages the DVFS-integrated EES technique. This algorithm incorporates a HWWO model, featuring an insert-reversed block operation and a mutation operator. The primary goal is to efficiently assign tasks to appropriate processors within computational environments, thereby achieving substantial reductions in energy consumption while simultaneously enhancing the overall reliability of the system. The mutation operator's main role is to diversify the population and boost HWWO's global exploration. This study employs the inversion mutation operator to fulfill this function.

5.2.1. Termination criteria

Optimization algorithms must have proper stopping conditions to ensure they do not run indefinitely. The algorithm iterates until convergence to find the best solution, making it essential to set termination criteria to assess the optimization process's convergence. In this case, the algorithm is designed to carry out 20 optimization iterations initially, before assessing any convergence criteria. Subsequent to this initial phase, it examines two criteria:

i The first criterion measures the change in the fitness function between the latest optimization iteration $f_{fn}(r)$ and the previous iteration $f_{fn}(r-1)$. When the relative change is less than a specified tolerance ϵ_f e.g., $\epsilon_f = 0.001$ or 0.1%, the stop criterion is met. This stop criterion is defined as:

$$\frac{f_{fn}(r) - f_{fn}(r-1)}{f_{fn}(r)} < \epsilon_f \tag{42}$$

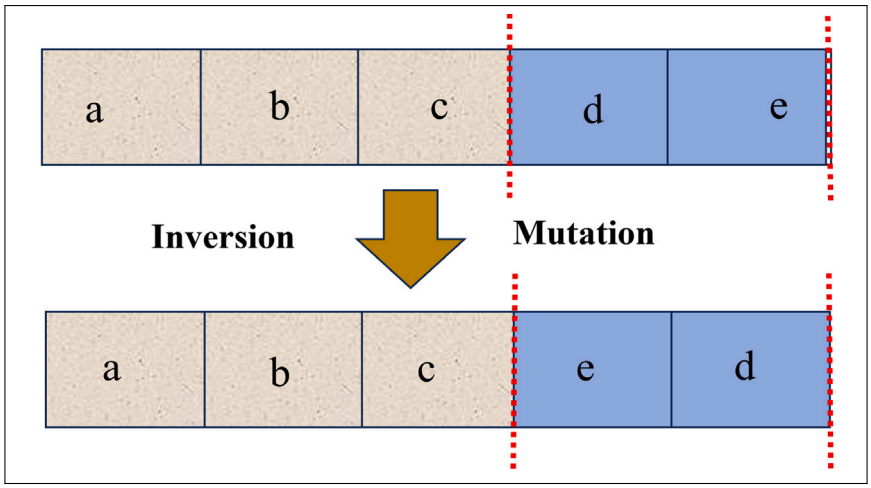

Fig. 10. Inversion mutation operator.

ii If the above criterion is not satisfied, the optimization process halts after reaching the maximum number of iterations, set here to 500.

5.2.2. Scheduling fitness function

HWWO integrates metaheuristic techniques for tasks scheduling, with the effectiveness contingent upon the crafted fitness function determining suitable computing resources. Eq. (43) outlines the formulation of the fitness function for each particle in HWWO.

$$f_{fn(\tau_i,Y_l,f_{l,h})} = min\left[\sigma_1 * \sum_{l=1}^{|Y|} P_{l,s} * \frac{MS(G)}{DL(G)} + \sigma_2 * \sum_{i=1}^{|X|} e^{-\lambda_{l,max}*\left(\frac{\hat{w}_{i,l}*f_{l,max}}{f_{l,h}}\right)}\right] \quad (43)$$

Here, σ_1 and σ_2 signify the optimal weights for energy consumption and computation time, subject to $\sigma_1 + \sigma_2 = 1$. During selection, each solution's objectives receive random weights, encouraging exploration in various directions.

5.2.3. Inversion mutation

Inversion mutation [92], an operator in evolutionary algorithms, randomly selects a segment of genes within a chromosome and reverses their order. This process fosters genetic diversity by exploring different regions of the search space, potentially leading to improved solutions. This can be understood by referring to the illustrative example presented in Fig. 10.

5.2.4. Insert-reversed block operation

Excessive mutation can disrupt previously found good solutions and impede the algorithm's convergence toward optimal or near-optimal solutions. This occurs because random changes introduced by mutation may not always improve the solutions. Consequently, after applying the inversion mutation operator, the insert-reversed block operation [93] is integrated into the algorithm. This operation inserts a reversed block of task permutation into all $(|X| - 1)$ conceivable positions within a $|X|$-dimensional search agent, as outlined in Algorithm 1 and illustrated in Fig. 11.

5.2.5. HWWO scheduling algorithm

The pseudocode in Algorithm 2 outlines the procedure for tasks scheduling employing the HWWO technique. A more in-depth elucidation of these steps is presented in the following paragraphs:

Step 1: Whale-wolf encoding and position vector initialization.

The developed technique views tasks as elements within the HWWO framework, which systematically refines their designated positions over successive iterations, ultimately converging on an optimal task allocation among the available processors. One of the key obstacles in formulating an effective HWWO approach for tackling optimization challenges lies in the intricate process of developing an appropriate encoding mechanism to model the constituent elements or search agents within the HWWO framework. For optimization scenarios involving $|Y|$ processors, a viable strategy to model the constituent search particles is through the utilization of $|Y|$-dimensional coordinate vectors, as illustrated in the subsequent representation.

$$\vec{X} = (\vec{X}_1, \vec{X}_2, \ldots, \vec{X}_{|Y|})$$

Algorithm 1 Pseudo code for the insert-reversed block operation

```
temp = random(2, |X|/2);
arr positions [temp];
function REPLACE(search_agent, i)
    for j = 0 to temp - 1 do
        search_agent[j] = switch(i, positions[value - j - 1]);
    end for
    return search_agent;
end function
function INSERT_REVERSED_BLOCK_OPERATION(assignment)
    matrix permutations[|X| − 1][|X|];
    positions = sorted(random(0, |X| - 1, temp));   ▷ Select 'temp' number of positions of tasks.
    permutations[0] = assignment;
    for i = 0 to |X| - 3 do
        if i not in positions then
            permutations[i + 1] = replace(permutations[i], i);
        end if
    end for
end function
```

Step 2: Evaluating fitness values for each particle in assignment.

This step calculates the fitness score for each HWWO particle, reflecting the task-to-processor assignment. The hybrid WOA–GWO framework evaluates individual particle fitness using the function defined in Eq. (43). When a particle's current fitness exceeds its previous best, the global best fitness is updated to the new higher value.

Step 3: Updating the position vector.

The proposed HWWO approach updates the position of each whale using equations ((2), (8), (10)) under different scenarios, whereas Eq. (17) facilitates the positional update of the wolf for every iteration,

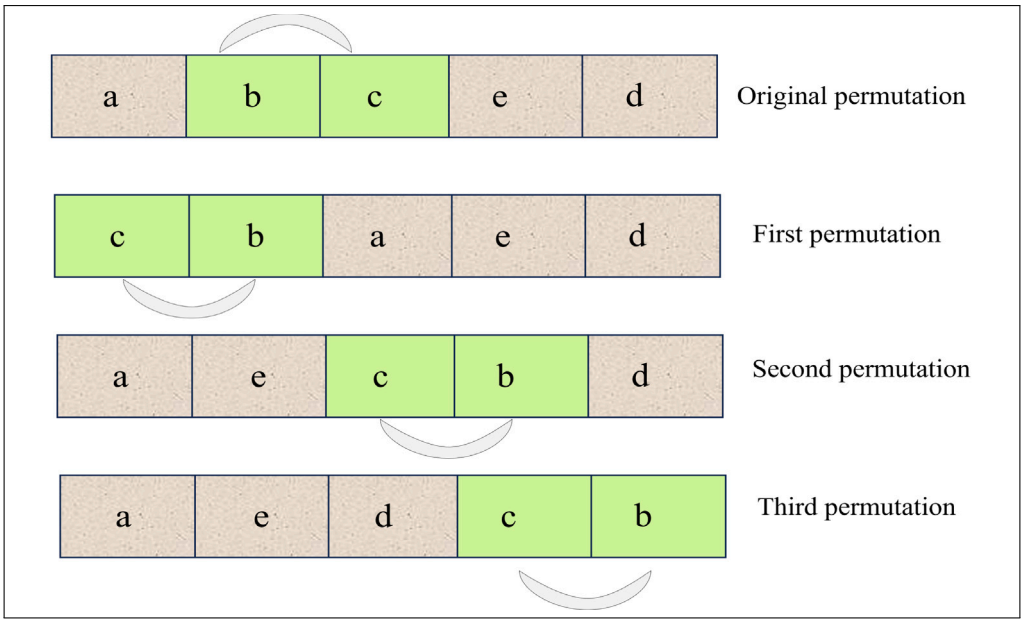

Fig. 11. Example demonstrating the insert-reversed block operation.

taking into consideration the values of $\vec{Z}_{\alpha}$, $\vec{Z}_{\beta}$, and $\vec{Z}_{\delta}$ as defined by Eqs. (13)–(16).

Step 4: Emergence of optimal scheduling.

The algorithm models task assignment using whales and wolves. Applying mutated and permutated HWWO, particle positions are adjusted for optimal tasks scheduling on processors. With optimal scheduling obtained, the EES algorithm then optimizes voltage and frequency distribution among all processors.

Step 5: Calculating total energy consumption, and reliability.

The aggregate energy consumption of the resulting set of particles can be determined by utilizing Eq. (25), while their reliability metric is ascertained through the application of Eq. (29).

The outlined steps should be iteratively executed until the stopping criteria or termination condition is met. The following provides a comprehensive description of Algorithm 2.

Fig. 12 visually depicts the scheduling outcomes obtained with the proposed HWWO algorithm, and Fig. 13 illustrates the flowchart of the proposed model. The algorithm effectively reduces total energy consumption and enhances reliability. Using Eq. (25), the total energy consumption $\hat{E}_{total}(G)$ is calculated as 75.673 units, with $\hat{E}_{s}(G)$ being 60.6 units and $\hat{E}_{d}(G)$ being 15.073 units. Furthermore, to satisfy the reliability goal $R_{e(goal)}(G)$ i.e., $R_{e(min)}(G) \leq R_{e(goal)}(G) \leq R_{e(max)}(G)$, the maximum reliability $R_{e(max)}(G)$ and the minimum reliability $R_{e(min)}(G)$ for the DAG are evaluated as 0.99989 and 0.89738 respectively. Here, the reliability goal, $R_{e(goal)}(G)$, is set at 0.95, and the obtained reliability, $R_{e}(G)$, stands at $0.978526 > R_{e(goal)}(G)$. The improvement is clear when contrasted with the energy consumption and reliability results shown in Figs. 5–9, representing the HEFT, DECM, EPM, REREC, and ESRG algorithms.

The results for the DAG, as depicted in Fig. 4, clearly highlight the superiority of the proposed algorithm in reducing energy consumption and enhancing system reliability.

5.2.6. Time and space complexities

The computational time complexities required by the WOA and the GWO expressed as $O(|X| * |Y|)$. In the proposed HWWO technique, the time taken for steps 14–17, during which the mutation operation is performed on each assignment, exhibits an identical complexity of $O(|X| * |Y|)$. Moreover, the evaluation of the fitness function for each assignment, carried out between lines 19–21, has a time complexity of $O(|Y|)$. The iterative loop spanning steps 33 to 63 also requires a time complexity scaling as $O(|Y| * |X|)$. As a result, the overall time complexity per iteration of the HWWO algorithm is the cumulative sum of these individual complexities, amounting to $O(|X| * |Y|) + O(|Y|)$.

To store the assignments of $|X|$ tasks across $|Y|$ processors, a matrix structure of size $|Y| * |X|$ is required, and an array of length $|Y|$ is needed to hold the fitness value of each assignment. Consequently, the overall space complexity for representing and evaluating these assignments is $O(|X| * |Y|) + O(|Y|)$.

Alternatively, the algorithms based on energy considerations, such as ESRG, EPM, and REREC techniques, exhibit time complexities of $O(|X|^2 * |Y|)$, $O(|X|^2 * |Y|^3)$, and $O(|X|^2 * |Y|)$, respectively. It is noteworthy that, in terms of time complexity, the proposed algorithm surpasses the performance of the other existing algorithms in this domain.

6. Experimental evaluation

This study aims to develop an approach that improves system reliability, reduces the overall computation time, and conserves energy usage. The proposed model's performance in handling tasks scheduling across heterogeneous distributed systems has been rigorously evaluated through a comprehensive set of experiments. This section outlines the simulations conducted and the evaluation metrics configured to address the stated problem using the proposed algorithm. An in-depth analysis of each experiment is provided in the following subsections.

6.1. Experimental metrics

The HWWO algorithm's efficacy in distributed computing is comprehensively evaluated using multiple performance metrics for analytical assessment. These metrics cover total energy consumption, system reliability, computation time, resource utilization, SLR, CCR, and sensitivity analysis. This comprehensive assessment aims to provide insights into the algorithm's performance and its potential impact on the distributed computing environment.

With regard to performance evaluation, the proposed algorithm's capabilities are benchmarked through two distinct stages. Firstly, a comparative analysis is conducted between the introduced HWWO

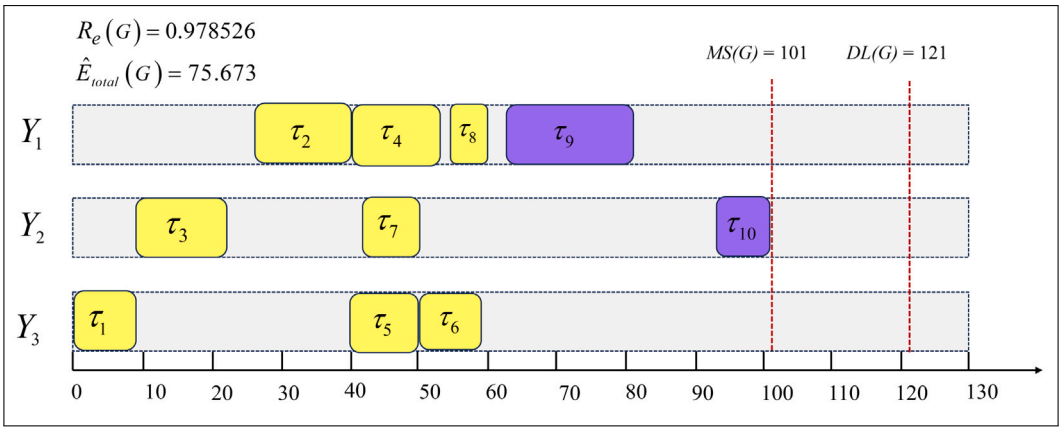

Fig. 12. Best scheduling result of the DAG shown in Fig. 4 using proposed HWWO technique.

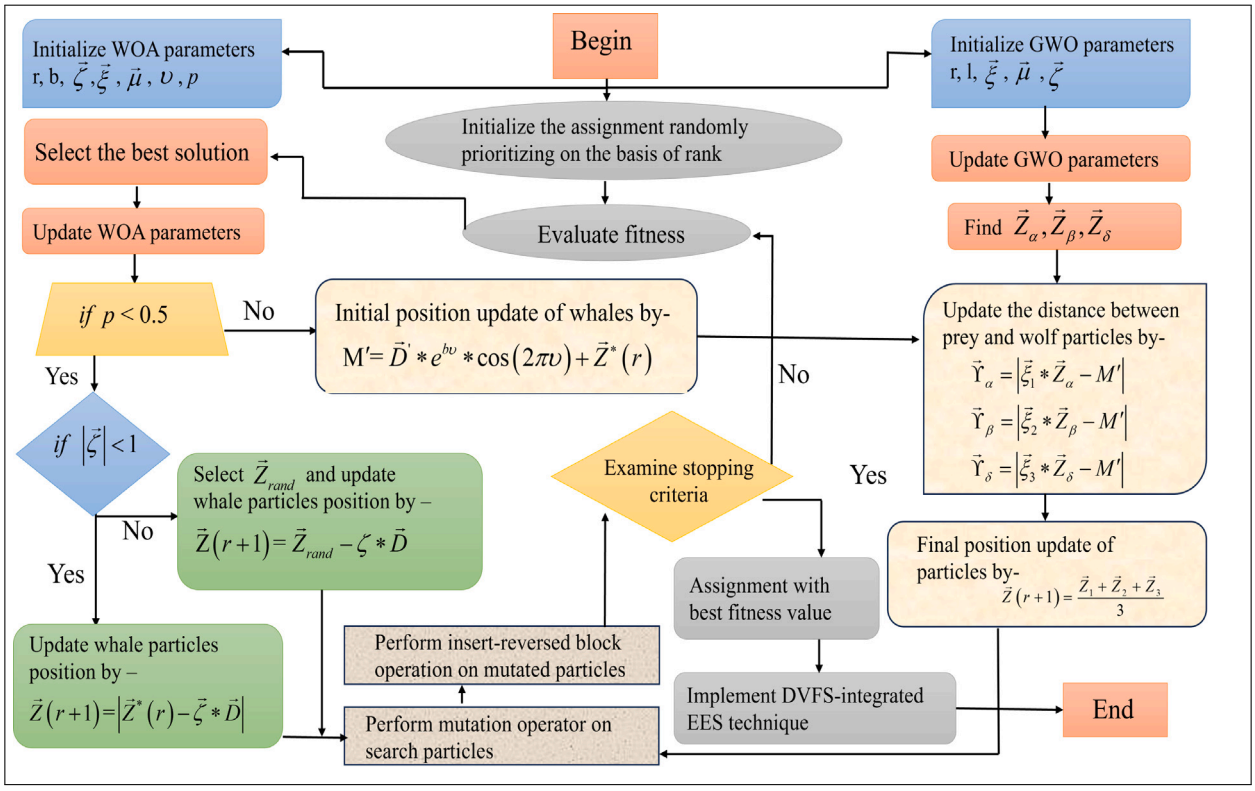

Fig. 13. Flowchart of the proposed HWWO model.

technique and several existing state-of-the-art methods, namely HEFT, DECM, EPM, REREC, and ESRG. Additionally, the successful state-of-the-art algorithms from the 'CEC2020 competition on real-world single objective constrained optimization' – specifically, SASS [94], sCMAgES [95], EnMODE [96], and COLSHADE [97] – are incorporated as four benchmark algorithms for comparative evaluation in the context of real-world optimization challenges, as outlined in the relevant literature [98]. Furthermore, the proposed methodology is evaluated against several metaheuristic approaches, including PSO, GWO, ACO, KH, WOA, DA (dragonfly algorithm) [67], and AHA (artificial hummingbird algorithm) [67], using various performance metrics. The algorithm's capabilities are also assessed through a series of benchmark tests involving unimodal test functions, which are compared against the aforementioned metaheuristic techniques. This comparative analysis is carried out across multiple scenarios, generated by varying the number of tasks over different generations, to provide a comprehensive understanding of the algorithm's performance under diverse conditions. The assessment of the presented benchmark suite is conducted on a personal computer equipped with the Microsoft Windows 11 operating system, featuring an INTEL Core i3 CPU and 8 GB of RAM.

Stage I

6.2. Benchmark analysis with state-of-the-art algorithms

The subsection delves into an assessment of the proposed algorithm's effectiveness, drawing comparisons with state-of-the-art methods across various performance metrics. The evaluation encompasses three distinct scenarios, wherein the validation process incorporates FFT, GE, and constrained optimization challenges from the renowned

Algorithm 2 The HWWO tasks scheduling algorithm

Input:
Dataset $\hat{WCET}$
No. of processors, $|Y|$
No. of tasks, $|X|$
Population size, $|Y|$
Control coefficient, $\vec{\mu}$
Maximum no. of iterations, r
Output:
Global best solution (best tasks assignment)
Begin
matrix assignments[$|Y|$][$|X|$] = randomly assign $|X|$ tasks in the processors prioritizing on the basis of rank;
tasks = order by (rank, decreasing);
function FITNESS(assignment)
$f_{fn(\tau_i,Y_j,f_{j,h})} = min\left[\sigma_1 * \sum_{l=1}^{|Y|} P_{l,s} * \frac{MS(G)}{DL(G)} + \sigma_2 * \sum_{i=1}^{|X|} e^{-\lambda_{l,max} * \left(\frac{\hat{w}_{i,l} * f_{l,max}}{f_{l,h}}\right)}\right]$;
return $f_{fn(\tau_i,Y_j,f_{j,h})}$;
end function
function INVERSION_MUTATION(assignment, mutation_rate)
if $random(0,1) < mutation\_rate$ **then**
i, j = sorted(random($|X|$, 2));
assignment[i: $j+1$] = assignment[i: $j+1$][:: -1];
end if
return assignment;
end function
for $i = 0$ to $|Y| - 1$ **do**
assignments[i] = random assignment to processors (tasks);
assignments[i] = inversion_mutation(assignments[i], 0.2);
end for
arr fitness_values[Y];
for $i = 0$ to $|Y| - 1$ **do**
fitness_values[i] = fitness(assignments[i]);
end for
$\vec{Z}$[0] = alpha assignment;
$\vec{Z}$[1] = beta assignment;
$\vec{Z}$[3] = delta assignment;
function WOLF(M', $\vec{\mu}$, $\vec{\xi}$, $\vec{\zeta}$)
matrix wolf_particles[3][$|X|$];
for $i = 0$ to 2 **do**
$\vec{Y} = |\vec{\xi} * Z[i] - M'|$;
$wolf\_particles[i] = \vec{Z}[i] - \vec{\zeta} * \vec{Y}$;
end for
return $\frac{wolf\_particles[0]+wolf\_particles[1]+wolf\_particles[2]}{3}$;
end function
while $r <$ max_iteration **do**
$\vec{\mu} = 2 - r * \left(\frac{2}{max\_iteration}\right)$;
b = random constant;
update best assignments on the basis of least fitness value;
for $i = 0$ to $|Y| - 1$ **do**
update r, $\vec{\zeta}$, $\vec{\xi}$, ν, p;
if $p < 0.5$ **then**
$\vec{D} = |\vec{\xi} * best\_assignment - assignments[i]|$;
if $|\vec{\zeta}| < 1$ **then**
assignments[i]=$|best\_assignment - \vec{\zeta} * \vec{D}|$;
else
j = random(0, $|Y|$);
$assignments[i] = assignments[j] - \vec{\zeta} * \vec{D}$;
end if
else
$\vec{D'} = |best\_assignment - assignments[i]|$;
$M' = \vec{D'} * e^{b\nu} * cos(2\pi\nu) + assignments[i]$;
assignments[i] = wolf($M', \vec{\mu}, \vec{\zeta}, \vec{\xi}$);
end if
assignments[i] = inversion_mutation(assignments[i], 0.2);
assignments[i] = insert_reversed_block_operation (assignments[i]);
end for
if $\left(\frac{f_{fn}(r)-f_{fn}(r-1)}{f_{fn}(r)} < \epsilon_f (= 0.001)\right)$ **then**
r = (max_iteration);
else
$r = r + 1$;
end if
for $i = 0$ to $|Y| - 1$ **do**
fitness[i] = fitness(assignments[i]);
end for
end while
best_assignment = assignment with least fitness value;
Implement the EES algorithm to optimize voltage and frequency distribution among all processors

Table 6
Parameter setting for stage I.

Parameter	Values
$\hat{w}_{i,l}(ms)$	[10, 100]
$\hat{c}_{i,k}(ms)$	[10, 100]
$P_{l,s}$	[0.1, 0.5]
$P_{l,ind}$	[0.03, 0.07]
$C_{l,eff}$	[0.8, 1.2]
m_l	[2.5, 3.0]
$f_{l,max}$	1 GHz
l_1 and l_2	[0, 1]
$\vec{\mu}$	[0, 2]
$\lambda_{l,max}$	[0.0003, 0.0009]
CCR	0.1, 0.5, 1, 5, 10

CEC2020 competition. Complementing these scenarios, a diverse array of experiments centered around tasks scheduling in a multiprocessing environment is conducted. The validation outcomes underscore the algorithm's efficacy. It is noteworthy that each algorithm undergoes independent iterations, refining its functions in pursuit of optimal performance. The comprehensive evaluation aims to understand the algorithm's capabilities in addressing real-world optimization and scheduling challenges. The validation of the algorithm is facilitated through simulations conducted within a replicated heterogeneous distributed embedded system environment, comprising 95 processors capable of handling tasks of varying complexities. These processors exhibit diverse processing capabilities, with their specifications and the corresponding application parameters closely mirroring the details outlined in Refs. [8,91]. The input data parameters employed in this stage are delineated in Table 6, wherein each frequency magnitude undergoes discretization and is represented with a precision of 0.01 GHz. Following the simulations, a comprehensive evaluation is undertaken, wherein various assessment metrics are computed, including the average execution time, as well as the standard deviation and mean of solutions across the iterative process, as described in [98]. This evaluation replicates real-world heterogeneous distributed embedded systems, providing insights into the algorithm's performance.

6.2.1. Scenario 1

This scenario conducts a rigorous assessment of the efficacy of the HWWO-based technique through an examination of 15 real-world optimization problems. The assessment utilizes four algorithms identified in the 'CEC2020 competition on real-world single objective constrained optimization', namely the SASS algorithm, the sCMAgES algorithm, the EnMODE, and the COLSHADE algorithm. The article leverages these algorithms to evaluate the performance of the HWWO-based

Table 7
15 real-world constrained optimization problems.

Problem	Name	D	g	h
C_{opt1}	Optimal power flow (minimization of active power loss)	126	0	116
C_{opt2}	Topology optimization	30	30	0
C_{opt3}	Process flow sheeting problem	3	3	0
C_{opt4}	Gas transmission compressor design (GTCD)	4	1	0
C_{opt5}	SOPWM for 3-level inverters	25	24	1
C_{opt6}	Optimal power flow (minimization of fuel cost)	126	0	116
C_{opt7}	Optimal power flow (minimization of active power loss and fuel cost)	126	0	116
C_{opt8}	SOPWM for 5-level inverters	25	24	1
C_{opt9}	Pressure vessel design	4	4	0
C_{opt10}	Optimal sizing of distributed generation for active power loss minimization	153	0	148
C_{opt11}	Wind farm layout problem	30	91	0
C_{opt12}	Microgrid power flow (islanded case)	76	0	76
C_{opt13}	Optimal setting of droop controller for minimization of active power loss in islanded microgrids	86	0	76
C_{opt14}	Microgrid power flow (grid-connected case)	74	0	74
C_{opt15}	Optimal setting of droop controller for minimization of reactive power loss in islanded microgrids	86	0	76

technique on the selected real-world optimization problems. The article presents a detailed description of the attributes that define the real-world problems under investigation, including their dimensions and the number of equality and inequality constraints involved. This information is comprehensively outlined in Table 7. To ensure a fair and consistent comparison, the parameters of the four algorithms are maintained at their original settings as documented in the relevant literature [94–97]. To ensure an impartial evaluation, the proposed algorithm is executed for 500 iterations, adhering to the guidelines outlined in [98]. This ensures an equal number of function evaluations across methods. Subsequently, a statistical analysis is conducted using the Wilcoxon signed-rank test [99] to assess the HWWO's performance relative to the other algorithms under consideration.

The simulation outcomes for 15 real-world challenges are showcased in Table 8, which provides comprehensive information regarding the best fitness value, mean fitness value, worst fitness value, and the standard deviation (St. Dev) of the fitness values.

The results displayed in Table 8 demonstrate the proficiency of the HWWO algorithm in tackling the majority of these problems, exhibiting commendable performance. Notably, some of the solutions obtained by HWWO surpass those achieved by competing algorithms. To facilitate a comprehensive comparison of algorithm performance on the proposed benchmark suite, we have adopted the ranking methodology outlined in the CEC2020 competition [98]. The evaluation process assigns weighted scores to the best, mean, and median results obtained from 25 independent runs of each algorithm to quantify their performance, as outlined in [98]. The weighted performance measures (PM) are: HWWO (0.321089, rank 1), SASS (0.335719, rank 2), EnMODE (0.351856, rank 3), sCMAgES (0.415387, rank 4), and COLSHADE (0.493992, rank 5). Outperforming the others, HWWO demonstrates its effectiveness in handling diverse real-world problems with acceptable performance.

The results of the Wilcoxon signed-rank test, presented in Table 9, highlight the effectiveness of the proposed HWWO algorithm when compared to other methods. The table shows the ranks of the HWWO algorithm relative to the second algorithm in terms of best fitness values, with T^+, T^-, and T indicating the statistical results. T^+ represents the superiority of the HWWO algorithm. The p-values, calculated at a 5% significance level, test the null hypothesis that the median difference between the algorithms is zero. Additionally, the final row of Table 9 summarizes the counts of T^+ and T^-, as well as the test statistic, offering a clear overview of the results. The Wilcoxon signed-rank test is particularly suited for paired comparisons in situations where normality cannot be assumed, making it a reliable tool for validating performance differences in real-world optimization problems. The results indicate significant improvements in the performance of the HWWO algorithm, with T^+ accounting for 86.96% and T^- for 13.03% of the evaluated

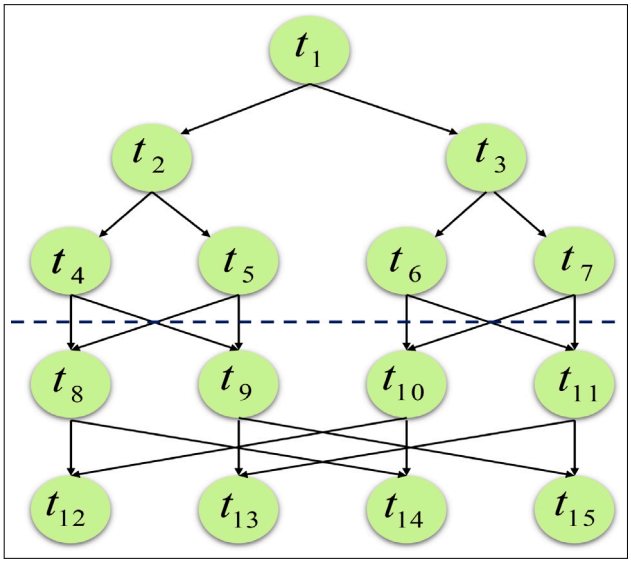

Fig. 14. DAG of FFT application with $\rho = 4$.

benchmarks ($p < 0.05$). These findings suggest that the HWWO algorithm provides faster convergence and improved accuracy in solving optimization problems, which could translate into better outcomes in practical applications such as logistics and scheduling optimization.

6.2.2. Scenario 2

In this scenario, the proposed approach rigorously evaluates the effectiveness of the HWWO technique through an analysis of the FFT algorithm. The visual depiction, illustrated in Fig. 14, showcases a parallel implementation of the FFT application [8,86], incorporating a crucial parameter value of $\rho = 4$. The parameter $'\rho'$ governs the application's task count $|X|$ via $|X| = (2 * \rho - 1) + \rho * log_2(\rho)$, and is exponentially related to an integer $'y'$ through $\rho = 2^y$. As illustrated in Fig. 14, the application achieves a task count of $|X| = 15$, which occurs when ρ is set to 4. The following are three experiments evaluated utilizing the FFT application.

Experiment 1: This experimental work aims to conduct a comprehensive comparative evaluation of the proposed HWWO technique against existing algorithms. The primary focus is to assess their respective performances concerning total energy consumption and computational time requirements within the context of parallel applications involving FFT. This experimental study employs a rigorous deadline and reliability constraints, expressed as $DL(G) = LB(G) * 1.4$ and $R_{e(goal)}(G) = 0.90$ respectively, where the complexity of the parallel application is

Table 8
Results of 15 real-world constrained optimization problems.

Problem	Performance metric	Optimization Algorithm				
		SASS	COLSHADE	EnMODE	sCMAgES	HWWO
C_{opt1}	Best fitness	**5.13E+4**	7.08E+4	7.10E+4	7.12E+4	5.19E+4
	Mean fitness	**5.33E+4**	7.08E+4	7.22E+4	7.22E+4	5.37E+4
	Worst fitness	**5.51E+4**	7.08E+4	7.61E+4	7.71E+4	5.66E+4
	St. Dev	1.07E+2	**2.12E−12**	1.14E+2	2.14E+2	1.15E+2
C_{opt2}	Best fitness	3.06E+4	3.06E+4	3.06E+04	3.06E+04	**3.05E+4**
	Mean fitness	**3.06E+4**	**3.06E+4**	3.08E+04	3.08E+04	**3.06E+4**
	Worst fitness	**3.06E+4**	3.07E+4	3.08E+04	3.08E+04	**3.06E+4**
	St. Dev	**3.07E−11**	2.51E−7	2.53E+1	2.51E+1	7.46E−1
C_{opt3}	Best fitness	3.68E+04	3.67E+4	3.67E+4	3.67E+4	**3.63E+4**
	Mean fitness	3.68E+04	3.69E+4	3.67E+4	3.67E+4	**3.63E+4**
	Worst fitness	3.68E+04	3.69E+4	3.67E+4	3.67E+4	**3.63E+4**
	St. Dev	1.69E−15	7.51E+1	4.53E−15	4.53E−15	1.66E−16
C_{opt4}	Best fitness	5.29E+05	5.46E+05	5.32E+05	5.45E+05	**5.26E+05**
	Mean fitness	5.29E+05	5.46E+05	5.35E+05	5.45E+05	**5.28E+05**
	Worst fitness	5.31E+05	5.48E+05	5.36E+05	5.48E+05	**5.28E+05**
	St. Dev	7.13E−4	**2.36E−4**	1.66E+2	3.31E−04	1.16E−1
C_{opt5}	Best fitness	1.60E+6	1.67E+06	**1.34E+06**	**1.34E+6**	**1.34E+06**
	Mean fitness	1.67E+6	1.67E+06	**1.34E+06**	1.38E+6	**1.34E+06**
	Worst fitness	1.67E+6	1.67E+06	**1.36E+06**	1.38E+6	**1.36E+06**
	St. Dev	2.06E+4	**1.05E−09**	3.66E−7	5.14E+2	3.66E−7
C_{opt6}	Best fitness	**6.85E+07**	6.91E+7	**6.85E+07**	**6.85E+07**	**6.85E+07**
	Mean fitness	**6.87E+07**	6.91E+7	**6.87E+07**	6.89E+07	**6.87E+07**
	Worst fitness	**6.89E+07**	6.93E+7	**6.89E+07**	**6.89E+07**	**6.89E+07**
	St. Dev	2.27E+04	**6.61E−04**	2.27E+04	5.42E+04	2.27E+04
C_{opt7}	Best fitness	**5.73E+6**	5.76E+6	**5.73E+6**	**5.77E+6**	**5.76E+6**
	Mean fitness	**5.74E+6**	5.79E+6	**5.74E+6**	5.79E+6	5.77E+6
	Worst fitness	**5.76E+6**	5.79E+6	**5.76E+6**	5.79E+6	5.77E+6
	St. Dev	1.05E+4	6.05E+4	1.05E+4	3.08E+4	**5.05E+3**
C_{opt8}	Best fitness	9.93E−4	9.97E−04	**8.92E−4**	**8.92E−4**	**8.92E−4**
	Mean fitness	9.95E−4	9.97E−04	**8.92E−4**	**8.92E−4**	8.95E−4
	Worst fitness	9.96E−4	9.97E−04	8.96E−4	**8.93E−4**	8.95E−4
	St. Dev	4.34E−6	5.41E−06	6.63E−4	**4.31E−6**	**4.31E−6**
C_{opt9}	Best fitness	**1.89E+2**	4.08E+03	4.08E+03	4.18E+03	**1.89E+2**
	Mean fitness	**1.91E+2**	4.25E+03	4.31E+03	4.26E+03	**1.91E+2**
	Worst fitness	**1.99E+2**	4.37E+03	4.37E+03	4.31E+03	**1.99E+2**
	St. Dev	**2.80E−1**	8.85E+01	5.85E+01	1.55E+01	**2.80E−1**
C_{opt10}	Best fitness	6.16E−02	6.74E−02	6.26E−02	**3.26E−02**	**3.26E−02**
	Mean fitness	**6.96E−02**	7.85E−02	7.59E−02	**6.96E−02**	**6.96E−02**
	Worst fitness	7.92E−02	9.04E−02	9.23E−02	**7.32E−02**	**7.32E−02**
	St. Dev	5.28E−02	5.26E−02	4.49E−05	**4.28E−05**	**4.28E−05**
C_{opt11}	Best fitness	3.06E+4	3.06E+4	3.11E+4	3.02E+4	**2.94E+4**
	Mean fitness	3.08E+4	3.09E+4	3.11E+4	3.07E+4	**2.94E+4**
	Worst fitness	3.11E+4	3.13E+4	3.13E+4	3.07E+4	**2.94E+4**
	St. Dev	7.12E+1	2.61E+1	4.64E−4	4.64E+1	**0**
C_{opt12}	Best fitness	**1.67E+1**	**1.67E+1**	1.68E+1	1.72E+1	1.70E+1
	Mean fitness	**1.67E+1**	1.68E+1	1.68E+1	1.74E+1	1.73E+1
	Worst fitness	**1.69E+1**	**1.69E+1**	1.71E+1	1.74E+1	1.73E+1
	St. Dev	**2.03E−1**	**2.03E−1**	3.08E−1	4.13E+1	**2.03E−1**
C_{opt13}	Best fitness	5.75E+2	5.71E+2	**5.24E+2**	5.57E+2	**5.24E+2**
	Mean fitness	5.78E+2	5.79E+2	**5.29E+2**	5.59E+2	**5.29E+2**
	Worst fitness	5.79E+2	5.79E+2	**5.33E+2**	5.61E+2	**5.33E+2**
	St. Dev	2.51E+1	4.43E+1	**1.61E+1**	7.01E+1	**1.61E+1**
C_{opt14}	Best fitness	3.55E+2	3.62E+2	3.60E+2	3.55E+2	**3.53E+2**
	Mean fitness	3.74E+2	3.78E+2	4.45E+2	**3.61E+2**	**3.61E+2**
	Worst fitness	3.79E+2	3.79E+2	4.71E+2	3.66E+2	**3.63E+2**
	St. Dev	8.01E+2	4.23E+2	7.32E+2	3.39E+1	**3.37E+1**
C_{opt15}	Best fitness	1.93E+5	**1.89E+5**	1.91E+5	**1.89E+5**	**1.89E+5**
	Mean fitness	1.95E+5	1.91E+5	1.96E+5	**1.89E+5**	1.92E+5
	Worst fitness	1.99E+5	1.97E+5	1.96E+5	**1.89E+5**	1.97E+5
	St. Dev	1.32E+3	4.15E+3	5.80E+1	**2.97E−11**	3.35E+1

intrinsically linked to the quantity of constituent tasks it comprises. The study deliberately varies $|X|$ from 95 (smaller scenarios) to 2559 (larger scenarios), while concurrently investigating the effects of ρ ranging from 16 to 256.

Tables 10 and 11 present the outcomes from using FFT applications with varying ρ values. In all experiments, HEFT (without DVFS technique) consistently consumes more energy. In Table 10, the parameter $|X|$ demonstrates a spectrum of values ranging from 95 to 2559. This underscores the superior energy consumption outcomes achieved by the HWWO algorithm in comparison to other existing algorithms. As the task count rises, both the DECM and REREC algorithms yield comparable performance levels. Notably, up to a task value of 511, the ESRG algorithm stands out for its lower energy consumption compared to EPM. Beyond this threshold, EPM gradually refines its outcomes, albeit at the expense of higher energy usage in contrast to DECM and REREC. The best outcomes, highlighted in bold text, are further illustrated in Fig. 15, which visually represents the data from Table 10, offering a comprehensive comparative analysis.

Table 9
Results of Wilcoxon signed-rank test of Table 8.

Problem	HWWO vs SASS	HWWO vs COLSHADE	HWWO vs EnMODE	HWWO vs sCMAgES
	Rank	Rank	Rank	Rank
C_{opt1}	7	4	3	4
C_{opt2}	2	1	1	1.5
C_{opt3}	6	10	9	9
C_{opt4}	4.5	5	10	3
C_{opt5}	10	7	–	–
C_{opt6}	–	12	–	–
C_{opt7}	4.5	–	6.5	1.5
C_{opt8}	1	2	–	–
C_{opt9}	–	9	8	8
C_{opt10}	3	8	6.5	–
C_{opt11}	8	3	2	10
C_{opt12}	11	6	4.5	5.5
C_{opt13}	13	11	7	
C_{opt14}	9	13	11	5.5
C_{opt15}	12	–	4.5	–
p-value	0.0116	0.0452	0.0370	0.0352
T^+ = Sum of positive number ranks	68.5	85	55	55
T^- = Sum of negative number ranks	22.5	6	11	0
$T = \min(T^+, T^-)$	22.5	6	11	0

Table 10
Energy consumption analysis for FFT parallel applications across task configurations.

$\|X\|$	Performance	Algorithm					
	metric	HEFT	DECM	EPM	REREC	ESRG	HWWO
95	Best	1.48E+4	2.73E+3	6.83E+3	3.07E+3	6.81E+3	**1.68E+3**
	Mean	1.53E+4	2.80E+3	6.90E+3	3.26E+3	6.89E+3	**1.72E+3**
	Worst	1.58E+4	2.83E+3	6.94E+3	3.33E+3	6.94E+3	**1.83E+3**
	St. Dev	3.17E+3	5.78E+1	7.31E+1	4.13E+1	7.23E+1	**3.14E+1**
223	Best	2.53E+4	7.13E+3	8.43E+3	7.44E+3	8.39E+3	**4.11E+3**
	Mean	2.61E+4	7.21E+3	8.47E+3	7.49E+3	8.47E+3	**4.18E+3**
	Worst	2.67E+4	7.33E+3	8.56E+3	7.57E+3	8.54E+3	**4.23E+3**
	St. Dev	6.20E+1	1.28E+1	6.16E+1	2.69E+2	3.80E+3	**1.02E+1**
511	Best	3.64E+4	1.32E+4	2.18E+4	1.27E+4	2.06E+4	**6.58E+3**
	Mean	3.72E+4	1.32E+4	2.27E+4	1.35E+4	2.17E+4	**6.61E+3**
	Worst	3.72E+4	1.32E+4	2.37E+4	1.46E+4	2.29E+4	**6.77E+3**
	St. Dev	1.28E+1	**3.47E−7**	5.34E+1	7.28E+2	4.21E+2	5.54E+1
1151	Best	7.34E+4	3.26E+4	4.73E+4	3.26E+4	4.79E+4	**9.75E+3**
	Mean	7.41E+4	3.35E+4	4.81E+4	3.37E+4	4.88E+4	**9.79E+3**
	Worst	7.49E+4	3.43E+4	4.83E+4	3.43E+4	4.97E+4	**9.79E+3**
	St. Dev	4.13E+4	**2.13E+1**	7.34E+2	5.81E+3	7.03E+2	2.34E+1
2559	Best	9.35E+4	6.17E+4	6.44E+4	6.21E+4	6.71E+4	**4.73E+4**
	Mean	9.43E+4	6.28E+4	6.51E+4	6.37E+4	6.82E+4	**4.86E+4**
	Worst	9.51E+4	6.39E+4	6.59E+4	6.47E+4	6.88E+4	**4.93E+4**
	St. Dev	5.82E+4	3.72E+2	**1.80E+2**	4.63E+2	**1.80E+2**	7.37E+2

Within Table 11, it is notable that the EPM algorithm requires significantly more CT_{TA}. However, as the number of tasks increases, ESRG surpasses the other three algorithms in producing higher energy values. The CT_{TA} of the newly proposed HWWO algorithm is projected to occupy between 31.20% and 35.61% of the computational time required by the DECM and REREC algorithms. Regarding performance metrics, across a spectrum of values for $|X|$ from 95 to 2559, the CT_{TA} of the HWWO closely mirrors that of the DECM algorithm, consistently surpassing it.

Experiment 2: The study evaluates the reliability metrics and total energy consumption of an extensive FFT application under varying reliability constraints. The experimental configuration involves 1151 tasks, with $\rho = 128$. Additionally, the reliability goal, $R_{e(goal)}(G)$, is systematically varied from 0.91 to 0.99 in increments of 0.01, enabling an assessment of the corresponding effects on reliability performance and energy utilization.

The graphical representation in Fig. 16 illustrates the actual reliability values attained by the large-scale FFT application when subjected to varying reliability criteria. Among the techniques evaluated, the HWWO algorithm demonstrates superior performance in accomplishing the specified reliability goals compared to other existing methods. Contrastingly, the HEFT, EPM, and ESRG algorithms exhibit an inability to fulfill the reliability constraints in the majority of scenarios. As the reliability objective escalates from 0.91 to 0.95, HEFT, EPM, and ESRG manage to comply with the requirements, but struggle beyond that range. Conversely, DECM, REREC, and HWWO successfully meet the reliability constraint within the range of 0.91 to 0.98, although none of the algorithms can fulfill the rigorous 0.99 requirement. It is noteworthy that if the upper bound for the reliability objective is established at an excessively elevated level, the maximum attainable reliability values for partial tasks may fall short of this upper bound in practical implementation scenarios.

Data presented in Table 12 has been shown graphically in Fig. 17. The table evaluates the energy consumption profiles of FFT applications when subjected to varying reliability criteria. For reliability thresholds up to 0.98, the techniques DECM, REREC, and HWWO exhibit superior energy consumption performance in comparison to HEFT, EPM, and ESRG. The algorithms HEFT, EPM, and ESRG are capable of producing energy outcomes only up to a reliability criterion of 0.95, as they fail to meet the reliability constraints beyond this point, as evidenced by the findings illustrated in Fig. 16. Until the 0.98 reliability threshold,

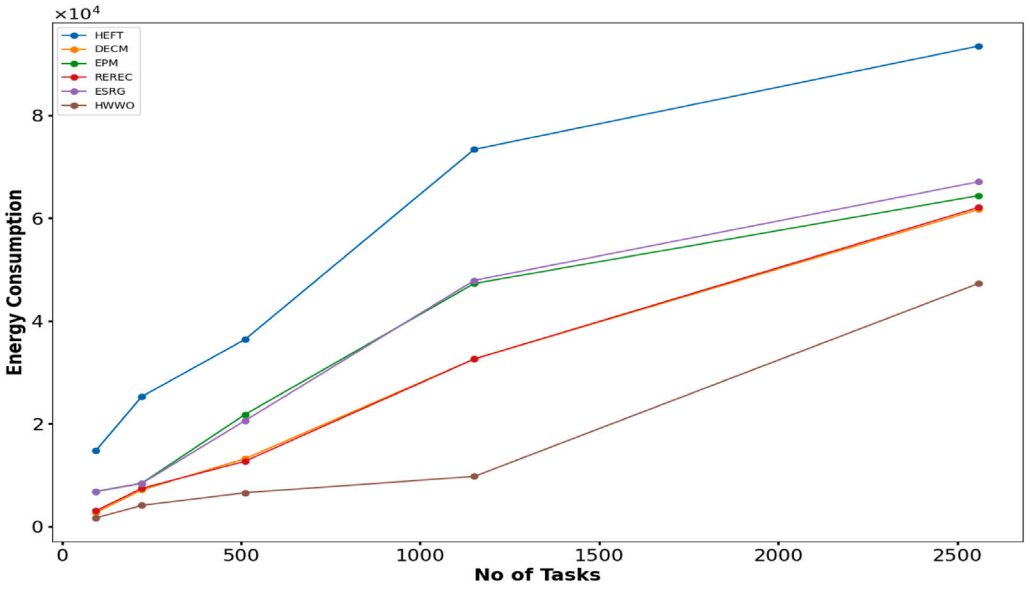

Fig. 15. Graphical representation of Table 10.

Table 11
CT_{TA} values of FFT applications across diverse task quantities.

$\|X\|$	Performance metric	Algorithm				
		DECM	EPM	REREC	ESRG	HWWO
95	Best	1.5E+1	2.57E+2	1.9E+1	2.31E+2	**1.2E+1**
	Mean	2.62E+1	2.73E+2	2.6E+1	2.66E+2	**2.2E+1**
	Worst	2.80E+1	3.27E+2	2.91E+1	3.25E+2	**2.6E+1**
	St. Dev	3.15E+1	1.19E+2	5.4E+1	2.4E+2	**4.8E+0**
223	Best	**2.1E+1**	5.04E+2	3.9E+1	4.6E+2	**2.1E+1**
	Mean	**2.7E+1**	5.73E+2	4.3E+1	4.8E+2	**2.7E+1**
	Worst	3.8E+1	5.91E+2	4.3E+1	5.3E+2	**3.1E+1**
	St. Dev	**3.42E−1**	4.21E+2	2.06E+0	3.07E+2	3.42E+0
511	Best	5.6E+1	3.5E+3	6.3E+1	3.2E+3	**4.7E+1**
	Mean	5.81E+1	3.6E+3	6.8E+1	3.4E+3	**4.7E+1**
	Worst	6.04E+1	3.6E+3	7.3E+1	3.5E+3	**4.92E+1**
	St. Dev	3.66E+1	**3.01E−1**	7.2E+1	6.7E+0	8.9E−1
1151	Best	2.62E+2	4.16E+3	3.31E+2	4.73E+3	**2.49E+2**
	Mean	2.74E+2	4.16E+3	3.31E+2	4.73E+3	**2.51E+2**
	Worst	2.74E+2	4.47E+3	3.47E+2	4.73E+3	**2.51E+2**
	St. Dev	3.31E−1	1.71E+3	**1.08E−1**	3.26E−1	3.31E−1
2559	Best	4.7E+2	8.33E+3	5.4E+2	8.72E+3	**3.48E+2**
	Mean	4.8E+2	8.42E+3	5.6E+2	8.74E+3	**3.71E+2**
	Worst	4.8E+2	8.46E+3	5.6E+2	8.83E+3	**3.71E+2**
	St. Dev	**1.07E+1**	3.72E+1	**1.07E+1**	6.01E+2	7.06E+2

DECM, REREC, and HWWO successfully fulfill the reliability constraints in the majority of scenarios. Notably, among these three techniques, the HWWO algorithm demonstrates more favorable results by further optimizing energy consumption through an expanded exploration of processor and frequency combination possibilities. HWWO surpasses DECM and REREC in energy savings, reducing consumption by 33% and 36% on average, correspondingly. However, it is pertinent to note that none of the algorithms evaluated can achieve the stringent 0.99 reliability requirement.

Experiment 3: The current experiment examines the SLR and CCR metrics for a comprehensive FFT application, considering variations in ρ. This experimental approach incorporates a stringent deadline requirement, expressed as $DL(G) = LB(G) * 1.4$, where the complexity of the parallel application is inherently tied to the number of constituent tasks it encompasses. The study systematically alters $|X|$ from 95 (representing smaller-scale scenarios) to 2559 (representing larger-scale scenarios), while simultaneously exploring the impacts of ρ ranging from 16 to 256.

The scheduling length ratio (SLR) is a widely adopted metric employed for evaluating and contrasting various scheduling algorithms. It is quantified as the ratio of the makespan to the cumulative sum of the minimum execution times of all tasks residing on the critical path of the DAG [86]. This can be expressed through the following mathematical formulation:

$$SLR = \frac{MS(G)}{\sum_{\tau_i \in CP_{MIN}} min_{Y_l \in Y}(\hat{w}_{i,l})} \tag{44}$$

The data presented in Table 13 and visually represented in Fig. 18 demonstrates the average performance of various tasks scheduling algorithms in terms of the SLR metric. The proposed HWWO algorithm exhibited the lowest SLR values across all experiments, outperforming the other techniques evaluated. Concerning SLR, HWWO established itself as the superior approach. Across all task sizes, the HEFT algorithm

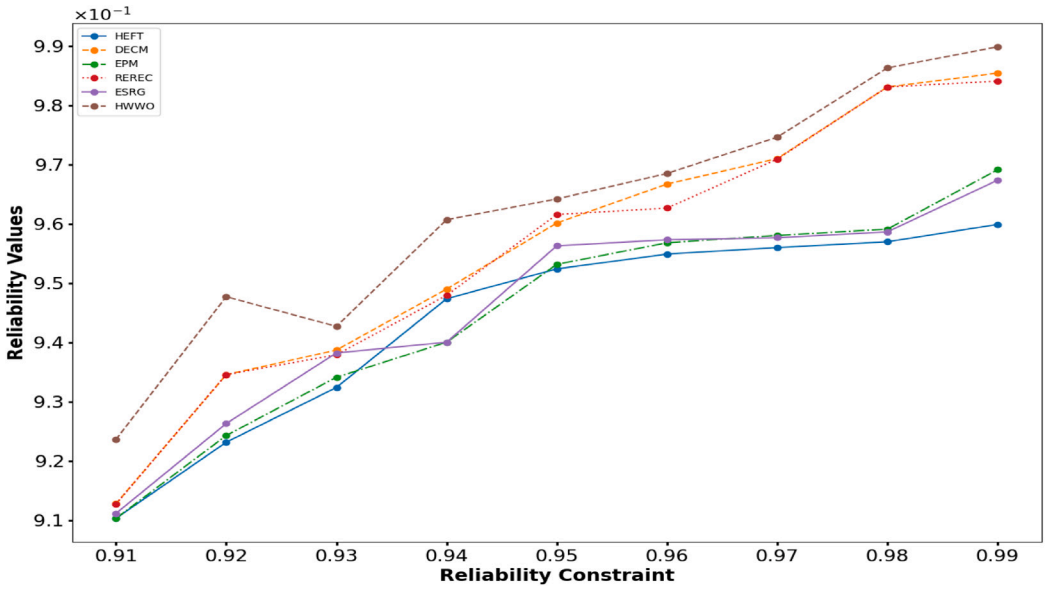

Fig. 16. Graphical representation of actual reliability values under varying reliability constraints (in case of FFT).

Table 12
Energy consumption assessment for FFT applications with varying reliability criteria.

Reliability goal	Performance metric	Algorithm					
		HEFT	DECM	EPM	REREC	ESRG	HWWO
0.91	Best	4.44E+4	2.36E+4	3.73E+4	2.46E+4	3.53E+4	**1.25E+4**
	Mean	4.51E+4	2.45E+4	3.81E+4	2.47E+4	3.71E+4	**1.29E+4**
	Worst	4.58E+4	2.47E+4	3.83E+4	2.53E+4	3.83E+4	**1.32E+4**
	St. Dev	4.23E+1	4.13E−2	5.24E+2	5.81E−2	6.34E+2	**5.24E−4**
0.92	Best	4.74E+4	2.43E+4	3.89E+4	2.54E+4	3.83E+4	**1.71E+4**
	Mean	4.81E+4	2.47E+4	3.91E+4	2.54E+4	3.86E+4	**1.78E+4**
	Worst	4.92E+4	2.47E+4	3.91E+4	2.54E+4	3.86E+4	**1.78E+4**
	St. Dev	7.32E+4	**6.28E−11**	7.04E−4	**6.28E−11**	5.34E−4	4.02E−7
0.93	Best	5.64E+4	3.71E+4	4.28E+4	3.71E+4	4.06E+4	**2.71E+4**
	Mean	5.72E+4	3.78E+4	4.33E+4	3.85E+4	4.37E+4	**2.88E+4**
	Worst	5.82E+4	3.88E+4	4.47E+4	3.92E+4	4.49E+4	**2.94E+4**
	St. Dev	3.28E+4	7.12E+4	**2.02E+4**	5.08E+4	**2.02E+4**	**2.02E+4**
0.94	Best	6.74E+4	3.87E+4	4.89E+4	3.81E+4	4.82E+4	**2.87E+4**
	Mean	6.74E+4	3.93E+4	4.93E+4	3.95E+4	4.89E+4	**2.97E+4**
	Worst	6.74E+4	3.99E+4	4.97E+4	3.99E+4	4.97E+4	**2.99E+4**
	St. Dev	3.28E+4	**1.12E+4**	7.02E+4	**1.12E+4**	5.52E+4	6.42E+4
0.95	Best	8.45E+4	6.17E+4	6.84E+4	6.37E+4	6.71E+4	**4.24E+4**
	Mean	8.48E+4	6.28E+4	6.88E+4	6.37E+4	6.82E+4	**4.46E+4**
	Worst	8.51E+4	6.39E+4	6.95E+4	6.37E+4	6.88E+4	**4.53E+4**
	St. Dev	5.82E+4	3.72E+2	1.80E+4	**4.63E−8**	8.75E+4	7.37E+4
0.96	Best	–	6.66E+4	–	6.71E+4	–	**4.51E+4**
	Mean	–	6.78E+4	–	6.87E+4	–	**4.51E+4**
	Worst	–	6.78E+4	–	6.91E+4	–	**4.63E+4**
	St. Dev	–	3.72E+4	–	4.63E−4	–	**1.17E−7**
0.97	Best	–	7.05E+4	–	7.21E+4	–	**6.19E+4**
	Mean	–	7.08E+4	–	7.21E+4	–	**6.19E+4**
	Worst	–	7.09E+4	–	7.21E+4	–	**6.19E+4**
	St. Dev	–	3.13E+3	–	**3.33E−11**	–	**3.33E−11**
0.98	Best	–	8.29E+4	–	8.41E+4	–	**7.39E+4**
	Mean	–	8.31E+4	–	8.48E+4	–	**7.57E+4**
	Worst	–	8.34E+4	–	8.48E+4	–	**7.61E+4**
	St. Dev	–	6.37E+2	–	**1.17E+2**	–	4.52E+3
0.99	Best	–	–	–	–	–	–
	Mean	–	–	–	–	–	–
	Worst	–	–	–	–	–	–
	St. Dev	–	–	–	–	–	–

consistently generated the poorest schedules, trailing behind EPM and ESRG. Initially, EPM underperformed compared to ESRG, but as the number of tasks increased, its performance surpassed that of ESRG. Notably, in scenarios where every path within the DAG constituted a critical path, the DECM and REREC algorithms achieved comparable and superior results to HEFT, EPM, and ESRG, regardless of the input size. The average SLR performance of HWWO across all generated graphs exceeded that of the DECM algorithm by 15% and the REREC algorithm by 20.98%.

The communication to computation ratio (CCR) is a metric that quantifies the relative significance of communication overhead by dividing the cumulative communication times across all edges by the

Table 13
Average SLR for all algorithms with respect to various tasks (in case of FFT).

$\|X\|$	Average SLR					
	HEFT	DECM	EPM	REREC	ESRG	HWWO
95	5.1E+1	3.4E+1	4.2E+1	3.7E+1	4.0E+1	**2.4E+1**
223	6.2E+1	4.6E+1	4.9E+1	4.6E+1	4.9E+1	**3.8E+1**
511	8.8E+1	5.2E+1	6.5E+1	5.8E+1	6.2E+1	**4.7E+1**
1151	1.1E+2	7.8E+1	8.8E+1	8.5E+1	9.2E+1	**6.9E+1**
2559	1.3E+2	8.3E+1	9.2E+1	8.9E+1	9.6E+1	**7.6E+1**

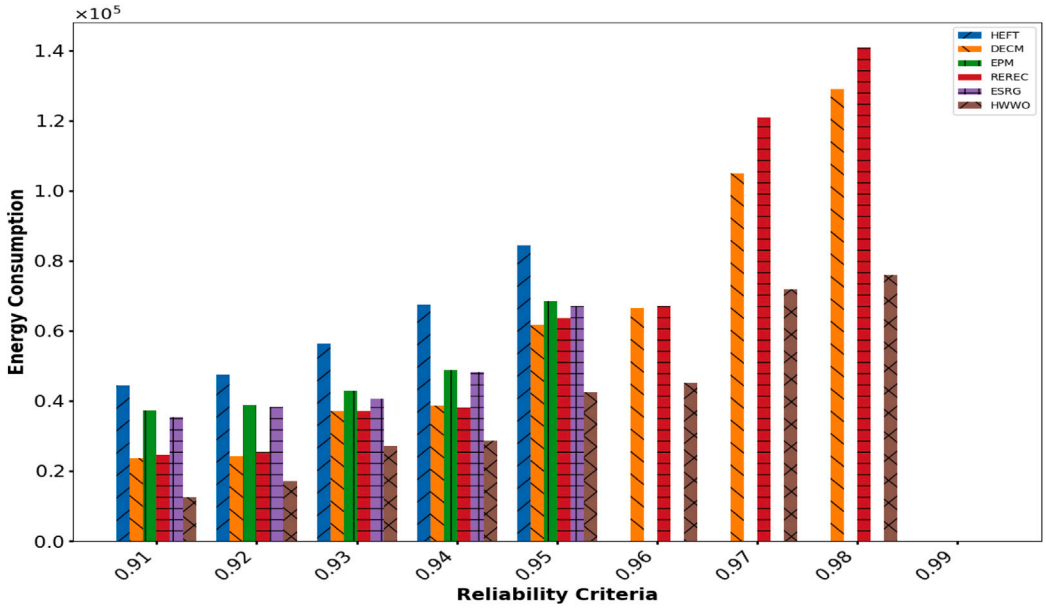

Fig. 17. Graphical representation of Table 12.

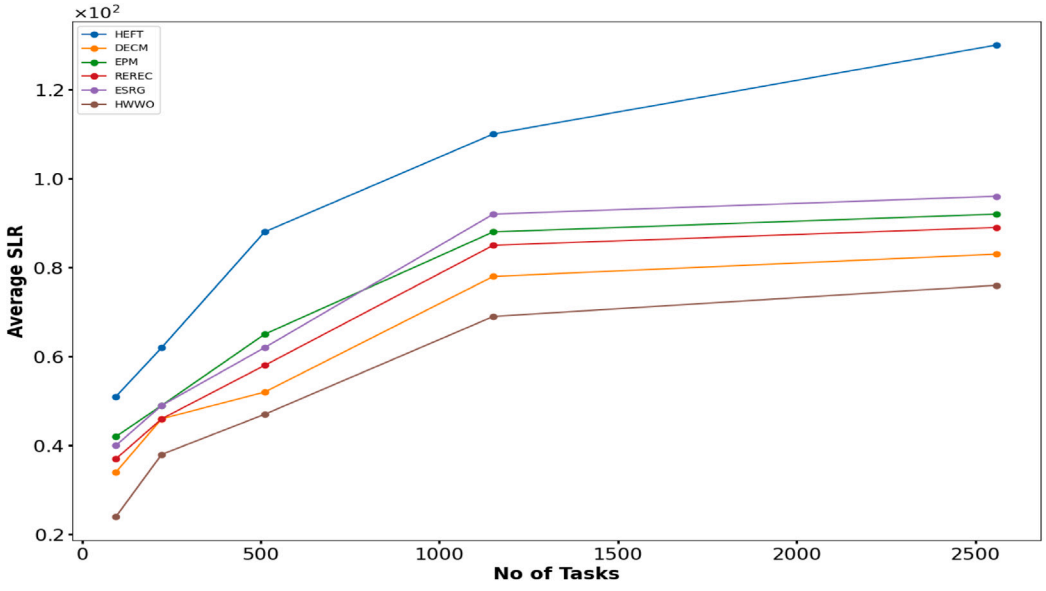

Fig. 18. Graphical representation of Table 13.

total execution times across all nodes in a DAG. Fig. 19 illustrates the average SLR performance of various algorithms as a function of the CCR. When considering CCR values, HEFT consistently exhibited the poorest SLR results, surpassed by both EPM and ESRG, while DECM, REREC, and HWWO demonstrated their ability to generate superior schedules compared to these algorithms. The schedules produced by DECM and REREC are comparable across different CCR values. However, HWWO emerged as the top-performing algorithm, yielding the best SLR outcomes for all CCR values considered. Notably, the average SLR performance of HWWO across all generated graphs surpassed that of DECM by 14.16% and REREC by 19.91%.

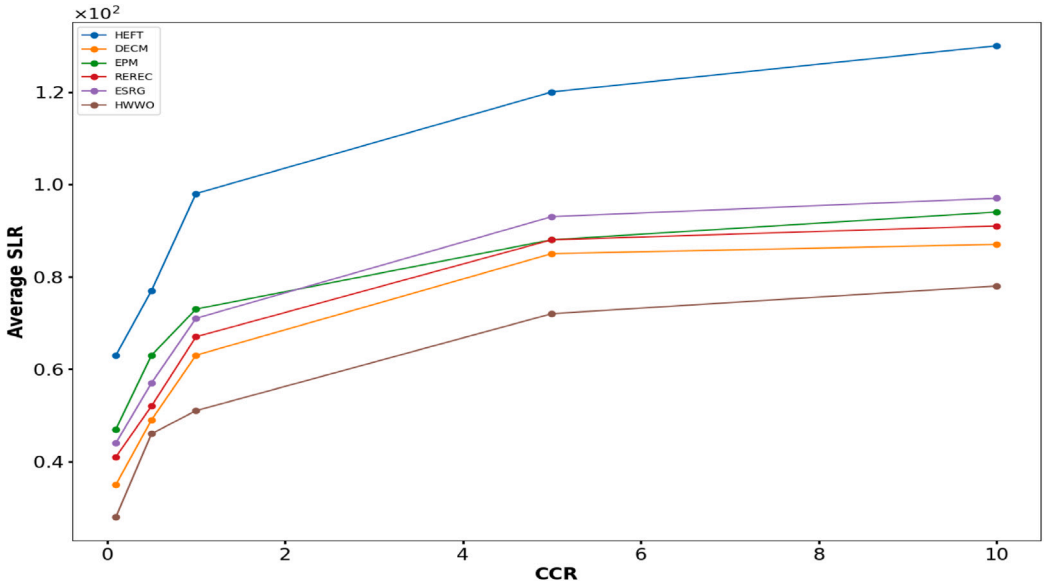

Fig. 19. Graphical representation of SLR values with respect to CCR (in case of FFT).

6.2.3. Scenario 3

To rigorously assess the proposed technique's effectiveness, the study conducts a comprehensive evaluation of the HWWO algorithm's performance through an analysis of the gaussian elimination (GE) problem. The visual representation in Fig. 20 depicts a parallel implementation of the GE application, incorporating a critical parameter value of $\rho = 5$, as described in [8,86]. Notably, the total number of tasks, denoted as $|X|$, is dynamically determined by the expression $|X| = \frac{\rho^2+\rho-2}{2}$. Specifically, when $\rho = 5$, the resulting task count is $|X| = 14$, as illustrated in Fig. 20. This scenario highlights the intricate interplay between the parameters ρ and $|X|$ in the context of parallel computing applications. The following three experiments are conducted to evaluate the performance of the proposed approach using the GE application as a benchmark.

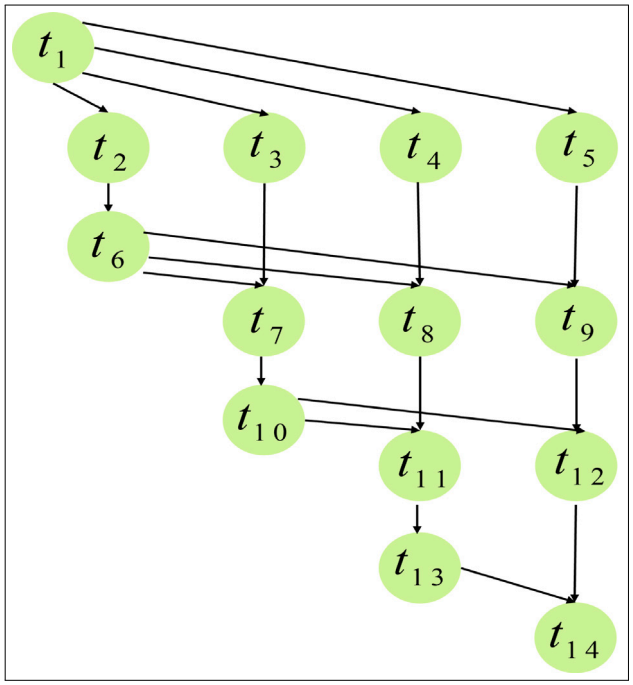

Fig. 20. DAG of GE application with $\rho = 5$.

Experiment 4: The overarching goal of this experimental endeavor is to conduct a meticulous comparative assessment of the proposed HWWO technique against existing algorithms. The primary goal is placed on evaluating their respective performances concerning total energy consumption and computational time requirements within the realm of parallel applications involving GE. Underpinning this experimental study is a stringent deadline and reliability constraints, formulated as $DL(G) = LB(G) * 1.4$ and $R_{e(goal)}(G) = 0.90$ respectively, where the complexity of the parallel application is intrinsically linked to the quantity of constituent tasks it comprises. To comprehensively analyze the techniques' behavior, the study methodically varies the task count $|X|$ over a substantial range, spanning from 90 tasks (representing smaller-scale scenarios) to 2555 tasks (larger-scale scenarios), while concurrently investigating the effects of ρ ranging from 13 to 71.

Tables 14 and 15 present the outcomes from using GE applications with varying ρ values. In all experiments, HEFT (without DVFS technique) consistently consumes more energy. Table 14 shows that the parameter $|X|$ has a wide range of values from 90 to 2555. This highlights the superior energy efficiency of the HWWO algorithm compared to other existing algorithms. As the number of tasks increases, both the DECM and REREC algorithms exhibit similar performance levels. Notably, the ESRG algorithm outperforms EPM in terms of lower energy consumption up to 495 tasks. Beyond that, EPM gradually improves its results but at the cost of higher energy usage compared to DECM and REREC. The best outcomes, highlighted in bold, are further illustrated in Fig. 21, providing a visual comparative analysis of the data from Table 14.

Table 15 shows that the EPM algorithm requires significantly higher CT_{TA}. However, as the number of tasks increases, ESRG outperforms the other three algorithms in terms of higher energy values. The proposed HWWO algorithm's CT_{TA} is projected to be 33.4% to 37.39% of the computational time required by DECM and REREC algorithms. In terms of performance metrics, for $|X|$ values ranging from 90 to 2555, the HWWO algorithm's CT_{TA} closely follows and consistently outperforms the DECM algorithm.

Table 14
Energy consumption analysis for GE parallel applications across task configurations.

$\|X\|$	Performance metric	Algorithm					
		HEFT	DECM	EPM	REREC	ESRG	HWWO
90	Best	3.18E+4	3.36E+3	7.88E+3	3.77E+3	7.81E+3	**2.78E+3**
	Mean	3.33E+4	3.43E+3	7.94E+3	3.86E+3	7.89E+3	**2.81E+3**
	Worst	3.38E+4	3.43E+3	7.94E+3	3.93E+3	7.94E+3	**2.88E+3**
	St. Dev	5.04E+1	**1.42E+0**	3.31E+1	4.13E+1	2.13E+1	5.07E+1
230	Best	4.73E+4	7.93E+3	8.63E+3	7.93E+3	8.49E+3	**4.71E+3**
	Mean	4.81E+4	7.93E+3	8.77E+3	7.93E+3	8.53E+3	**4.79E+3**
	Worst	4.81E+4	7.93E+3	8.78E+3	7.93E+3	8.59E+3	**4.83E+3**
	St. Dev	6.20E+4	**2.32E−7**	3.26E+1	**2.32E−7**	3.80E+1	2.32E+1
495	Best	6.27E+4	2.62E+4	3.08E+4	2.67E+4	3.05E+4	**9.68E+3**
	Mean	6.27E+4	2.62E+4	3.16E+4	2.73E+4	3.11E+4	**9.68E+3**
	Worst	6.27E+4	2.62E+4	3.16E+4	2.83E+4	3.16E+4	**9.72E+3**
	St. Dev	8.08E−4	**3.47E−7**	2.34E+2	1.28E+3	4.44E+1	5.54E−3
1127	Best	9.44E+4	6.05E+4	7.03E+4	6.35E+4	7.59E+4	**4.25E+4**
	Mean	9.44E+4	6.05E+4	7.13E+4	6.35E+4	7.62E+4	**4.25E+4**
	Worst	9.49E+4	6.11E+4	7.13E+4	6.41E+4	7.67E+4	**4.25E+4**
	St. Dev	1.13E+2	1.25E+4	2.14E+4	1.15E+4	5.03E+4	**7.34E−9**
2555	Best	9.75E+4	7.65E+4	9.44E+4	7.97E+4	9.54E+4	**7.13E+4**
	Mean	9.82E+4	7.67E+4	9.51E+4	7.97E+4	9.54E+4	**7.17E+4**
	Worst	9.88E+4	7.67E+4	9.59E+4	7.97E+4	9.59E+4	**7.17E+4**
	St. Dev	1.02E+3	2.37E−1	4.80E+4	**5.87E−7**	3.80E+1	2.37E+2

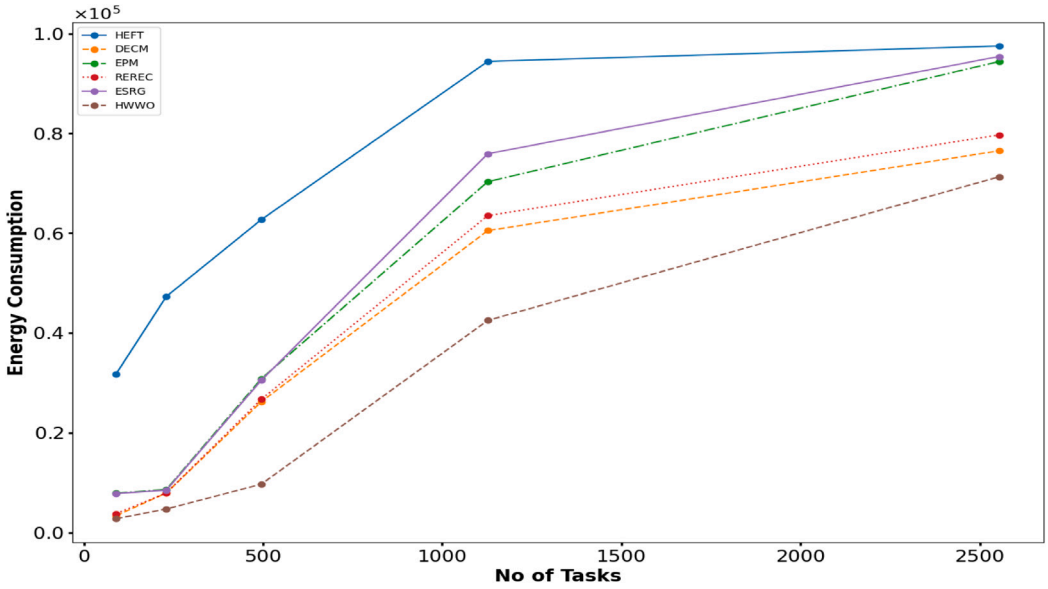

Fig. 21. Graphical representation of Table 14.

Experiment 5: This experiment evaluates the reliability metrics and total energy consumption of an extensive GE application under varying reliability constraints. The experimental configuration involves 1127 tasks, with $\rho = 47$. Additionally, the reliability goal, $R_{e(goal)}(G)$, is systematically varied from 0.91 to 0.99 in increments of 0.01, enabling an assessment of the corresponding effects on reliability performance and energy utilization.

The graphical representation in Fig. 22 illustrates the actual reliability values attained by the GE application when subjected to varying reliability criteria. Among the techniques evaluated, the HWWO algorithm demonstrates superior performance in accomplishing the specified reliability goals compared to other existing methods. In contrast, the HEFT, EPM, and ESRG algorithms struggle to meet the reliability constraints in most scenarios. While they can comply when $R_{e(goal)}(G)$ is between 0.91 and 0.95, their performance deteriorates beyond that range. On the other hand, DECM, REREC, and HWWO successfully satisfy the reliability constraint from 0.91 to 0.98, but none of the algorithms can meet the stringent 0.99 requirement.

Fig. 23 visually represents the data from Table 16, showing the energy consumption of GE applications under different reliability constraints. Up to 0.98 reliability, DECM, REREC, and HWWO are more energy-efficient than HEFT, EPM, and ESRG. While HEFT, EPM, and ESRG fail to meet reliability constraints beyond 0.95, as evident from Fig. 22, DECM, REREC, and HWWO successfully fulfill the reliability requirements up to 0.98 in most cases. Among them, HWWO achieves better energy savings by exploring more processor and frequency combinations, reducing consumption by 33% and 37% compared to DECM and REREC, respectively. However, none of the algorithms meet the stringent 0.99 reliability requirement.

Experiment 6: The current experiment examines the SLR and CCR metrics for a comprehensive GE application, considering variations in ρ. This experimental approach incorporates a stringent deadline

Table 15
CT_{TA} values of GE applications across diverse task quantities.

$\|X\|$	Performance metric	Algorithm				
		DECM	EPM	REREC	ESRG	HWWO
90	Best	3.5E+1	4.31E+2	3.5E+1	3.81E+2	**2.6E+1**
	Mean	3.6E+1	4.43E+2	3.6E+1	3.91E+2	**2.6E+1**
	Worst	3.8E+1	4.43E+2	3.8E+1	3.95E+2	**2.6E+1**
	St. Dev	7.5E−1	5.6E+0	7.5E−1	5.8E+0	**0**
230	Best	5.1E+1	7.14E+2	5.1E+1	6.84E+2	**3.9E+1**
	Mean	5.9E+1	7.23E+2	6.7E+1	6.93E+2	**4.4E+1**
	Worst	6.8E+1	7.41E+2	6.7E+1	6.94E+2	**4.9E+1**
	St. Dev	6.4E+0	1.21E+2	7.4E+0	**3.20E−1**	4.02E−1
495	Best	**6.4E+1**	9.5E+3	7.1E+1	8.5E+3	**6.4E+1**
	Mean	**6.9E+1**	1.6E+4	8.3E+1	1.3E+4	**6.9E+1**
	Worst	**7.2E+1**	2.2E+4	9.2E+1	2.7E+4	**7.2E+1**
	St. Dev	**3.3E+0**	5.10E+3	8.3E+1	2.09E+2	**3.3E+0**
1127	Best	7.7E+2	8.19E+3	7.9E+2	8.73E+3	**7.49E+2**
	Mean	7.88E+2	8.25E+3	8.15E+2	8.83E+3	**7.67E+2**
	Worst	7.94E+2	8.37E+3	8.7E+2	8.87E+3	**7.77E+2**
	St. Dev	4.31E+1	2.71E+3	7.08E+1	**5.08E+0**	1.1E+1
2555	Best	**8.57E+2**	9.03E+3	8.77E+2	9.12E+3	**8.57E+2**
	Mean	**8.61E+2**	9.22E+3	8.81E+2	9.28E+3	**8.61E+2**
	Worst	**8.69E+2**	9.36E+3	8.91E+2	9.39E+3	**8.69E+2**
	St. Dev	4.9E−2	**3.32E+1**	6.07E+1	6.17E+1	4.9E+1

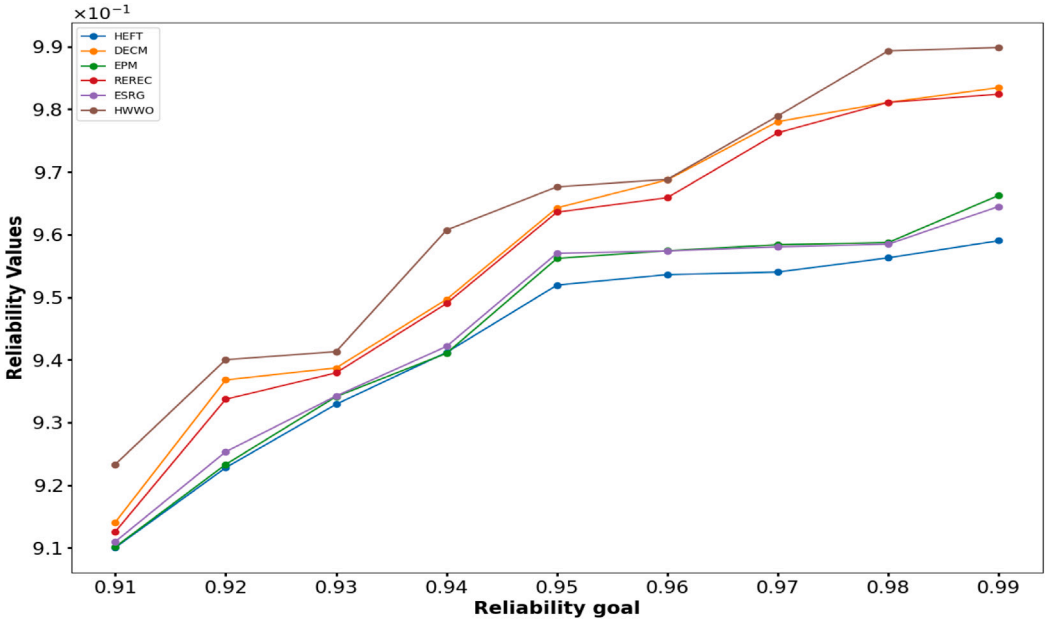

Fig. 22. Graphical representation of actual reliability values under varying reliability constraints (in case of GE).

constraint, formulated as $DL(G) = LB(G) * 1.4$, where the complexity of the parallel application is inherently tied to the number of constituent tasks it encompasses. The study systematically alters $|X|$ from 90 (representing smaller-scale scenarios) to 2555 (representing larger-scale scenarios), while simultaneously exploring the impacts of ρ ranging from 13 to 71.

Table 17 and Fig. 24 demonstrate the average SLR performance of various tasks scheduling algorithms. The proposed HWWO algorithm exhibited the lowest SLR values, outperforming others. HWWO emerged as the superior approach in terms of SLR. Across all task sizes, HEFT consistently generated the poorest schedules, trailing EPM and ESRG. Initially, EPM underperformed compared to ESRG but surpassed it as task count increased. In critical DAG path scenarios, DECM and REREC outperformed HEFT, EPM, and ESRG, regardless of input size.

Fig. 25 shows the average SLR performance of various algorithms as a function of CCR. HEFT consistently exhibited the poorest SLR results, while DECM, REREC, and HWWO generated superior schedules compared to HEFT, EPM, and ESRG. DECM and REREC produced comparable schedules across different CCR values. However, HWWO outperformed all others, yielding the best SLR outcomes for all considered CCR values.

Stage II

6.3. Benchmark analysis with metaheuristic algorithms

In this stage the proposed algorithm HWWO performance is evaluated across three scenarios and compared with several metaheuristic methods using various metrics. The evaluation considers different task and processor configurations, as well as benchmark tests involving unimodal functions. Additionally, experiments are conducted for tasks scheduling in a multiprocessing environment, with input parameters described in Table 18. After simulations, a comprehensive assessment

Table 16
Energy consumption assessment for GE applications with varying reliability criteria.

Reliability goal	Performance metric	Algorithm					
		HEFT	DECM	EPM	REREC	ESRG	HWWO
0.91	Best	4.54E+4	3.45E+4	3.83E+4	3.53E+4	3.73E+4	**2.55E+4**
	Mean	4.54E+4	3.45E+4	3.83E+4	3.53E+4	3.73E+4	**2.55E+4**
	Worst	4.58E+4	3.45E+4	3.83E+4	3.53E+4	3.79E+4	**2.55E+4**
	St. Dev	4.23E+2	4.13E−4	5.24E−4	1.13E−4	6.34E−2	**5.24E−8**
0.92	Best	4.74E+4	3.77E+4	4.18E+4	3.96E+4	4.16E+4	**2.88E+4**
	Mean	4.81E+4	3.78E+4	4.19E+4	3.98E+4	4.19E+4	**2.88E+4**
	Worst	4.92E+4	3.78E+4	4.19E+4	3.98E+4	4.19E+4	**2.88E+4**
	St. Dev	7.40E+2	4.27E−1	7.04E+1	9.42E+1	2.04E+2	**7.02E−7**
0.93	Best	5.74E+4	3.81E+4	4.28E+4	4.18E+4	4.28E+4	**2.91E+4**
	Mean	5.74E+4	3.83E+4	4.33E+4	4.20E+4	4.33E+4	**2.94E+4**
	Worst	5.74E+4	3.86E+4	4.47E+4	4.26E+4	4.47E+4	**2.95E+4**
	St. Dev	**3.28E−4**	4.40E+3	8.02E+2	3.30E+3	8.02E+2	1.69E+2
0.94	Best	6.64E+4	**3.97E+4**	4.93E+4	4.51E+4	4.97E+4	**3.97E+4**
	Mean	6.69E+4	**3.97E+4**	4.93E+4	4.65E+4	4.98E+4	**3.97E+4**
	Worst	6.74E+4	**3.99E+4**	4.93E+4	4.77E+4	4.98E+4	**3.99E+4**
	St. Dev	4.08E+2	**9.42E+1**	7.12E+2	3.12E+2	4.13E+2	**9.42E+1**
0.95	Best	8.45E+4	6.52E+4	6.84E+4	6.63E+4	6.84E+4	**4.44E+4**
	Mean	8.48E+4	6.52E+4	6.84E+4	6.67E+4	6.84E+4	**4.46E+4**
	Worst	8.51E+4	6.59E+4	6.85E+4	6.67E+4	6.85E+4	**4.53E+4**
	St. Dev	5.02E+3	**3.72E−4**	4.07E+3	4.63E+3	4.07E+3	5.17E+2
0.96	Best	–	6.76E+4	–	6.90E+4	–	**4.86E+4**
	Mean	–	6.78E+4	–	6.90E+4	–	**4.86E+4**
	Worst	–	6.78E+4	–	6.90E+4	–	**4.86E+4**
	St. Dev	–	3.72E−2	–	4.63E−5	–	**4.06E−13**
0.97	Best	–	7.15E+4	–	7.25E+4	–	**6.55E+4**
	Mean	–	7.18E+4	–	7.25E+4	–	**6.55E+4**
	Worst	–	7.18E+4	–	7.25E+4	–	**6.55E+4**
	St. Dev	–	4.03E+2	–	5.33E−11	–	**4.22E−11**
0.98	Best	–	9.37E+4	–	9.41E+4	–	**7.31E+4**
	Mean	–	9.37E+4	–	9.41E+4	–	**7.31E+4**
	Worst	–	9.37E+4	–	9.48E+4	–	**7.37E+4**
	St. Dev	–	**4.37E−9**	–	1.17E+2	–	4.52E−4
0.99	Best	–	–	–	–	–	–
	Mean	–	–	–	–	–	–
	Worst	–	–	–	–	–	–
	St. Dev	–	–	–	–	–	–

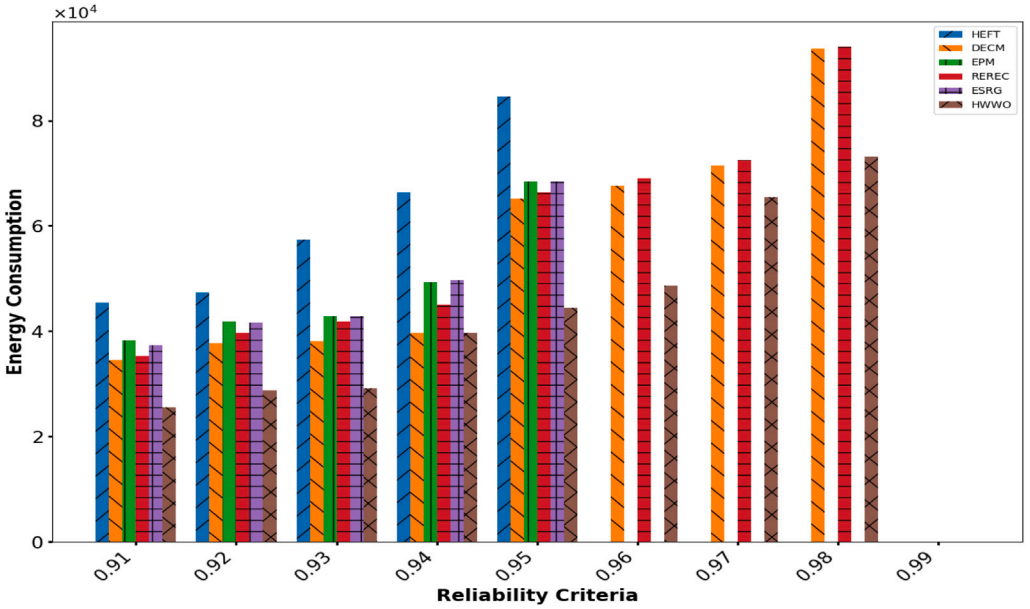

Fig. 23. Graphical representation of Table 16.

is carried out, calculating metrics such as average execution time, standard deviation, and mean across iterations. The results highlight the algorithm's effectiveness in terms of energy consumption, system reliability, resource utilization, and sensitivity analysis.

Table 17
Average SLR for all algorithms with respect to various tasks (in case of GE).

$\|X\|$	Average SLR					
	HEFT	DECM	EPM	REREC	ESRG	HWWO
90	6.3E+1	3.4E+1	4.4E+1	3.9E+1	4.0E+1	**3.4E+1**
230	7.2E+1	4.7E+1	5.8E+1	4.9E+1	5.4E+1	**3.8E+1**
495	8.9E+1	5.8E+1	6.6E+1	6.1E+1	6.2E+1	**4.9E+1**
1127	1.2E+2	7.8E+1	9.3E+1	8.5E+1	9.8E+1	**7.2E+1**
2555	1.3E+2	8.7E+1	9.5E+1	8.9E+1	9.8E+1	**7.9E+1**

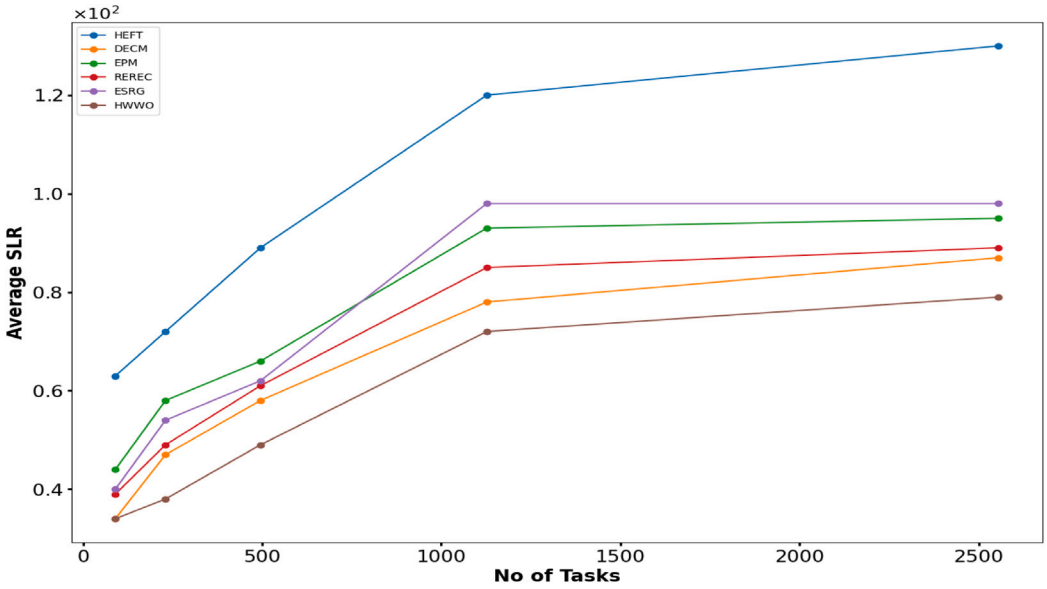

Fig. 24. Graphical representation of Table 17.

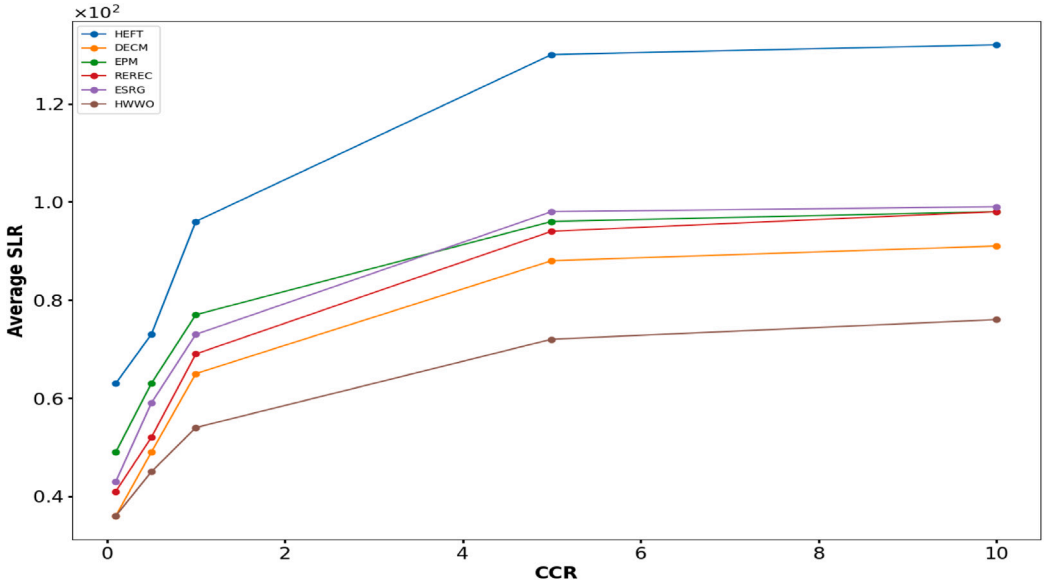

Fig. 25. Graphical representation of SLR values with respect to CCR (in case of GE).

6.3.1. Scenario 1

The study aims to thoroughly evaluate the effectiveness of the proposed HWWO-based approach by testing it with different numbers of tasks. Seven well-known metaheuristic algorithms – PSO, ACO, KH, DA, AHA, GWO, and WOA – are employed alongside the HWWO algorithm. These algorithms are utilized to assess the performance of the HWWO technique when varying the number of tasks while keeping the number of processors constant. The detailed findings and analysis are presented subsequently.

Tasks range:	100–1000
Processor count:	100

For an impartial and consistent evaluation, the parameter settings of the seven algorithms remained unchanged from their default values

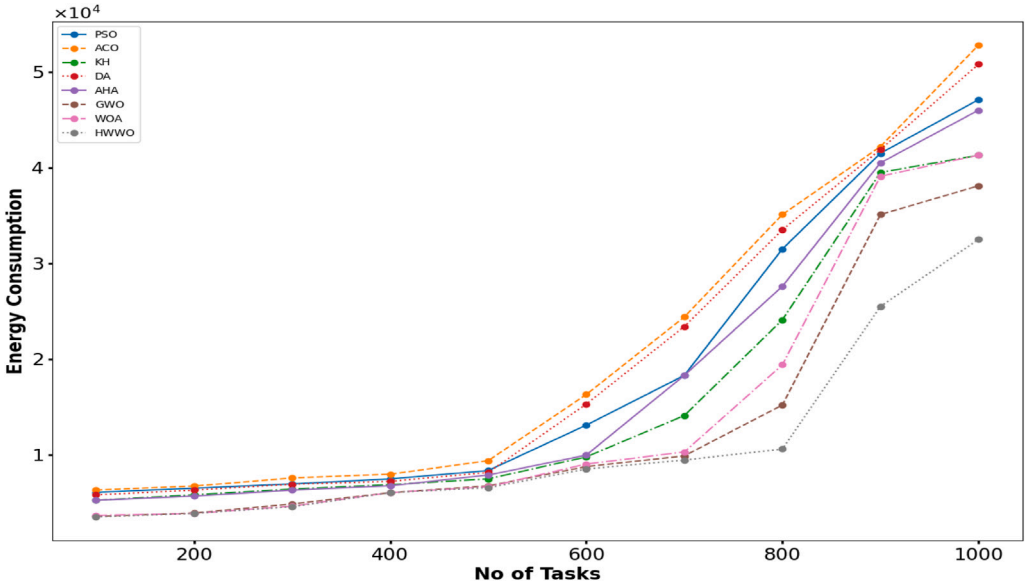

Fig. 26. Graphical representation of Table 19.

Table 18
Parameter setting for stage II.

Algorithm	Parameter
PSO [57]	Inertia weight = 0.4-0.9
	Cognitive component = 1.50
	Social component= 2
ACO [55]	Evaporation rate = 0.2
	Weight of pheromone on decision = 0.5
	Weight of heuristic information on decision = 0.5
	Q = 2
KH [59]	Foraging motion = 0.2
	Induced motion = 0.006
	Inertia weight = 0.5
DA [67]	Separation weight = 0.12
	Alignment weight = 0.12
	Cohesion weight = 0.75
	Food factor = 1
	Enemy factor =1
AHA [67]	Inertia weight = 0.5
	Local search probability = 0.5
GWO [29]	$\vec{\mu}$ = [0, 2]
	l_1, l_2 = [0,1]
	$\vec{\zeta}$ = [−1, 1]
WOA [28]	ν = [−1, 1]
	$\vec{\mu}$ = [0, 2]
	l_1, l_2 = [0,1]
Proposed HWWO	$\hat{u}_{i,l}(ms)$ = [10, 100]
	$\hat{c}_{i,k}(ms)$ = [10, 100]
	$P_{l,s}$ = [0.1, 0.5]
	$P_{l,ind}$ = [0.03, 0.07]
	$C_{l,ef}$ = [0.8, 1.2]
	m_l = [2.5, 3.0]
	$f_{l,max}$ = 1 GHz
	$\lambda_{l,max}$ = [0.0003, 0.0009]

as presented in Table 18. The findings accentuated the algorithms' proficiency in several key areas, including energy efficiency, system reliability, resource utilization, and sensitivity analysis across a range of input variations.

i Energy consumption
The goal of this evaluation is to assess the proposed algorithm's performance by analyzing its energy consumption for different combinations of tasks. The assessment utilizes a set of parallel applications that are randomly generated via a DAG generator [100], where the deadline and reliability requirements for completing each application are calculated as $DL(G) = LB(G) * 1.4$ and $R_{e(goal)}(G) = 0.90$ respectively.
A Table 19 is presented that compares the energy consumption results from the proposed algorithm against seven other metaheuristic algorithms. This allows a thorough analysis and side-by-side comparison of the energy efficiencies across these different solution techniques when dealing with the randomized parallel application workloads with the specified deadlines.
The data presented in Table 19 indicates that when evaluating performance metrics, the ACO algorithm consistently exhibits higher energy consumption compared to the seven other algorithms under consideration. In the initial stages, the AHA and DA algorithms deliver superior and comparable results to KH and PSO, respectively. However, as the problem size scales up, the KH and PSO algorithms outperform AHA and DA, demonstrating more efficient outcomes. Among the remaining algorithms, the GWO technique demonstrates its strength by outperforming the WOA method while attaining comparable energy efficiency. Strikingly, the newly proposed HWWO algorithm surpasses all the other contenders, exhibiting substantially lower energy consumption levels. The HWWO algorithm establishes itself as the leading performer in terms of optimizing energy consumption across the diverse set of test scenarios explored in this evaluation. The graphical representation in Fig. 26 depicts energy consumption levels across different sets of tasks. A clear pattern emerges: higher energy usage as more tasks is added. However, the proposed algorithm's energy consumption values are noticeably lower than existing methods, indicating superior efficiency. For 100 processors, the HWWO algorithm minimizes energy consumption by 18%–24% less than GWO and WOA respectively. This substantial reduction highlights the proposed approach's energy-saving advantages, especially with increasing tasks.

ii System reliability
This part evaluates the proposed algorithm's effectiveness by examining reliability across varying task combinations. It analyzes reliability metrics for different task counts, targeting a reliability goal of $R_{e(min)}(G) \leq R_{e(goal)}(G) \leq R_{e(max)}(G)$ or $0.88371 \leq R_{e(goal)}(G) \leq 0.98577$ at 0.90. The evaluation utilizes randomly generated parallel applications with deadlines calculated as

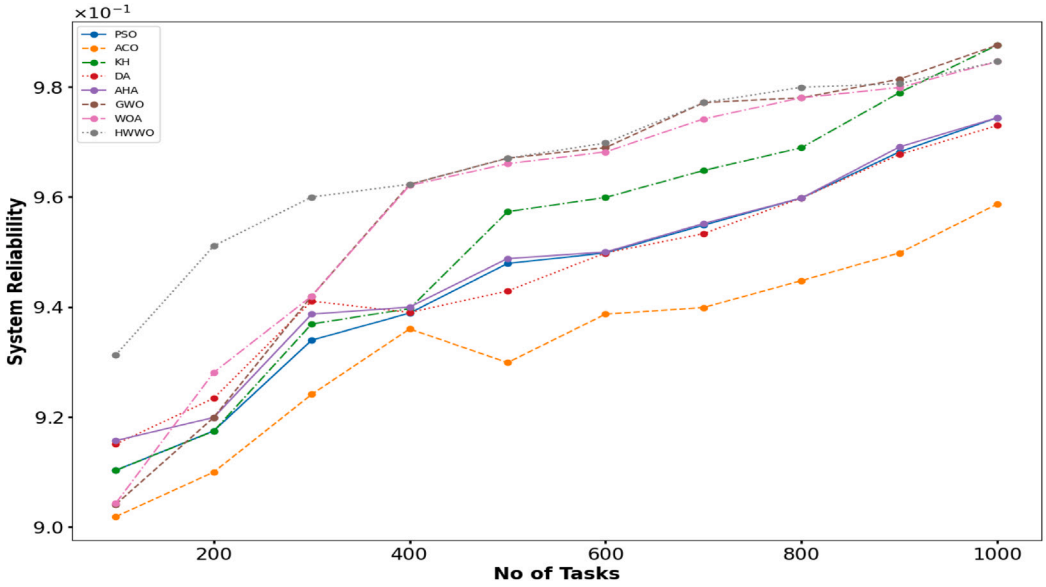

Fig. 27. Reliability outcomes for metaheuristic algorithms with varying task numbers.

$DL(G) = LB(G) * 1.4$. Fig. 27 presents a comparative analysis of reliability results obtained from the proposed algorithm and seven other metaheuristic algorithms. This comparison enables a thorough assessment of reliability performance across these solution techniques when handling randomized parallel application workloads with specified deadlines.

Fig. 27 highlights the performance metrics, indicating that the ACO algorithm consistently exhibits lower system reliability compared to the seven other algorithms evaluated. Initially, AHA and DA algorithms demonstrate superior and comparable reliability results to KH and PSO respectively, but as the problem size increases, the latter two outperform the former, exhibiting more efficient outcomes. Among the remaining algorithms, the GWO technique outperforms the WOA method, while the newly proposed HWWO algorithm surpasses all other contenders, exhibiting substantially higher reliability levels in most cases and maximizing system reliability by 8%–9% more than GWO and WOA for 100 processors.

iii Resource utilization

This meticulously designed study aims to evaluate the effectiveness of the proposed algorithm by thoroughly examining resource utilization across various task combinations. The examination of resource utilization [78,79] mainly focuses on computation time and compares it with other existing models. The assessment encompasses a variety of randomly generated parallel applications, where the application deadline is determined by the formula $DL(G) = LB(G) * 1.4$. To offer an in-depth and thorough comparative analysis of resource utilization, Fig. 28 (in %) has been carefully crafted. This figure showcases the optimal results achieved by eight distinct metaheuristic algorithms.

Optimal resource utilization is a critical factor in achieving profitability for heterogeneous computing systems. Higher resource utilization directly corresponds to increased profits for service providers. The figure presents a comparative analysis of resource utilization between the proposed HWWO approach and established metaheuristic frameworks. The results indicate that the HWWO algorithm demonstrates superior performance, substantially enhancing resource utilization by 42%, 62%, 22%, 42.48%, 21.86%, 11.83%, and 14% in comparison to PSO, ACO, KH, DA, AHA, GWO, and WOA respectively, across a range of computational tasks. The empirical evidence derived from the experimental findings validates that the proposed HWWO approach facilitates more efficient resource utilization than the existing metaheuristic frameworks.

iv Sensitivity analysis

In this subsection, the performance of the proposed method is evaluated through sensitivity analysis. Sensitivity analysis is a technique employed to ascertain the extent to which the output of a model is influenced by variations in the input parameters. It helps to identify the inputs that have the most significant influence on the output and assess the model's robustness to variations in these inputs.

The assessment encompasses a variety of randomly generated parallel applications, where the application deadline is determined by the formula $DL(G) = LB(G) * 1.4$. In this evaluation, the sensitivity of the proposed HWWO model is investigated concerning the completion time of tasks scheduling. The overall completion time for HWWO and other existing metaheuristic approaches is presented in Table 20 with varying task quantities, for which a sensitivity analysis is conducted. The optimization problem is addressed using a One-at-a-Time (OAT) based method, where the proposed technique's performance is assessed through sensitivity analysis.

From Table 20, it can be concluded that the proposed HWWO technique gave superior outcomes compared to other techniques. It decreased computation time by 46.07%, 47.59%, 43.81%, 46.11%, 41.84%, 28.85%, and 30.27% in comparison to PSO, ACO, KH, DA, AHA, GWO, and WOA respectively. To further analyze the performance, the average sensitivity of each algorithm is calculated using the OAT technique, as detailed in Table 21.

The table analysis shows the proposed HWWO model has the lowest average sensitivity ratio (0.19), indicating least sensitivity to task number changes among the algorithms. This lower sensitivity suggests HWWO is more robust and reliable for varying workloads, making it preferable in environments with fluctuating task quantities.

6.3.2. Scenario 2

This part focuses on an in-depth evaluation of the proposed HWWO-based approach by experimenting with different processor counts. The HWWO algorithm is tested alongside seven well-established metaheuristic algorithms: PSO, ACO, KH, DA, AHA, GWO, and WOA. These

Table 19
Analysis of energy consumptions for varying tasks.

$\|X\|$	Performance metric	Algorithm							
		PSO	ACO	KH	DA	AHA	GWO	WOA	HWWO
100	Best	6.08E+3	6.32E+3	5.25E+3	5.82E+3	5.25E+3	3.57E+3	3.67E+3	**3.52E+3**
	Mean	6.13E+3	6.37E+3	5.25E+3	5.84E+3	5.25E+3	3.57E+3	3.67E+3	**3.52E+3**
	Worst	6.13E+3	6.37E+3	5.29E+3	5.84E+3	5.29E+3	3.57E+3	3.68E+3	**3.52E+3**
	St. Dev	5.30E+1	7.23E+2	5.55E−3	4.15E+1	5.55E−3	**1.13E−13**	4.61E−4	**1.13E−13**
200	Best	6.51E+3	6.74E+3	5.83E+3	6.31E+3	5.69E+3	3.92E+3	**3.88E+3**	**3.88E+3**
	Mean	6.51E+3	6.79E+3	5.85E+3	6.31E+3	5.69E+3	3.93E+3	**3.89E+3**	**3.89E+3**
	Worst	6.51E+3	6.79E+3	5.87E+3	6.35E+3	5.69E+3	3.95E+3	**3.89E+3**	**3.89E+3**
	St. Dev	2.07E−4	7.03E+1	3.33E+1	7.09E+1	**3.08E−7**	5.12E+1	1.02E+1	1.02E+1
300	Best	6.97E+3	7.58E+3	6.42E+3	6.93E+3	6.31E+3	4.87E+3	4.64E+3	**4.58E+3**
	Mean	6.99E+3	7.58E+3	6.49E+3	6.95E+3	6.33E+3	4.87E+3	4.66E+3	**4.59E+3**
	Worst	6.99E+3	7.58E+3	6.49E+3	6.95E+3	6.38E+3	5.95E+3	4.69E+3	**4.61E+3**
	St. Dev	7.35E+1	**2.43E−6**	3.11E+2	7.61E+1	1.01E+1	4.02E−1	3.12E+0	7.17E+1
400	Best	7.48E+3	7.98E+3	6.89E+3	7.23E+3	6.77E+3	**6.05E+3**	**6.05E+3**	**6.05E+3**
	Mean	7.48E+3	7.99E+3	6.91E+3	7.28E+3	6.77E+3	**6.05E+3**	**6.05E+3**	**6.05E+3**
	Worst	7.48E+3	7.99E+3	6.96E+3	7.28E+3	6.77E+3	**6.05E**+3	**6.05E+3**	**6.05E+3**
	St. Dev	7.15E−4	1.85E−1	1.13E+2	4.25E−4	1.33E−4	**3.02E−8**	**3.02E−8**	**3.02E−8**
500	Best	8.36E+3	9.39E+3	7.49E+3	8.16E+3	7.89E+3	6.76E+3	6.66E+3	**6.57E+3**
	Mean	8.53E+3	9.46E+3	7.49E+3	8.23E+3	7.89E+3	6.81E+3	6.68E+3	**6.65E+3**
	Worst	8.67E+3	9.51E+3	7.49E+3	8.27E+3	7.92E+3	6.81E+3	6.68E+3	**6.65E+3**
	St. Dev	1.81E+1	1.33E+1	**2.73E−6**	1.44E+0	4.53E+1	4.11E+3	5.00E+1	9.01E+3
600	Best	1.31E+4	1.63E+4	9.78E+3	1.53E+4	9.97E+3	8.77E+3	9.04E+3	**8.52E+3**
	Mean	1.33E+4	1.67E+4	9.78E+3	1.55E+4	9.99E+3	8.79E+3	9.14E+3	**8.52E+3**
	Worst	1.37E+4	1.67E+4	9.81E+3	1.59E+4	9.99E+3	8.93E+3	9.14E+3	**8.52E+3**
	St. Dev	6.01E−1	4.35E−2	4.47E+0	2.33E−4	3.43E−2	9.11E−1	7.12E+1	**6.16E−12**
700	Best	1.83E+4	2.44E+4	1.41E+4	2.34E+4	1.83E+4	9.89E+3	1.03E+4	**9.46E+3**
	Mean	1.83E+4	2.54E+4	1.49E+4	2.39E+4	1.83E+4	9.89E+3	1.11E+4	**9.47E+3**
	Worst	1.83E+4	2.57E+4	1.58E+4	2.43E+4	1.83E+4	9.93E+3	1.22E+4	**9.49E+3**
	St. Dev	**3.01E−8**	6.63E+4	2.65E+2	2.73E+4	3.01E−8	6.66E−2	3.01E+2	3.22E−6
800	Best	3.15E+4	3.51E+4	2.41E+4	3.35E+4	2.76E+4	1.52E+4	1.94E+4	**1.06E+4**
	Mean	3.35E+4	3.51E+4	2.53E+4	3.35E+4	2.78E+4	1.64E+4	1.94E+4	**1.06E+4**
	Worst	3.35E+4	3.51E+4	2.58E+4	3.37E+4	2.78E+4	1.64E+4	1.94E+4	**1.06E+4**
	St. Dev	2.32E+4	2.32E−7	4.08E+2	2.32E+1	8.22E+1	3.07E+4	8.17E−8	**3.07E−8**
900	Best	4.15E+4	4.22E+4	3.95E+4	4.19E+4	4.05E+4	3.51E+4	3.91E+4	**2.55E+4**
	Mean	4.15E+4	4.22E+4	3.95E+4	4.22E+4	4.11E+4	3.55E+4	3.95E+4	**2.55E+4**
	Worst	4.15E+4	4.27E+4	3.95E+4	4.29E+4	4.11E+4	3.55E+4	3.95E+4	**2.55E+4**
	St. Dev	1.81E−6	6.21E+0	**5.15E−12**	1.81E+2	1.81E+1	8.25E+0	5.15E−2	**5.15E−12**
1000	Best	4.71E+4	5.28E+4	4.13E+4	5.08E+4	4.60E+4	3.81E+4	4.13E+4	**3.25E+4**
	Mean	4.73E+4	5.32E+4	4.23E+4	5.13E+4	4.60E+4	3.85E+4	4.23E+4	**3.41E+4**
	Worst	4.73E+4	5.32E+4	4.23E+4	5.13E+4	4.60E+4	3.85E+4	4.23E+4	**3.47E+4**
	St. Dev	2.01E+1	6.08E+1	1.51E+3	8.08E+4	**3.01E−7**	4.44E−2	1.51E+3	6.61E+4

Table 20
Comparative analysis of task completion times across various metaheuristic techniques under varying tasks.

$\|X\|$	Algorithm							
	PSO	ACO	KH	DA	AHA	GWO	WOA	HWWO
100	1.58E+2	1.58E+2	1.56E+2	1.56E+2	1.35E+2	9.57E+1	1.03E+2	**7.62E+1**
200	2.01E+2	2.01E+2	1.88E+2	1.98E+2	1.78E+2	1.68E+2	1.22E+2	**8.98E+1**
300	2.67E+2	2.58E+2	2.42E+2	2.51E+2	2.36E+2	2.18E+2	1.88E+2	**1.07E+2**
400	4.49E+2	4.68E+2	4.17E+2	4.38E+2	4.17E+2	2.75E+2	2.54E+2	**1.59E+2**
500	4.96E+2	4.96E+2	4.85E+2	5.09E+2	4.98E+2	3.12E+2	3.37E+2	**2.32E+2**
600	5.31E+2	5.53E+2	5.08E+2	5.63E+2	5.37E+2	3.78E+2	3.95E+2	**2.98E+2**
700	5.83E+2	5.96E+2	5.51E+2	5.93E+2	5.83E+2	4.39E+2	4.39E+2	**3.55E+2**
800	6.15E+2	6.58E+2	6.01E+2	6.55E+2	6.26E+2	5.07E+2	5.07E+2	**3.83E+2**
900	6.61E+2	7.12E+2	6.66E+2	7.09E+2	6.66E+2	5.61E+2	5.71E+2	**4.02E+2**
1000	7.11E+2	7.55E+2	6.76E+2	7.38E+2	6.88E+2	6.04E+2	6.04E+2	**4.59E+2**

Table 21
The average sensitivity for each algorithm of Table 20.

Algorithm	PSO	ACO	KH	DA	AHA	GWO	WOA	HWWO
Avg sensitivity	0.37	0.42	0.39	0.38	0.34	0.27	0.29	0.19

algorithms are employed to examine the performance of the HWWO method under varying numbers of processors, while maintaining a constant number of tasks. The detailed findings and analysis are presented subsequently.

Processors range:	100–1000
Task count:	1000

The findings accentuated the algorithms' proficiency in several key areas, including energy efficiency, system reliability, resource utilization, and sensitivity analysis across a range of input variations.

i Energy consumption
The goal of this evaluation is to assess the proposed algorithm's performance by analyzing its energy consumption for different

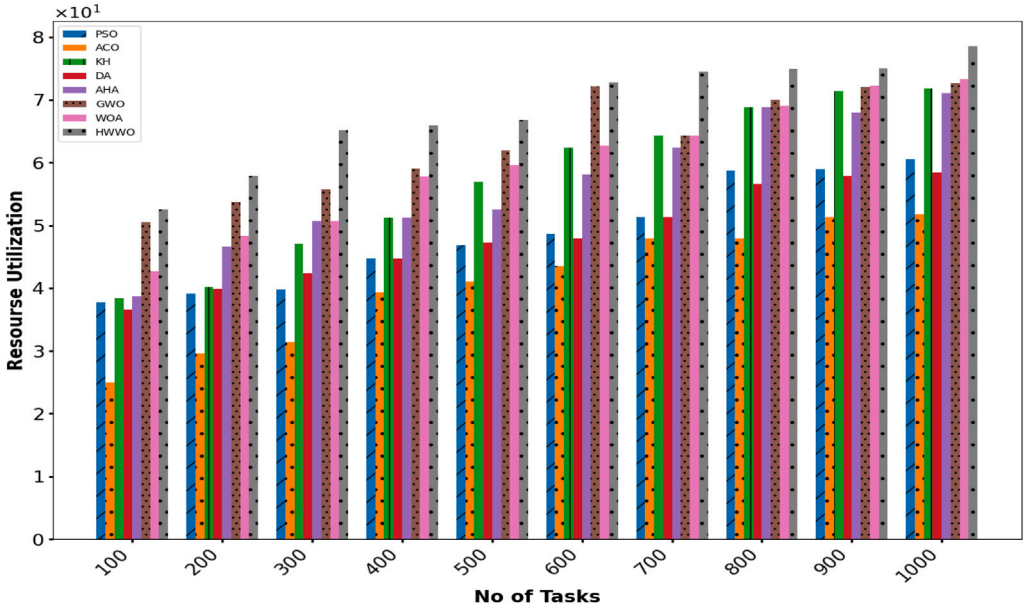

Fig. 28. Comparative analysis graph of resource utilization for metaheuristic techniques with varying tasks.

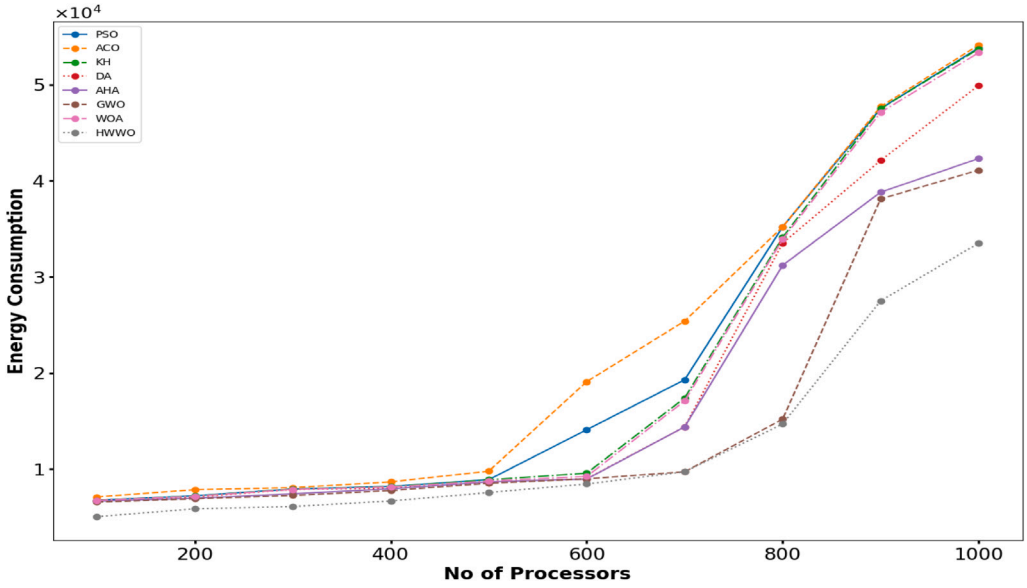

Fig. 29. Energy consumption for metaheuristic algorithms with respect to varying processors.

combinations of processors. The assessment utilizes a set of parallel applications that are randomly generated via a DAG generator [100], where the deadline and reliability requirements for completing each application are calculated as $DL(G) = LB(G) * 1.4$ and $R_{e(goal)}(G) = 0.90$ respectively.

The energy consumption results displayed in Fig. 29 compare the proposed algorithm against other metaheuristic algorithms. The figure reveals that the ACO algorithm consistently consumes more energy than the other algorithms evaluated. Among these algorithms, the GWO technique outperforms the WOA method while achieving similar energy efficiency as the AHA algorithm. Notably, the proposed algorithm exhibits significantly lower energy consumption than existing methods, indicating superior efficiency. This is due to the proposed algorithm defining a circular neighborhood around solutions based on its encirclement mechanism, enabling the HWWO algorithm to outperform others and find better solutions. When tested with 1000 tasks, the HWWO algorithm reduced energy consumption by 13.46-23.81% compared to GWO and WOA, respectively. This substantial reduction underscores the energy-saving advantages of the proposed approach, especially as the number of processors increases.

ii System reliability

This part evaluates the proposed algorithm's effectiveness by examining reliability across varying processors combinations. It analyzes reliability metrics for different task counts, targeting a reliability goal of $R_{e(min)}(G) \leq R_{e(goal)}(G) \leq R_{e(max)}(G)$ or $0.88371 \leq R_{e(goal)}(G) \leq 0.98577$ at 0.90. The evaluation utilizes randomly generated parallel applications with deadlines calculated as $DL(G) = LB(G) * 1.4$.

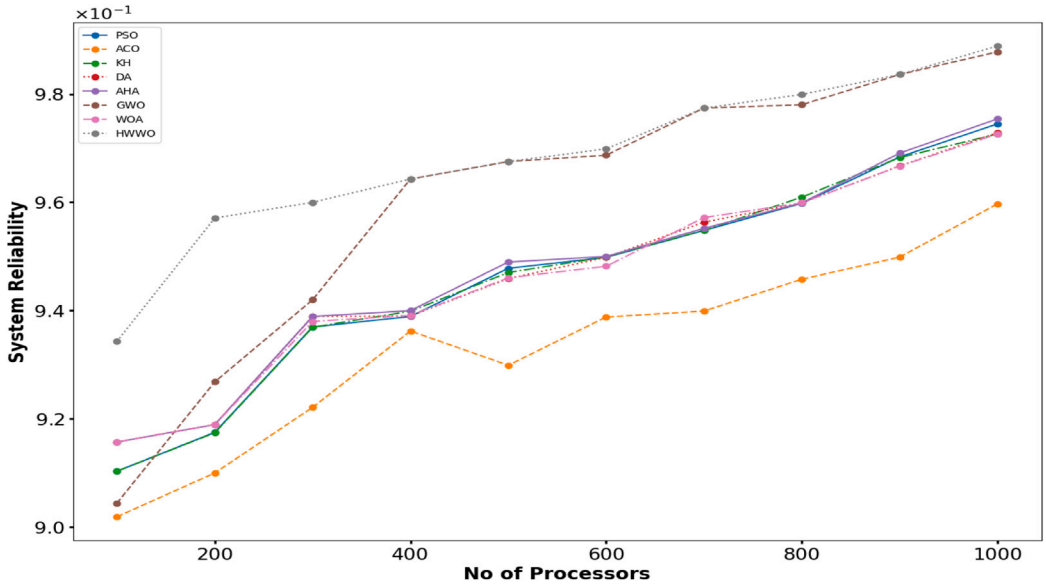

Fig. 30. Reliability outcomes for metaheuristic algorithms with varying processor numbers.

Fig. 30 highlights the performance metrics, indicating that the ACO algorithm consistently exhibits lower system reliability compared to the seven other algorithms evaluated. Among the algorithms, the GWO technique outperforms the WOA method, while the newly proposed HWWO algorithm surpasses all other contenders, exhibiting substantially higher reliability levels in most cases and maximizing system reliability by 5%–8% more than GWO and WOA for 1000 tasks.

iii Resource utilization

This part intends to assess the efficacy of the proposed algorithmic approach by conducting a comprehensive analysis of resource utilization across different processor configurations. The assessment encompasses a variety of randomly generated parallel applications, where the application deadline is determined by the formula $DL(G) = LB(G) * 1.4$. To offer an in-depth and thorough comparative analysis of resource utilization, Fig. 31 (in %) has been carefully crafted. This figure showcases the optimal results achieved by eight distinct metaheuristic algorithms.

The results from figure indicate that the HWWO algorithm demonstrates superior performance, substantially enhancing resource utilization by 26.92%, 29.79%, 19.34%, 16.48%, 16.86%, 9.83%, and 25.15% in comparison to PSO, ACO, KH, DA, AHA, GWO, and WOA respectively, across a range of computational processors. The empirical evidence derived from the experimental findings validates that the proposed HWWO approach facilitates more efficient resource utilization than the existing metaheuristic frameworks.

iv Sensitivity analysis

In this subsection, the effectiveness of the proposed method is assessed via sensitivity analysis in relation to different processor counts. The assessment encompasses a variety of randomly generated parallel applications, where the application deadline is determined by the formula $DL(G) = LB(G) * 1.4$. In this evaluation, the sensitivity of the HWWO model concerning tasks scheduling completion times is explored. Table 22 displays the overall completion times for HWWO and other existing metaheuristic approaches across varying processors, for which a sensitivity analysis is performed.

Table 22 shows that the HWWO technique yielded better results than other methods, reducing computation time by 30.68% to 47.94% compared to various metaheuristic techniques. Additionally, Table 23 provides the average sensitivity of each algorithm, determined through the OAT technique, for further performance analysis.

The table analysis unveils that the proposed HWWO model demonstrates the minimal average sensitivity ratio (0.22), indicating its superior resistance to fluctuations in processor availability compared to other algorithms. This lower sensitivity makes HWWO more robust and reliable for different workloads, making it ideal for environments with varying processor numbers.

6.3.3. Scenario 3

To validate the proposed HWWO algorithm's efficacy against established metaheuristic optimization techniques, this section employs a set of unimodal test functions. These benchmark functions, sourced from [28] and tabulated in Table 24, assess the algorithm's exploitation capabilities and overall optimization performance. Ensuring a fair comparison, all tests utilize a population size of 30, with a maximum of 15,000 function evaluations across 500 iterations. Each algorithm is executed 30 times independently on these functions. The evaluation metrics, including mean, standard deviation, best and worst fitness values from the independent runs, are then computed and presented in Table 25.

An analysis of Table 25, which presents the results for unimodal functions, clearly demonstrates the superior exploitation capability of the proposed HWWO algorithm. This is evident from the fact that the HWWO algorithm achieves the best mean fitness values in the majority of cases, as indicated by the bold entries. In contrast, the existing algorithms being evaluated display comparatively inferior performance.

In evaluating the algorithms' performance based on the highest fitness scores across 30 runs, it is observed that the HWWO algorithm outperformed others, securing the highest number of best fitness scores (5/7). In comparison, WOA and GWO attained fewer best fitness scores (1/7 each), while all other algorithms do not achieve the best fitness in any of the runs. These results suggest that the HWWO algorithm demonstrates greater consistency and reliability in attaining optimal fitness values compared to others.

The statistical analysis using the Wilcoxon signed-rank test is presented in Table 26, which evaluates the performance of the HWWO

Table 22
Comparative analysis of task completion times across various metaheuristic techniques under varying processors.

$\|Y\|$	Algorithm							
	PSO	ACO	KH	DA	AHA	GWO	WOA	HWWO
100	1.28E+2	1.28E+2	1.26E+2	9.56E+1	8.62E+1	8.02E+1	1.07E+2	**6.82E+1**
200	2.12E+2	2.21E+2	1.88E+2	1.88E+2	1.73E+2	1.64E+2	1.85E+2	**8.88E+1**
300	2.67E+2	2.71E+2	2.71E+2	2.51E+2	2.38E+2	2.38E+2	2.65E+2	**1.17E+2**
400	4.89E+2	4.98E+2	4.72E+2	4.48E+2	4.37E+2	2.84E+2	4.59E+2	**1.54E+2**
500	4.98E+2	5.21E+2	4.85E+2	5.19E+2	4.78E+2	3.62E+2	5.19E+2	**2.22E+2**
600	5.71E+2	5.73E+2	5.64E+2	5.62E+2	5.17E+2	3.73E+2	5.64E+2	**2.98E+2**
700	6.13E+2	6.26E+2	5.95E+2	5.89E+2	5.73E+2	4.49E+2	5.93E+2	**3.75E+2**
800	6.65E+2	6.68E+2	6.61E+2	6.46E+2	6.46E+2	5.17E+2	6.55E+2	**3.86E+2**
900	7.41E+2	7.52E+2	7.36E+2	7.15E+2	6.76E+2	5.61E+2	7.27E+2	**4.32E+2**
1000	7.53E+2	7.75E+2	7.46E+2	7.43E+2	6.91E+2	6.12E+2	7.43E+2	**4.54E+2**

Table 23
The average sensitivity for each algorithm of Table 22.

Algorithm	PSO	ACO	KH	DA	AHA	GWO	WOA	HWWO
Avg sensitivity	0.41	0.45	0.42	0.39	0.33	0.29	0.37	0.22

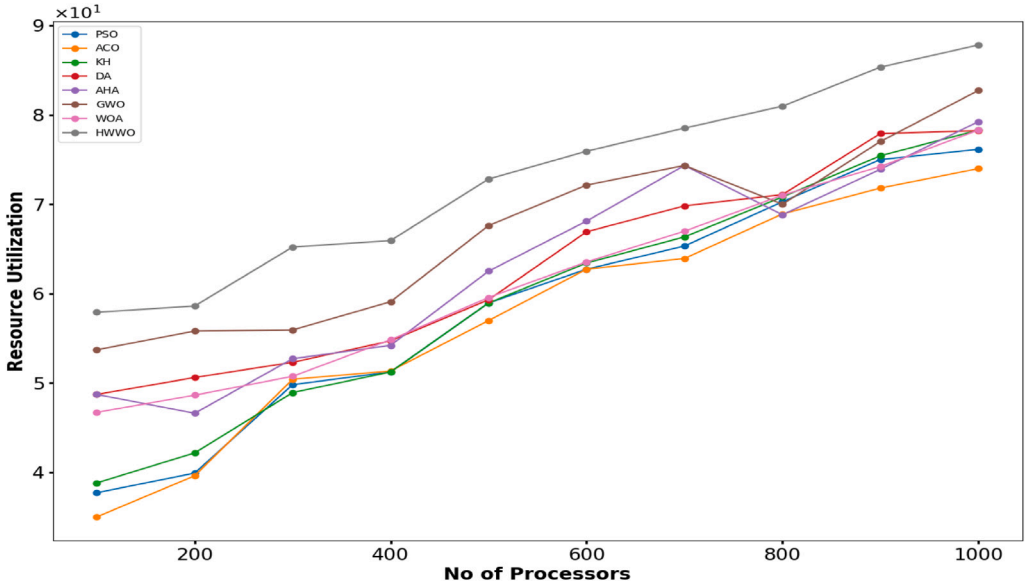

Fig. 31. Comparative analysis graph of resource utilization for metaheuristic techniques with varying processors.

Table 24
Description of unimodal benchmark functions.

Function	Dimensions	Range	f_{min}
$F_1 = \sum_{i=1}^{n} x_i^2$	30	[−100, 100]	0
$F_2 = \sum_{i=1}^{n} \|x_i\| + \prod_{i=1}^{n} \|x_i\|$	30	[−10, 10]	0
$F_3 = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2$	30	[−100, 100]	0
$F_4 = max_i\{\|x_i\|, 1 \le i \le n\}$	30	[−100, 100]	0
$F_5 = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	30	[−30, 30]	0
$F_6 = \sum_{i=1}^{n} ([x_i + 0.5])^2$	30	[−100, 100]	0
$F_7 = \sum_{i=1}^{n} i x_i^4 + random[0, 1)$	30	[−1.28, 1.28]	0

algorithm based on unimodal benchmark functions. This table shows the rank of the HWWO algorithm in comparison to the second algorithm, focusing on the best fitness values. T^+ represents the superiority of the proposed HWWO technique. P-values, calculated at a 5% significance level, test the null hypothesis that the median difference between the algorithms is zero. The final row of Table 26 consolidates the counts of T^+ and T^-, along with the test statistic, offering a clear summary of the results. The analysis reveals significant improvements in the HWWO algorithm's performance, with T^+ accounting for 93.87% and T^- for 6.12% of the evaluated benchmarks ($p < 0.05$). These results suggest that the proposed algorithm achieves superior performance in solving unimodal benchmark problems, demonstrating faster convergence and greater accuracy compared to existing methods.

Finally, the convergence behavior of the proposed HWWO algorithm is depicted through convergence curves compared with other algorithms in Fig. 32. The convergence rate, which evaluates an algorithm's efficiency in reaching the optimal solution, is analyzed by comparing HWWO's performance with existing metaheuristic techniques. In the graph, the x-axis represents the number of iterations, while the y-axis shows the average fitness values computed over 1000

Table 25
Outcomes of using unimodal functions.

F	Measure	Optimization algorithm							
		PSO	ACO	KH	DA	AHA	GWO	WOA	HWWO
F_1	Best	1.709E−13	1.471E−10	1.660E−19	1.835E−13	3.003E−49	3.010E−79	5.763E−60	**3.755E−94**
	Mean	8.644E−11	1.121E−09	6.660E−17	4.992E−12	9.206E−48	2.013E−67	1.847E−57	**2.401E−87**
	Worst	2.483E−10	6.170E−09	5.873E−16	3.849E−11	2.162E−47	8.057E−67	9.399E−57	**8.552E−87**
	St. Dev	1.889E−01	7.223E−04	1.861E−01	3.046E−03	1.418E−47	**3.413E−72**	2.149E−03	4.027E−72
F_2	Best	5.069E−12	8.035E−06	1.803E−34	1.609E−09	7.112E−29	4.925E−55	2.757E−65	**2.283E−80**
	Mean	9.183E−11	1.290E−03	6.257E−33	1.325E−08	1.116E−28	2.350E−52	9.432E−45	**5.662E−60**
	Worst	2.272E−10	5.056E−03	1.230E−32	3.626E−08	1.931E−28	9.217E−52	3.775E−44	**2.220E−59**
	St. Dev	9.594E−11	2.511E−03	6.834E−33	1.611E−08	5.517E−27	3.559E−52	1.088E−44	**1.106E−59**
F_3	Best	5.077E−07	5.783E−02	2.497E−06	3.273E−26	3.156E−05	**3.10E−128**	6.837E−03	6.389E−28
	Mean	3.377E−01	1.123E+00	8.221E−05	1.984E−24	9.296E−03	**2.683E−82**	1.808E−02	2.711E−27
	Worst	1.350E+00	3.310E+00	1.607E−04	7.912E−24	3.513E−02	**1.073E−81**	4.471E−02	1.080E−26
	St. Dev	6.753E−01	1.481E+00	8.191E−05	3.952E−24	1.723E−02	**5.367E−82**	1.784E−02	5.398E−27
F_4	Best	7.275E−04	1.655E−03	5.300E−17	3.102E−05	1.969E−54	2.551E−21	1.388E−62	**2.277E−78**
	Mean	1.654E−02	5.793E−02	8.969E−16	2.358E−04	2.149E−49	1.030E−19	2.346E−43	**2.971E−53**
	Worst	4.365E−02	1.838E−01	1.704E−15	4.706E−04	8.596E−49	3.632E−19	9.346E−43	**1.188E−52**
	St. Dev	1.975E−02	8.624E−02	8.320E−16	2.214E−04	4.297E−49	1.741E−19	4.667E−43	**5.942E−53**
F_5	Best	7.65461E	7.68388E	9.70569E	3.126E−01	6.15363E	4.291E−02	1.18142E	**3.113E−04**
	Mean	7.99255E	8.42474E	5.91556E+02	2.924E+00	6.68606E	7.121E−01	4.65548E	**1.267E−03**
	Worst	8.42156E	9.31820E	1.20253E+03	5.389E+00	7.23273E	1.596E+00	6.10117E	**2.696E−03**
	St. Dev	3.32511E−01	7.36628E−01	5.00585E+02	2.924E+00	6.07412E−1	7.068E−01	2.32924E	**1.013E−03**
F_6	Best	3.543E−01	7.685E−05	3.620E−06	9.22444E−10	2.039E−03	2.907E−16	**6.490E−20**	1.549E−12
	Mean	6.382E−01	6.237E−03	4.380E−06	1.23821E−09	4.113E−03	7.377E−14	**3.630E−16**	7.981E−11
	Worst	8.362E−01	1.772E−02	4.993E−06	1.62791E−09	7.156E−03	2.903E−13	**1.324E−15**	1.891E−10
	St. Dev	2.061E−01	1.444E−13	5.766E−07	2.97023E−10	2.229E−03	8.229E−03	**6.436E−16**	9.242E−11
F_7	Best	3.457E−03	6.312E−03	8.926E−04	9.48224E−04	2.857E−04	3.366E−05	1.03E+4	**6.443E−06**
	Mean	9.097E−03	2.395E−02	1.791E−03	1.97178E−03	6.290E−04	2.899E−04	1.11E+4	**2.188E−04**
	Worst	2.001E−02	5.483E−02	2.728E−03	3.58221E−03	8.085E−04	8.911E−04	1.22E+4	**5.489E−04**
	St. Dev	7.412E−03	2.124E−02	7.973E−04	1.24604E−03	2.388E−04	4.095E−04	3.01E+2	**2.247E−04**

Table 26
Results of Wilcoxon signed-rank test of Table 25.

Problem	HWWO vs PSO	HWWO vs ACO	HWWO vs KH	HWWO vs DA	HWWO vs AHA	HWWO vs GWO	HWWO vs WOA
	Rank	Rank	Rank	Rank	Rank	Rank	Rank
F_1	1	1	2	2	2	1	3
F_2	2	2	1	4	3	2	1
F_3	3	6	4	1	4	3	5
F_4	4	4	3	5	1	4	2
F_5	7	7	7	7	7	7	6
F_6	6	3	5	3	6	5	4
F_7	5	5	6	6	5	6	7
p-value	0.0355	0.0013	0.0281	0.0176	0.0262	0.0474	0.0087
T^+ = Sum of positive number ranks	28	28	28	28	28	20	24
T^- = Sum of negative number ranks	0	0	0	0	0	8	4
$T = \min(T^+, T^-)$	0	0	0	0	0	8	4

tasks using 100 processors. For clarity, the graph illustrates the average fitness values from 10 independent runs, each evaluated over 500 iterations. As shown in Fig. 32, the HWWO algorithm demonstrates rapid convergence toward optimal solutions, outperforming other algorithms. This superior performance stems from HWWO's hybrid design, which integrates the strengths of WOA and GWO. By combining these techniques, HWWO effectively overcomes the limitations of premature and slow convergence inherent in WOA. The figure underscores the limitations of PSO and ACO algorithms, primarily their weak exploitation capabilities. Despite iterating extensively through the solution space, these algorithms often fail to reach the optimal solution due to an imbalance between exploration and exploitation phases. Similarly, DA and KH exhibit strong exploratory abilities in the initial stages but frequently become trapped in local optima, preventing convergence to the optimal solution. In contrast, WOA initially outperforms GWO in generating promising solutions, but GWO surpasses WOA in later iterations by refining the search process. HWWO, however, demonstrates steady improvement in fitness value with increasing iterations, reflecting its well-balanced integration of exploration and exploitation for enhanced optimization performance.

7. Conclusions

To address the challenge of tasks scheduling in a heterogeneous distributed computing environment, this research proposes a hybrid meta-heuristic technique called HWWO, which amalgamates the WOA and the GWO. The paper presents a reliability-based energy-efficient scheduling model designed to reduce energy consumption and enhance the reliability of applications running on heterogeneous computing platforms, all while adhering to strict deadline requirements. The applications are elegantly modeled using DAGs. The article proposes a novel scheduling algorithm that combines the WOA and the GWO with DVFS capabilities, along with an insert-reversed block operation. This hybrid approach aims to minimize both static and dynamic energy consumption. The article presents a refined technique to simultaneously tackle the challenges of tasks scheduling on appropriate processors while considering multiple objectives. The proposed method seeks to

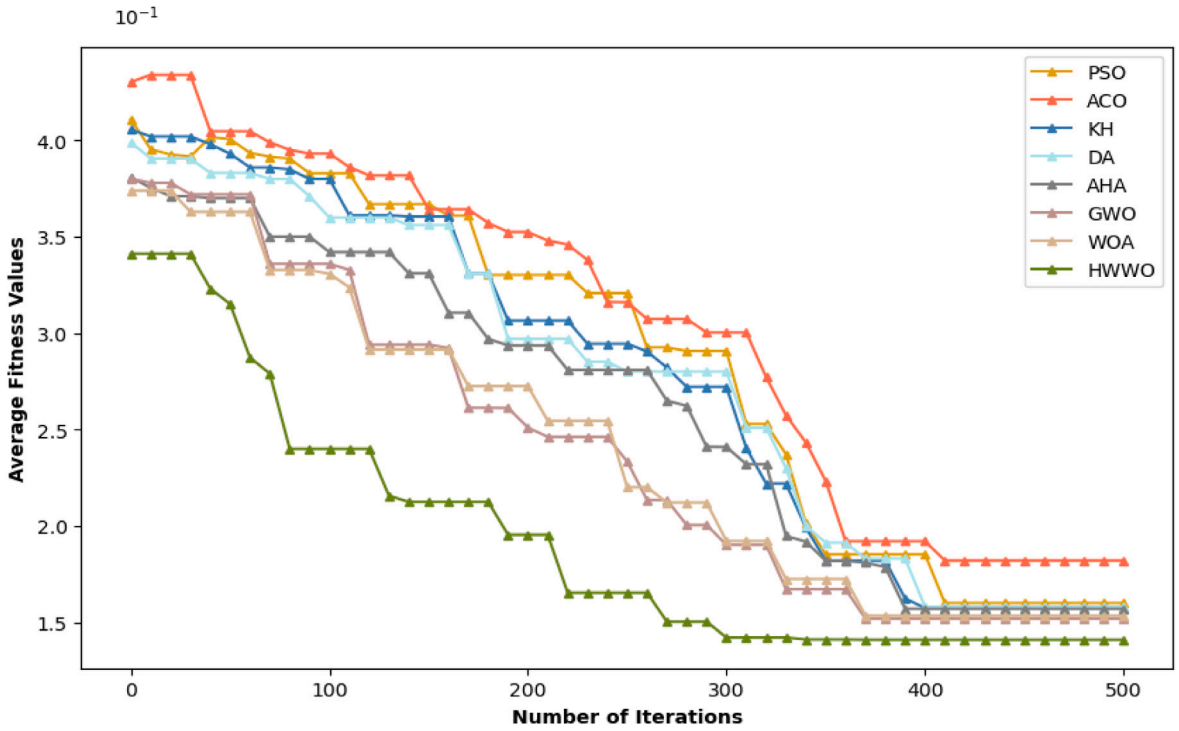

Fig. 32. Comparison of convergence curves of HWWO and literature algorithms.

optimize overall energy consumption, computational time, and system reliability concurrently, offering a comprehensive solution to address these critical factors. Extensive experiments highlight the proposed model's effectiveness in considerably reducing energy consumption and processing time, increasing system reliability, and maintaining low complexities. The key contributions based on experimental findings are:

i The article introduces a hybrid scheduling mechanism, termed HWWO, that integrates SI techniques, specifically WOA and the GWO, to tackle real-world applications effectively.
ii The WOA algorithm exhibits rapid convergence and strikes a balance between exploration and exploitation when solving optimization problems. However, its encircling search mechanism can occasionally lead it to converge prematurely on local optima. To mitigate this issue, a hybrid approach has been devised by incorporating the GWO, synergizing the capabilities of both optimization techniques.
iii Extensive evaluations are conducted on real-world FFT and GE applications to compare the proposed model's performance against various state-of-the-art methods.
iv The proposed algorithm is rigorously evaluated on real-world single-objective constrained optimization problems from the CEC 2020 competition. Comprehensive comparisons are conducted against the competition's state-of-the-art algorithms, including SASS, EnMODE, sCMAgES, and COLSHADE. Additionally, the algorithm's performance is assessed on a set of unimodal benchmark test functions and compared to established metaheuristic approaches.
v The experiments reveal the proposed algorithm's superiority over existing state-of-the-art and metaheuristic methods. It excels in energy efficiency, reliability maximization, computation time and SLR minimization, CCR optimization, and resource utilization enhancement across diverse scale conditions and deadline constraints.
vi The effectiveness and scalability of the proposed HWWO method are assessed through sensitivity analysis and implementation on varying tasks and processor counts. The outcomes revealed that HWWO consistently demonstrated the minimal average sensitivity ratio and rapid convergence towards optimal solutions, outperforming existing metaheuristic algorithms.
vii A Wilcoxon-signed rank test is utilized to statistically evaluate the effectiveness of the results.
viii The run time and space complexities for the proposed method are calculated, both equating to $O(|Y| * |X|) + O(|Y|)$. Notably, in terms of complexities, the proposed algorithm demonstrates superior performance compared to other existing algorithms in this domain.

7.1. Limitations and future work

In this article, a hybrid model is developed to solve the reliability-based energy-efficient tasks scheduling problem with multiple objectives. The model successfully reduces total energy consumption compared to existing methods, although it fails to reduce static energy consumption individually. As shown in Tables 12 and 16, the proposed HWWO approach exhibits superior energy consumption performance for reliability thresholds up to 0.98. However, beyond this threshold, the method fails to meet reliability constraints due to the absence of fault-tolerance mechanisms, indicating that $R_{e(goal)}(G)$ cannot always be satisfied. Addressing these limitations, future research will focus on integrating fault-tolerance mechanisms into the hybrid model more efficiently. This integration could involve implementing error detection and correction techniques and robust optimization strategies to ensure continuous and accurate tasks scheduling, even in the presence of faults.

The method is effective within computing environments where processors are fully connected. To address its limitations, enhancing the framework involves refining scheduling algorithms and assessing them across various workflows such as LIGO, SIPHT, and molecular dynamic code. Furthermore, the proposed framework's versatility allows for potential extensions to diverse computing system environments such as grid computing, cloud computing, and cluster computing.

CRediT authorship contribution statement

Karishma: Writing – original draft, Validation, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **Harendra Kumar:** Validation, Supervision, Methodology, Investigation, Formal analysis, Conceptualization.

Funding

The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Ethics approval and consent to participate

This article does not contain any studies with human participants or animals performed by any authors.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] M. Agarwal, G.M.S. Srivastava, Opposition-based learning inspired particle swarm optimization (OPSO) scheme for task scheduling problem in cloud computing, J. Ambient. Intell. Humaniz. Comput. 12 (10) (2021) 9855–9875, http://dx.doi.org/10.1007/s12652-020-02730-4.

[2] I. Strumberger, N. Bacanin, M. Tuba, E. Tuba, Resource scheduling in cloud computing based on a hybridized whale optimization algorithm, Appl. Sci. 9 (22) (2019) 4893, http://dx.doi.org/10.3390/app9224893.

[3] H. Kumar, I. Tyagi, Hybrid model for tasks scheduling in distributed real time system, J. Ambient. Intell. Humaniz. Comput. 12 (2021) 2881–2903, http://dx.doi.org/10.1007/s12652-020-02445-6.

[4] Karishma, H. Kumar, A new hybrid particle swarm optimizationalgorithm for optimal tasks scheduling in distributed computing system, Intell. Syst. Appl. 18 (2023) 200219, http://dx.doi.org/10.1016/j.iswa.2023.200219.

[5] G. Taheri, A. Khonsari, R. Entezari-Maleki, L. Sousa, A hybrid algorithm for task scheduling on heterogeneous multiprocessor embedded systems, Appl. Soft Comput. 91 (2020) 106202, http://dx.doi.org/10.1016/j.asoc.2020.106202.

[6] G. Taheri, A. Khonsari, R. Entezari-Maleki, M. Baharloo, L. Sousa, Temperature-aware dynamic voltage and frequency scaling enabled MPSoC modeling using stochastic activity networks, Microprocess. Microsyst. 60 (2018) 15–23, http://dx.doi.org/10.1016/j.micpro.2018.03.011.

[7] B.M.H. Zade, N. Mansouri, M.M. Javidi, SAEA: A security-aware and energy-aware task scheduling strategy by parallel squirrel search algorithm in cloud environment, Expert Syst. Appl. 176 (2021) 114915, http://dx.doi.org/10.1016/j.eswa.2021.114915.

[8] G. Xie, Y. Chen, X. Xiao, C. Xu, R. Li, K. Li, Energy-efficient fault-tolerant scheduling of reliable parallel applications on heterogeneous distributed embedded systems, IEEE Trans. Sustain. Comput. 3 (3) (2017) 167–181, http://dx.doi.org/10.1109/TSUSC.2017.2711362.

[9] B. Hu, Z. Cao, M. Zhou, Energy-minimized scheduling of real-time parallel workflows on heterogeneous distributed computing systems, IEEE Trans. Serv. Comput. 15 (5) (2021) 2766–2779, http://dx.doi.org/10.1109/TSC.2021.3054754.

[10] S. Chen, Z. Li, B. Yang, G. Rudolph, Quantum-inspired hyper-heuristics for energy-aware scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 27 (6) (2016) 1796–1810, http://dx.doi.org/10.1109/TPDS.2015.2462835.

[11] M. Safari, R. Khorsand, PL-DVFS: combining power-aware list-based scheduling algorithm with DVFS technique for real-time tasks in cloud computing, J. Supercomput. 74 (10) (2018) 5578–5600, http://dx.doi.org/10.1007/s11227-018-2498-z.

[12] K. Li, Energy-efficient task scheduling on multiple heterogeneous computers: Algorithms, analysis, and performance evaluation, IEEE Trans. Sustain. Comput. 1 (1) (2016) 7–19, http://dx.doi.org/10.1109/TSUSC.2016.2623775.

[13] H. Xu, R. Li, C. Pan, K. Li, Minimizing energy consumption with reliability goal on heterogeneous embedded systems, J. Parallel Distrib. Comput. 127 (2019) 44–57, http://dx.doi.org/10.1016/j.jpdc.2019.01.006.

[14] L. Zhang, M. Ai, K. Liu, J. Chen, K. Li, Reliability enhancement strategies for workflow scheduling under energy consumption constraints in clouds, IEEE Trans. Sustain. Comput. 9 (2) (2024) 155–169, http://dx.doi.org/10.1109/TSUSC.2023.3314759.

[15] L. Zhang, K. Li, K. Li, Y. Xu, Joint optimization of energy efficiency and system reliability for precedence constrained tasks in heterogeneous systems, Int. J. Electr. Power Energy Syst. 78 (2016) 499–512, http://dx.doi.org/10.1016/j.ijepes.2015.11.102.

[16] G. Xie, H. Peng, Z. Li, J. Song, Y. Xie, R. Li, K. Li, Reliability enhancement toward functional safety goal assurance in energy-aware automotive cyber-physical systems, IEEE Trans. Ind. Informatics 14 (12) (2018) 5447–5462, http://dx.doi.org/10.1109/TII.2018.2854762.

[17] L. Ye, Y. Xia, S. Tao, C. Yan, R. Gao, Y. Zhan, Reliability-aware and energy-efficient workflow scheduling in IaaS clouds, IEEE Trans. Autom. Sci. Eng. 20 (3) (2023) 2156–2169, http://dx.doi.org/10.1109/TASE.2022.3195958.

[18] X. Xiao, G. Xie, C. Xu, C. Fan, R. Li, K. Li, Maximizing reliability of energy constrained parallel applications on heterogeneous distributed systems, J. Comput. Sci. 26 (2018) 344–353, http://dx.doi.org/10.1016/j.jocs.2017.05.002.

[19] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, K. Li, Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems, IEEE Trans. Ind. Informatics 13 (4) (2016) 1629–1640, http://dx.doi.org/10.1109/TII.2016.2641473.

[20] H. Djigal, J. Feng, J. Lu, J. Ge, IPPTS: An efficient algorithm for scientific workflow scheduling in heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2021) 1057–1071, http://dx.doi.org/10.1109/TPDS.2020.3041829.

[21] Z. Deng, Z. Yan, H. Huang, H. Shen, Energy-aware task scheduling on heterogeneous computing systems with time constraint, IEEE Access 8 (2020) 23936–23950, http://dx.doi.org/10.1109/ACCESS.2020.2970166.

[22] Z. Quan, Z.-J. Wang, T. Ye, S. Guo, Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 31 (5) (2020) 1165–1182, http://dx.doi.org/10.1109/TPDS.2019.2959533.

[23] Y. Hu, J. Li, L. He, A reformed task scheduling algorithm for heterogeneous distributed systems with energy consumption constraints, Neural Comput. Appl. 32 (10) (2020) 5681–5693, http://dx.doi.org/10.1007/s00521-019-04415-2.

[24] J. Ababneh, A hybrid approach based on grey wolf and whale optimization algorithms for solving cloud task scheduling problem, Math. Probl. Eng. 2021 (1) (2021) 3517145, http://dx.doi.org/10.1155/2021/3517145.

[25] F.W. Ipeayeda, M.O. Oyediran, S.A. Ajagbe, J.O. Jooda, M.O. Adigun, Optimized gravitational search algorithm for feature fusion in a multimodal biometric system, Results Eng. 20 (2023) 101572, http://dx.doi.org/10.1016/j.rineng.2023.101572.

[26] M.O. Oyediran, S.A. Ajagbe, O.S. Ojo, R. Alshahrani, O.O. Awodoye, M.O. Adigun, White shark optimizer via support vector machine for video-based gender classification system, Multimedia Tools Appl. 84 (2025) 34645–34661, http://dx.doi.org/10.1007/s11042-024-20500-8.

[27] Karishma, H. Kumar, GWO based energy-efficient workflow scheduling for heterogeneous computing systems, Soft Comput. 29 (2025) 3469–3508, http://dx.doi.org/10.1007/s00500-025-10614-y.

[28] S. Mirjalili, A. Lewis, The whale optimization algorithm, Adv. Eng. Softw. 95 (2016) 51–67, http://dx.doi.org/10.1016/j.advengsoft.2016.01.008.

[29] S. Mirjalili, S.M. Mirjalili, A. Lewis, Grey wolf optimizer, Adv. Eng. Softw. 69 (2014) 46–61, http://dx.doi.org/10.1016/j.advengsoft.2013.12.007.

[30] H.M. Mohammed, S.U. Umar, T.A. Rashid, A systematic and meta-analysis survey of whale optimization algorithm, Comput. Intell. Neurosci. 2019 (1) (2019) 8718571, http://dx.doi.org/10.1155/2019/8718571.

[31] Z. Xu, Y. Yu, H. Yachi, J. Ji, Y. Todo, S. Gao, A novel memetic whale optimization algorithm for optimization, in: Advances in Swarm Intelligence: 9th International Conference, ICSI 2018, Shanghai, China, June 17-22, 2018, Proceedings, Part I 9, Springer, 2018, pp. 384–396, http://dx.doi.org/10.1007/978-3-319-93815-8_37.

[32] S. Li, F. Broekaert, Low-power scheduling with DVFS for common RTOS on multicore platforms, ACM SIGBED Rev. 11 (1) (2014) 32–37, http://dx.doi.org/10.1145/2597457.2597461.

[33] X. Tang, W. Shi, F. Wu, Interconnection network energy-aware workflow scheduling algorithm on heterogeneous systems, IEEE Trans. Ind. Informatics 16 (12) (2020) 7637–7645, http://dx.doi.org/10.1109/TII.2019.2962531.

[34] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced CPU energy, in: Proceedings of IEEE 36th Annual Foundations of Computer Science, IEEE, 1995, pp. 374–382, http://dx.doi.org/10.1109/SFCS.1995.492493.

[35] M. Safari, R. Khorsand, Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment, Simul. Model. Pr. Theory 87 (2018) 311–326, http://dx.doi.org/10.1016/j.simpat.2018.07.006.

[36] Z. Tang, L. Qi, Z. Cheng, K. Li, S.U. Khan, K. Li, An energy-efficient task scheduling algorithm in DVFS-enabled cloud environment, J. Grid Comput. 14 (2016) 55–74, http://dx.doi.org/10.1007/s10723-015-9334-y.

[37] G. Xie, J. Jiang, Y. Liu, R. Li, K. Li, Minimizing energy consumption of real-time parallel applications using downward and upward approaches on heterogeneous systems, IEEE Trans. Ind. Informatics 13 (3) (2017) 1068–1078, http://dx.doi.org/10.1109/TII.2017.2676183.
[38] G. Xie, H. Peng, Z. Li, J. Song, Y. Xie, R. Li, K. Li, Reliability enhancement toward functional safety goal assurance in energy-aware automotive cyber-physical systems, IEEE Trans. Ind. Informatics 14 (12) (2018) 5447–5462, http://dx.doi.org/10.1109/TII.2018.2854762.
[39] A. Javadpour, A.K. Sangaiah, P. Pinto, F. Ja'fari, W. Zhang, A.M.H. Abadi, H. Ahmadi, An energy-optimized embedded load balancing using DVFS computing in cloud data centers, Comput. Commun. 197 (2023) 255–266, http://dx.doi.org/10.1016/j.comcom.2022.10.019.
[40] S. Ijaz, E.U. Munir, S.G. Ahmad, M.M. Rafique, O.F. Rana, Energy-makespan optimization of workflow scheduling in fog–cloud computing, Computing 103 (2021) 2033–2059, http://dx.doi.org/10.1007/s00607-021-00930-0.
[41] B. Kocot, P. Czarnul, J. Proficz, Energy-aware scheduling for high-performance computing systems: A survey, Energies 16 (2) (2023) 890, http://dx.doi.org/10.3390/en16020890.
[42] Y. Hu, J. Li, L. He, A reformed task scheduling algorithm for heterogeneous distributed systems with energy consumption constraints, Neural Comput. Appl. 32 (10) (2020) 5681–5693, http://dx.doi.org/10.1007/s00521-019-04415-2.
[43] A. Benoit, M. Hakem, Y. Robert, Fault tolerant scheduling of precedence task graphs on heterogeneous platforms, in: 2008 IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–8, http://dx.doi.org/10.1109/IPDPS.2008.4536133.
[44] J. Huang, R. Li, X. Jiao, Y. Jiang, W. Chang, Dynamic DAG scheduling on multiprocessor systems: Reliability, energy, and makespan, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 39 (11) (2020) 3336–3347, http://dx.doi.org/10.1109/TCAD.2020.3013045.
[45] S. Saroja, T. Revathi, N. Auluck, Multi-criteria decision-making for heterogeneous multiprocessor scheduling, Int. J. Inf. Technol. Decis. Mak. 17 (05) (2018) 1399–1427, http://dx.doi.org/10.1142/S0219622018500311.
[46] L. Zhao, Y. Ren, K. Sakurai, Reliable workflow scheduling with less resource redundancy, Parallel Comput. 39 (10) (2013) 567–585, http://dx.doi.org/10.1016/j.parco.2013.06.003.
[47] G. Xie, X. Xiao, H. Peng, R. Li, K. Li, A survey of low-energy parallel scheduling algorithms, IEEE Trans. Sustain. Comput. 7 (1) (2022) 27–46, http://dx.doi.org/10.1109/TSUSC.2021.3057983.
[48] S. Safari, M. Ansari, H. Khdr, P. Gohari-Nazari, S. Yari-Karin, A. Yeganeh-Khaksar, S. Hessabi, A. Ejlali, J. Henkel, A survey of fault-tolerance techniques for embedded systems from the perspective of power, energy, and thermal issues, IEEE Access 10 (2022) 12229–12251, http://dx.doi.org/10.1109/ACCESS.2022.3144217.
[49] M. Cui, A. Kritikakou, L. Mo, E. Casseau, Near-optimal energy-efficient partial-duplication task mapping of real-time parallel applications, J. Syst. Archit. 134 (2023) 102790, http://dx.doi.org/10.1016/j.sysarc.2022.102790.
[50] I. Strumberger, N. Bacanin, S. Tomic, M. Beko, M. Tuba, Static drone placement by elephant herding optimization algorithm, in: 2017 25th Telecommunication Forum, TELFOR, 2017, pp. 1–4, http://dx.doi.org/10.1109/TELFOR.2017.8249469.
[51] E. Tuba, E. Dolicanin, M. Tuba, Chaotic brain storm optimization algorithm, in: H. Yin, Y. Gao, S. Chen, Y. Wen, G. Cai, T. Gu, J. Du, A.J. Tallón-Ballesteros, M. Zhang (Eds.), Intelligent Data Engineering and Automated Learning – IDEAL 2017, Springer International Publishing, Cham, 2017, pp. 551–559.
[52] M. Subotic, M. Tuba, N. Bacanin, D. Simian, Parallelized cuckoo search algorithm for unconstrained optimization, in: Proceedings of the 5th WSEAS Congress on Applied Computing Conference, and Proceedings of the 1st International Conference on Biologically Inspired Computation, BICA '12, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2012, pp. 151–156.
[53] A. Mohammadi, F. Sheikholeslam, S. Mirjalili, Nature-inspired metaheuristic search algorithms for optimizing benchmark problems: Inclined planes system optimization to state-of-the-art methods, Arch. Comput. Methods Eng. 30 (1) (2023) 331–389, http://dx.doi.org/10.1007/s11831-022-09800-0.
[54] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Tech. Rep. 200, Technical report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005, pp. 1–10.
[55] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE Comput. Intell. Mag. 1 (4) (2006) 28–39, http://dx.doi.org/10.1109/MCI.2006.329691.
[56] R. Rajabioun, Cuckoo optimization algorithm, Appl. Soft Comput. 11 (8) (2011) 5508–5518, http://dx.doi.org/10.1016/j.asoc.2011.05.008.
[57] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948, http://dx.doi.org/10.1109/ICNN.1995.488968.
[58] F. MiarNaeimi, G. Azizyan, M. Rashki, Horse herd optimization algorithm: A nature-inspired algorithm for high-dimensional optimization problems, Knowl.-Based Syst. 213 (2021) 106711, http://dx.doi.org/10.1016/j.knosys.2020.106711.
[59] A.H. Gandomi, A.H. Alavi, Krill herd: A new bio-inspired optimization algorithm, Commun. Nonlinear Sci. Numer. Simul. 17 (12) (2012) 4831–4845, http://dx.doi.org/10.1016/j.cnsns.2012.05.010.
[60] A. Askarzadeh, A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm, Comput. Struct. 169 (2016) 1–12, http://dx.doi.org/10.1016/j.compstruc.2016.03.001.
[61] S. Shadravan, H. Naji, V. Bardsiri, The sailfish optimizer: A novel nature-inspired metaheuristic algorithm for solving constrained engineering optimization problems, Eng. Appl. Artif. Intell. 80 (2019) 20–34, http://dx.doi.org/10.1016/j.engappai.2019.01.001.
[62] X. Chen, L. Cheng, C. Liu, Q. Liu, J. Liu, Y. Mao, J. Murphy, A WOA-based optimization approach for task scheduling in cloud computing systems, IEEE Syst. J. 14 (3) (2020) 3117–3128, http://dx.doi.org/10.1109/JSYST.2019.2960088.
[63] S. Mangalampalli, G.R. Karri, U. Kose, Multi objective trust aware task scheduling algorithm in cloud computing using whale optimization, J. King Saud Univ. - Comput. Inf. Sci. 35 (2) (2023) 791–809, http://dx.doi.org/10.1016/j.jksuci.2023.01.016.
[64] Z. Deng, D. Cao, H. Shen, Z. Yan, H. Huang, Reliability-aware task scheduling for energy efficiency on heterogeneous multiprocessor systems, J. Supercomput. 77 (2021) 11643–11681.
[65] M. Abdel-Basset, D. El-Shahat, K. Deb, M. Abouhawwash, Energy-aware whale optimization algorithm for real-time task scheduling in multiprocessor systems, Appl. Soft Comput. 93 (2020) 106349, http://dx.doi.org/10.1016/j.asoc.2020.106349.
[66] S. Goyal, S. Bhushan, Y. Kumar, A.u.H.S. Rana, M.R. Bhutta, M.F. Ijaz, Y. Son, An optimized framework for energy-resource allocation in a cloud environment based on the whale optimization algorithm, Sensors 21 (5) (2021) http://dx.doi.org/10.3390/s21051583.
[67] R. Ghafari, N. Mansouri, Cost-aware and energy-efficient task scheduling based on grey wolf optimizer, J. Mahani Math. Res. 12 (1) (2023) 257–288.
[68] B.V. Natesha, N. Kumar Sharma, S. Domanal, R.M. Reddy Guddeti, GWOTS: Grey wolf optimization based task scheduling at the green cloud data center, in: 2018 14th International Conference on Semantics, Knowledge and Grids, SKG, 2018, pp. 181–187, http://dx.doi.org/10.1109/SKG.2018.00034.
[69] N. Arora, R.K. Banyal, A particle grey wolf hybrid algorithm for workflow scheduling in cloud computing, Wirel. Pers. Commun. 122 (4) (2022) 3313–3345, http://dx.doi.org/10.1007/s11277-021-09065-z.
[70] F.A. Saif, R. Latip, Z.M. Hanapi, K. Shafinah, Multi-objective grey wolf optimizer algorithm for task scheduling in cloud-fog computing, IEEE Access 11 (2023) 20635–20646, http://dx.doi.org/10.1109/ACCESS.2023.3241240.
[71] N. Bacanin, T. Bezdan, E. Tuba, I. Strumberger, M. Tuba, M. Zivkovic, Task scheduling in cloud computing environment by grey wolf optimizer, in: 2019 27th Telecommunications Forum, TELFOR, 2019, pp. 1–4, http://dx.doi.org/10.1109/TELFOR48224.2019.8971223.
[72] R. Masadeh, A. Sharieh, B. Mahafzah, Humpback whale optimization algorithm based on vocal behavior for task scheduling in cloud computing, Int. J. Adv. Sci. Technol. 13 (3) (2019) 121–140.
[73] J.P.P.M. Sanaj MS, V. Alappatt, Profit maximization based task scheduling in hybrid clouds using whale optimization technique, Inf. Secur. J.: A Glob. Perspect. 29 (4) (2020) 155–168, http://dx.doi.org/10.1080/19393555.2020.1716116.
[74] N. Rana, M.S.A. Latiff, S.M. Abdulhamid, S. Misra, A hybrid whale optimization algorithm with differential evolution optimization for multi-objective virtual machine scheduling in cloud computing, Eng. Optim. 54 (12) (2022) 1999–2016, http://dx.doi.org/10.1080/0305215X.2021.1969560.
[75] K. Sreenu, M. Sreelatha, W-scheduler: whale optimization for task scheduling in cloud computing, Clust. Comput. 22 (2019) 1087–1098, http://dx.doi.org/10.1007/s10586-017-1055-5.
[76] A. Chhabra, S.K. Sahana, N.S. Sani, A. Mohammadzadeh, H.A. Omar, Energy-aware bag-of-tasks scheduling in the cloud computing system using hybrid oppositional differential evolution-enabled whale optimization algorithm, Energies 15 (13) (2022) http://dx.doi.org/10.3390/en15134571.
[77] L. Jia, K. Li, X. Shi, Cloud computing task scheduling model based on improved whale optimization algorithm, Wirel. Commun. Mob. Comput. 2021 (1) (2021) 4888154, http://dx.doi.org/10.1155/2021/4888154.
[78] N. Manikandan, N. Gobalakrishnan, K. Pradeep, Bee optimization based random double adaptive whale optimization model for task scheduling in cloud computing environment, Comput. Commun. 187 (2022) 35–44, http://dx.doi.org/10.1016/j.comcom.2022.01.016.
[79] V. Punyakum, K. Sethanan, K. Nitisiri, R. Pitakaso, Hybrid particle swarm and whale optimization algorithm for multi-visit and multi-period dynamic workforce scheduling and routing problems, Mathematics 10 (19) (2022) http://dx.doi.org/10.3390/math10193663.
[80] K. Pradeep, L.J. Ali, N. Gobalakrishnan, C.J. Raman, N. Manikandan, CWOA: Hybrid Approach for Task Scheduling in Cloud Environment, Comput. J. 65 (7) (2021) 1860–1873, http://dx.doi.org/10.1093/comjnl/bxab028.
[81] N. Manikandan, A. Pravin, Hybrid resource allocation and task scheduling scheme in cloud computing using optimal clustering techniques, Int. J. Serv. Oper. Informatics 10 (2) (2019) 104–121, http://dx.doi.org/10.1504/IJSOI.2019.103403.
[82] P. Albert, M. Nanjappan, WHOA: Hybrid based task scheduling in cloud computing environment, Wirel. Pers. Commun. 121 (3) (2021) 2327–2345, http://dx.doi.org/10.1007/s11277-021-08825-1.

[83] P. Gupta, S. Bhagat, D.K. Saini, A. Kumar, M. Alahmadi, P.C. Sharma, Hybrid whale optimization algorithm for resource optimization in cloud E-healthcare applications, Comput. Mater. Contin. 71 (3) (2022) 5659–5676, http://dx.doi.org/10.32604/cmc.2022.023056.

[84] S. Mangalampalli, G.R. Karri, G.N. Satish, Efficient workflow scheduling algorithm in cloud computing using whale optimization, Procedia Comput. Sci. 218 (2023) 1936–1945, http://dx.doi.org/10.1016/j.procs.2023.01.170.

[85] F.S. Gharehchopogh, H. Gholizadeh, A comprehensive survey: Whale optimization algorithm and its applications, Swarm Evol. Comput. 48 (2019) 1–24, http://dx.doi.org/10.1016/j.swevo.2019.03.004.

[86] 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, http://dx.doi.org/10.1109/71.993206.

[87] Z. Quan, Z.-J. Wang, T. Ye, S. Guo, Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 31 (5) (2020) 1165–1182, http://dx.doi.org/10.1109/TPDS.2019.2959533.

[88] G. Xie, X. Xiao, R. Li, K. Li, Schedule length minimization of parallel applications with energy consumption constraints using heuristics on heterogeneous distributed systems, Concurr. Comput.: Pr. Exp. 29 (16) (2017) e4024, http://dx.doi.org/10.1002/cpe.4024.

[89] N. Singh, H. Hachimi, A new hybrid whale optimizer algorithm with mean strategy of grey wolf optimizer for global optimization, Math. Comput. Appl. 23 (1) (2018) http://dx.doi.org/10.3390/mca23010014.

[90] S. Sandokji, F. Eassa, Dynamic variant rank HEFT task scheduling algorithm toward exascle computing, Procedia Comput. Sci. 163 (2019) 482–493, http://dx.doi.org/10.1016/j.procs.2019.12.131, 16th Learning and Technology Conference 2019Artificial Intelligence and Machine Learning: Embedding the Intelligence.

[91] G. Xie, G. Zeng, R. Li, K. Li, Energy-aware processor merging algorithms for deadline constrained parallel applications in heterogeneous cloud computing, IEEE Trans. Sustain. Comput. 2 (2) (2017) 62–75, http://dx.doi.org/10.1109/TSUSC.2017.2705183.

[92] K. Deep, H. Mebrahtu, Combined mutation operators of genetic algorithm for the travelling salesman problem, Int. J. Comb. Optim. Probl. Informatics 2 (3) (2011) 1–23.

[93] M. Abdel-Basset, G. Manogaran, D. El-Shahat, S. Mirjalili, RETRACTED: A hybrid whale optimization algorithm based on local search strategy for the permutation flow shop scheduling problem, Future Gener. Comput. Syst. 85 (2018) 129–145, http://dx.doi.org/10.1016/j.future.2018.03.020.

[94] A. Kumar, S. Das, I. Zelinka, A self-adaptive spherical search algorithm for real-world constrained optimization problems, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 13–14, http://dx.doi.org/10.1145/3377929.3398186.

[95] A. Kumar, S. Das, I. Zelinka, A modified covariance matrix adaptation evolution strategy for real-world constrained optimization problems, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 11–12, http://dx.doi.org/10.1145/3377929.3398185.

[96] K.M. Sallam, S.M. Elsayed, R.K. Chakrabortty, M.J. Ryan, Improved multi-operator differential evolution algorithm for solving unconstrained problems, in: 2020 IEEE Congress on Evolutionary Computation, CEC, 2020, pp. 1–8, http://dx.doi.org/10.1109/CEC48606.2020.9185577.

[97] J. Gurrola-Ramos, A. Hernàndez-Aguirre, O. Dalmau-Cedeño, COLSHADE for real-world single-objective constrained optimization problems, in: 2020 IEEE Congress on Evolutionary Computation, CEC, 2020, pp. 1–8, http://dx.doi.org/10.1109/CEC48606.2020.9185583.

[98] A. Kumar, G. Wu, M.Z. Ali, R. Mallipeddi, P.N. Suganthan, S. Das, A test-suite of non-convex constrained optimization problems from the real-world and some baseline results, Swarm Evol. Comput. 56 (2020) 100693, http://dx.doi.org/10.1016/j.swevo.2020.100693.

[99] R.F. Woolson, Wilcoxon signed-rank test, in: Wiley Encyclopedia of Clinical Trials, John Wiley & Sons, Ltd, 2008, pp. 1–3, http://dx.doi.org/10.1002/9780471462422.eoct979.

[100] SourceForge, Task graph generator, 2015, URL https://sourceforge.net/projects/taskgraphgen/.