Damping-Assisted Evolutionary Swarm Intelligence for Industrial IoT Task Scheduling in Cloud Computing

Ahmed G. Gad, *Member, IEEE*, Essam H. Houssein, *Member, IEEE*, MengChu Zhou, *Fellow, IEEE*, Ponnuthurai Nagaratnam Suganthan, *Fellow, IEEE*, and Yaser M. Wazery

***Abstract*—Advancements in the Industrial Internet of Things (IIoT) have yielded massive volumes of data, taxing the capabilities of cloud computing infrastructure. Allocating limited computing resources to numerous incoming requests is crucial for cloud computing and referred to as a task-scheduling-in-cloud-computing (TSCC) problem. In order to ameliorate the performance of a particle swarm optimizer (PSO) and broaden its application to TSCC, this article introduces an opposition-based simulated annealing particle swarm optimizer (OSAPSO) to address PSO's premature convergence issue, particularly when tackling high-dimensional complex problems like TSCC. OSAPSO is a novel combination of opposition-based learning (OBL), evolution strategy, simulated annealing (SA), and swarm intelligence. At its initial stage, a swarm is formed at random by using OBL to guarantee its diversity with a light computational burden. A multiway tournament selection approach is then utilized to pick parents to produce a new offspring swarm by using two novel evolutionary operators, namely, damping-based mutation and inversion–scrambling-based crossover. OSAPSO is given a powerful exploration capacity by adopting the survivor probabilistic selection of SA, which accepts subpar solutions with a certain probability. Finally, PSO itself kicks in, making a good tradeoff between solution diversity and convergence speed of OSAPSO. Due to the nonconvex discontinuous nature of TSCC, OSAPSO is modified to clone it into a discrete optimization problem. Within a heterogeneous cloud computing environment, OSAPSO and eight well-regarded competitors are examined on a set of multiscale IIoT heterogeneous task groups. In terms of power consumption, monetary cost, service makespan, and system throughput, experimental results reveal that OSAPSO beats its peers in IIoT task scheduling of cloud systems.**

Manuscript received 19 May 2023; accepted 25 June 2023. Date of publication 3 July 2023; date of current version 25 December 2023. This work was supported in part by the Fundo para o Desenvolvimento das Ciencias e da Tecnologia (FDCT) under Grant 0047/2021/A1. *(Corresponding authors: Ahmed G. Gad; MengChu Zhou; Ponnuthurai Nagaratnam Suganthan.)*

Ahmed G. Gad is with the Faculty of Computers and Information, Kafrelsheikh University, Kafrelsheikh 33516, Egypt (e-mail: ahmed.gad@fci.kfs.edu.eg).

Essam H. Houssein and Yaser M. Wazery are with the Faculty of Computers and Information, Minia University, Minia 61519, Egypt (e-mail: essam.halim@mu.edu.eg; yaser.wazery@minia.edu.eg).

MengChu Zhou is with the Macao Institute of Systems Engineering and Collaborative Laboratory for Intelligent Science and Systems, Macau University of Science and Technology, Macau, China, and also with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: zhou@njit.edu).

Ponnuthurai Nagaratnam Suganthan is with the KINDI Center for Computing Research, College of Engineering, Qatar University, Doha, Qatar (e-mail: p.n.suganthan@qu.edu.qa).

This article has supplementary downloadable material available at https://doi.org/10.1109/JIOT.2023.3291367, provided by the authors.

Digital Object Identifier 10.1109/JIOT.2023.3291367

***Index Terms*—Cloud task scheduling, evolutionary computation, Industrial Internet of Things (IIoT), particle swarm optimizer (PSO), power consumption, simulated annealing (SA), swarm intelligence (SI), system throughput.**

I. Introduction

THE CLOUD computing paradigm (CCP) has radically altered the computing sector by relieving users/consumers of the burden of operating their own, sometimes costly, information technology infrastructure [1], [2]. CCP's infrastructure has elastic, reliable, and cost-effective computing resources to accommodate millions of physical machines, storage devices, network equipment, and cooling facilities [3]. These resources are shared across global Industrial-Internet of Things (IIoT) users to execute a range of heterogeneous deadline-constrained applications, such as scientific computing, high-performance simulation, big data analysis, and e-commerce [4]. Due to the massive amount of data generated by the sensing surrounding environment of these applications, CCP's electric energy usage has increased significantly. In the U.S. alone, CCP's data centers used energy during 2020 that was about equal to that of 50 large power plants [5]. Researchers, especially environmentalists, are increasingly concerned about such massive energy use. Due to the rising pressure from the governments, society, and media, energy-efficient CCP is fast being deployed globally and is equipped with energy facilities and lowered running costs. For instance, Apple Inc. has finished a 100-acre solar farm close to the CCP of iCloud in North Carolina, U.S., which is energy efficient. Every year, it delivers about 84 million kWh of energy [6].

When IIoT and cloud computing are integrated, new difficulties emerge. According to a recent report by information handling services (IHSs) Markit, the number of connected Internet of Things (IoT) devices globally is expected to increase by 12% on average every year, from almost 27 billion in 2017 to 125 billion in 2030 [7]. With this exponential growth in the number of IoT linked devices, CCP with its typical centralized processing features (in which computational resources are aggregated and stored in numerous data centers) would not be capable of fulfilling the requirements of IIoT heterogeneous applications. The prime factor is the enormous physical separation between IoT devices and the cloud. The congestion, especially at bottlenecks, is expected

as a result of the enormous amount of data that IoT devices are sending to the cloud through the Internet; and network speed and bandwidth are also to be impacted. Since latency sensitivity is one of the aspects of IIoT applications, transmission delays yield reduced Quality of Service (QoS), thus leaving negative impact on an end user's experience. Due to IIoT's widespread acceptance and deployment, tasks for IIoT heterogeneous applications in CCP surge, as do the energy consumption and expenses required by cloud service providers (CSPs). According to [8], energy-related expenditures account for 41.6% of CCP's overall operational expenses.

Consequently, CCP must perform the energy-efficient and cost-effective scheduling of numerous IIoT heterogeneous applications while meeting customer demands for a dependable and prompt cloud service. Still, it has certain attributes that make such scheduling particularly difficult. First, CCP is often dispersed among several sites with a wide geographical range of energy availability and power grid prices. Second, every single data center often links to several CSPs in order to transmit data from consumers throughout the world. Indeed, CCP interchanges vast amounts of data while incurring significant monetary expenditures, execution time, and energy consumption from data storage, computational processing, and data transmission over a network. Third, while tightly enforcing deadline constraints, a typical CCP must execute energy-efficient and cost-time-effective scheduling of many IIoT heterogeneous tasks [9]. Numerous research from the academic and industrial communities [10], [11] strive to reduce the amount of energy used in CCP by scheduling tasks judiciously while taking expenses, makespan, and throughput into consideration. In [10], energy consumption is minimized by accounting for the variety of power grid prices. However, the data from IIoT applications and the available infrastructure resources from various CSPs are not examined for heterogeneity. Shabestari et al. [11] presented an optimization method for big data scheduling problem to minimize power consumption while executing the applications in a timely fashion. Their method offers a generic scheduling model, but it ignores the type and size of applications. Furthermore, the heterogeneity of cloud infrastructure resources and variations of their capabilities are not considered. Therefore, it is challenging to minimize energy consumption, monetary cost, and execution time in CCP while taking into account disparities in data storage, computational processing capacity, network bandwidth, and task completing deadline. A task schedule, which satisfies these conditions, fulfils the service level agreement (SLA) signed with users. Task scheduling in CCP is designed to assist both CSPs and end users. Energy efficiency, resource utilization, system throughput, and load balancing are all substantial for CSPs. As to the end users, their concerns span security, deadline, budget, and makespan.

Metaheuristic algorithms (MHAs) are useful approaches for locating approximations of solutions due to their simplicity, flexibility, and potentiality to avoid local optima traps [12]. Popular MHAs include particle swarm optimizer (PSO) [13], simulated annealing (SA) [14], differential evolution (DE) [15], and genetic algorithm (GA) [16]. Whale optimization algorithm (WOA) [17] and gray wolf optimizer (GWO) [18] are two examples of newly proposed algorithms, which, compared against the well-known algorithms above, show competitive performances. Many studies have generally dealt with task-scheduling-in-cloud-computing (TSCC) problems in CCP by using different methods. Tran et al. [19] employed machine learning and fuzzy techniques to predict system workloads for making scaling resource decisions properly. Meanwhile, game and queuing theories are applied to the scheduling problems [20]. Furthermore, Agarwal and Srivastava [21], Singh et al. [22], and Mangalampalli et al. [23] presented the approaches to distribute incoming requests within a typical CCP by using PSO, SA, and GWO methods. Overall, whereas these studies can achieve satisfactory results, their main flaw is that they only satisfy the needs of CSPs or end users at once, rather than both. A good scheduling solution should have the potency to get both sides satisfied. Since multiobjective optimization is often much more difficult than single-objective one, most of researchers have preferred to formulate those criteria into a single-objective. Due to high demand from real-world applications, multiobjective optimization remains to be an active area of research [24].

Among many MHAs, PSO is a formidable nature-inspired MHA under the swarm intelligence (SI) umbrella and can be effectively used for solving challenging multiobjective optimization problems [13]. Due to its global optimization ability and calibratable parameter setting, it has been successfully applied in different industrial applications and multidisciplinary scientific and engineering research areas, including feature selection [25], image segmentation [26], static and dynamic clustering [27], deep neural network research [28], multiobjective control [29], and many others. In view of this, high-performance solutions to TSCC problems may be derived very effectively by using PSO [30], [31], [32], [33].

Although advancements reported in the literature improve the optimization efficiency of PSO, they run the risk of slipping into local optima traps owing to premature convergence [13]. A TSCC problem is nonlinear and complex and tends to have many locally optimal solutions. Additionally, the "No Free Lunch" theorem [34] for optimization states that there is no universal algorithm that guarantees constant effective performance with all problems [35]. Furthermore, the existence of a challenging NP-complete problem like TSCC, which has no definite solution, is another drive for developing a robust scheduling approach to tackle this type of problem [36]. Namely, we develop a novel MHA with a robust global optimization capability, called opposition-based SA PSO (OSAPSO), in order to effectively relieve the foregoing difficulties and broaden PSO utilization for TSCC to pursue the most effective, instantaneous, near-optimal solutions. From the existing literature, we observe that hybrid optimization algorithms, especially PSO-integrated ones, are immensely popular in this domain that have been steadily performing well [13], [33], [37], [38], [39]. In the original PSO, the swarm candidates mainly guide the position updating process, which does not allow full utilization of the swarm information and whole search space. Meanwhile, PSO's initial swarm is formed at random, which makes it impossible to assure the initial swarm's quality. Therefore, the mechanisms

of swarm initialization and position updating in OSAPSO are updated as follows. First, by utilizing the whole swarm's information and opposite solutions, a partial opposition-based learning (POBL) strategy is embedded into OSAPSO's swarm initialization procedure to partially expand search space of the initial swarm in the opposite direction, thereby increasing its probability of finding the global optimum and ability to avoid becoming trapped into local optima, with less computational complexity. By searching the neighborhood space's position information, an evolution strategy (ES) is utilized in OSAPSO's position updating mechanism for swarm reproduction, which greatly preserves both local search capability and swarm diversity by learning from elite particles. The survivor probabilistic selection procedure of SA, named herein an objective evaluation strategy (OES), is rendered to equip OSAPSO with a greedy search capability by chasing superior solutions while, with a certain probability, accepting inferior solutions, within the global solution's regions discovered by OSAPSO. Finally, advantageous information sharing capability of PSO is utilized to boost global search ability for robust global optimization.

As per distinct environmental circumstances in terms of the number of nodes and the size of task sets, OSAPSO's optimization performance is investigated on two different scenarios of heterogeneous cloud systems with four multiscale cloud datasets of 20 different TSCC instances. As far as we know, there is no work in this research area that comprehensively covers the four datasets. When compared to some state-of-the-art optimization algorithms, including the standard PSO and SA, comprehensive learning particle swarm optimization (CLPSO) [40], chaos particle swarm optimization (CPSO) [41], adaptive DE (JADE) [42], linear population size reduction success-history adaptive DE (LSHADE) [43], WOA, and GWO, OSAPSO is found to be effective at solving TSCC problems with superior final objective values and decent convergence speed in the early stage of optimization.

The novel contributions of this article are fourfold.

1) Proposing OSAPSO to elevate PSO's performance by introducing: a) a POBL strategy to reinforce the global search ability and heighten the initial swarm's quality; b) an ES strategy injected by SA's damping characteristic and a novel, equiprobable-switching crossover mechanism, in order to maintain swarm diversity and local search ability; and c) a greedy OES strategy to enable OSAPSO to chase superior solutions while, with a certain probability, accepting inferior solutions to provide a good exploration–exploitation balance.
2) Formulating TSCC as a constrained multiobjective combinatorial optimization problem, which considers power consumption, monetary cost, service makespan, and system throughput while fulfilling task deadlines.
3) Evaluating OSAPSO for effectiveness and practicality to solve TSCC problems by comparing it to several well-regarded methods on two distinct scenarios of cloud systems with four multiscale cloud datasets (synthetic and realistic) of 20 different TSCC instances.
4) Modeling and simulating a *heterogeneous* cloud computing platform to test task scheduling strategies for IIoT *heterogeneous* applications.

Section II introduces OSAPSO and TSCC problems. The details of OSAPSO are explained in Section III. Its comparison results with its peers are shown in Section IV. Section V concludes this article.

II. Problem Formulation and Synopses

A. System Architecture

In CCP, processing occurs within cloud nodes (e.g., workstation server, virtual machines, and high-performance computing servers) to satisfy IIoT user's needs. Despite substantial latency issues, the cloud remains indispensable due to its incredibly powerful computing. Our system model is assumed to consist of a set of cloud nodes, as well as a cloud broker. All IIoT user's requests sent via various IoT devices, including smart sensors and wearables, and responses replied from cloud nodes, are forwarded to the broker, where all tasks submitted to the cloud system are analyzed, estimated, and then scheduled. Since the broker and cloud nodes are to be close to each other by assuming their inclusion both in the cloud system with multiple brokers highly deployed, the delay for data communication between broker and nodes can be acceptably neglected. In order to maintain CCP's high performance, our OSAPSO is installed on the broker so as to find the most favorable task executing schedule that achieves high energy efficiency, cost–time effectiveness, and throughput. To maintain system stability and reliability, OSAPSO is regularly run at the broker to bulk and schedule a set of incoming tasks, which have already been submitted at different times during the previous run of the algorithm. Our future work intends to consider immediate (dynamic) scheduling and precedence executing of tasks generated from critical applications, e.g., medical diagnosis and self-driving vehicles, which cannot wait until the next run of the algorithm.

A potential single-point-of-failure problem may disturb the system. First, we assume that IIoT users can connect to a cloud broker via multiple links, and so there are multiple links among system nodes. Thus, the cloud broker has more than one route to deliver a task to its respective node. At the worst, when there is an inevitable failure to deliver a specific task to its allocated node via any of the available routes, or the allocated node itself is down, OSAPSO continues to search other solutions until finding a more decent node for that task. Although this issue is not explicitly considered in our simulation as CSPs already take care of it by providing backup links, it is worthy considering as future work. The operational sequence of the proposed system is shown in Fig. 1.

B. Task Scheduling Problem Statement

The cloud system handles data processing requests from IIoT heterogeneous applications to be executed across the cloud computing infrastructure, posterior to decomposing them into small, independent tasks, each with attributes, including the number of instructions, I/O file size, memory required, and completion deadline. Suppose that T_k refers to an IIoT task k. A set of n independent tasks are received each time in the system and expressed as $\mathbf{T} = \{T_1, T_2, T_3, \ldots, T_n\}$. Assume that there exist m heterogeneous processors (cloud nodes) composing the cloud computing infrastructure. They

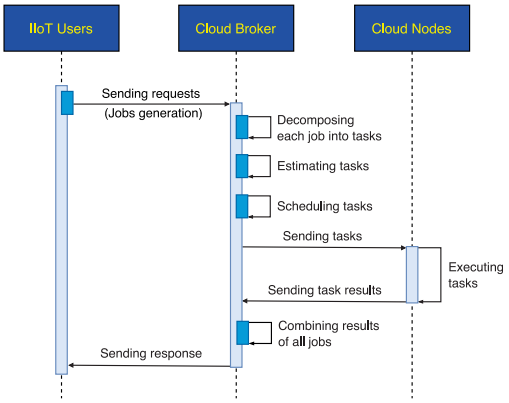


Fig. 1. Task scheduling flow within the cloud system architecture.

have typical properties like processing capacity, data transmission delay, and monetary cost and power consumption from computation, storage, and bandwidth usage. If the cloud processing node numbered i is represented by N_i, then a set of m processors can be denoted as $\mathbf{N} = \{N_1, N_2, N_3, \ldots, N_m\}$. Thus, for all IIoT tasks $\mathbf{T}$ and cloud nodes $\mathbf{N}$, each task T_k is to be scheduled (assigned) to a node N_i, which is represented by $\mathcal{S}^i = \{T_x^i, T_y^i, \ldots, T_z^i\}$, in which a set of one or more tasks can be scheduled to the same processor for execution. In sum, TSCC can be described as looking for a scheduling scheme $\mathcal{S} = \{T_1^a, T_2^b, T_3^c, \ldots, T_n^p\}$. To effectively finish all tasks within cloud nodes, our task scheduling target takes into account the following influence factors: power consumption, monetary cost, service makespan, and system throughput.

1) Power Consumption: $\mathcal{P}$ denotes the total power consumed for completing all tasks. We need to consider power consumption from each cloud node N_i for data transmission from IIoT users to N_i and vice versa ($\mathcal{P}_{1i}$), storage data on N_i ($\mathcal{P}_{2i}$), execution at N_i ($\mathcal{P}_{3i}$), as well as the power consumption from N_i in the idle mode (P_{ji}^0) for each corresponding activity $j \in \{1, 2, 3\}$. We have

$$\mathcal{P} = \sum_{i=1}^{m} \mathcal{P}_{1i} + \mathcal{P}_{2i} + \mathcal{P}_{3i} \tag{1}$$

where

$$\mathcal{P}_{1i} = P_{1i}^0 + \frac{\sum_{T_k \in \mathcal{S}^i} S_I(T_k) + S_O(T_k)}{P_{1i}}$$
$$\mathcal{P}_{2i} = P_{2i}^0 + \frac{\sum_{T_k \in \mathcal{S}^i} S_I(T_k) + S_O(T_k)}{P_{2i}}$$
$$\mathcal{P}_{3i} = P_{3i}^0 + \frac{\sum_{T_k \in \mathcal{S}^i} L(T_k)}{P_{3i}}$$

with $S_I(T_k)$ and $S_O(T_k)$ being the sizes of, respectively, input and output files of an IIoT task T_k; P_{1i} and P_{2i} being the powers required to, respectively, transfer and store one data unit from an IIoT user to/on N_i; and P_{3i} being the power required to execute one instruction unit at N_i.

2) Monetary Cost: When a cloud system processes a task, a constant fee must be paid for ingress and egress data transmission ($\mathcal{C}_{1i}$), data storage ($\mathcal{C}_{2i}$), data processing ($\mathcal{C}_{3i}$), and memory usage ($\mathcal{C}_{4i}$) for an allocated node N_i. The total cost $\mathcal{C}$ when all nodes processes all tasks assigned to them is

$$\mathcal{C} = \sum_{i=1}^{m} \mathcal{C}_{1i} + \mathcal{C}_{2i} + \mathcal{C}_{3i} + \mathcal{C}_{4i} \tag{2}$$

where

$$\mathcal{C}_{1i} = C_{1i}^0 + \frac{\sum_{T_k \in \mathcal{S}^i} S_I(T_k) + S_O(T_k)}{C_{1i}}$$
$$\mathcal{C}_{2i} = C_{2i}^0 + \frac{\sum_{T_k \in \mathcal{S}^i} S_I(T_k) + S_O(T_k)}{C_{2i}}$$
$$\mathcal{C}_{3i} = C_{3i}^0 + \frac{\sum_{T_k \in \mathcal{S}^i} L(T_k)}{D_{2i} C_{3i}}$$
$$\mathcal{C}_{4i} = C_{4i}^0 + \frac{\sum_{T_k \in \mathcal{S}^i} S_M(T_k)}{C_{4i}}$$

with C_{ji}^0 being the cost required for each corresponding activity $j \in \{1, 2, 3, 4\}$ for the entire time when N_i is in idle mode; $L(T_k)$ being the length (the number of instructions) of task T_k; and D_{2i} being the delay of computation of N_i, which is estimated via instruction level parallelism, number of cores, clock rate, etc. S_M is the memory size needed for T_k. C_{1i} and C_{2i} are the average costs to, respectively, transfer and store one data unit to/on N_i. C_{3i} and C_{4i} are the average costs of, respectively, processing one instruction unit at N_i and utilizing memory required by T_k.

3) Service Makespan: Given the maximum time period of transmission (τ_{1i}) and execution time (τ_{2i}) among all nodes, we have the total time needed for all tasks to be completed, defined from the moment of submitting the first task's request to that of completing the last task and returning its results, or the last machine (node) is unloaded

$$\mathcal{M} = \max_{i=1}^{m} \tau_{1i} + \tau_{2i} \tag{3}$$

where τ_{1i} is the transmission time required to transfer the ingress and egress data of all tasks assigned to N_i, i.e.,

$$\tau_{1i} = \frac{\sum_{T_k \in \mathcal{S}^i} S_I(T_k) + S_O(T_k)}{D_{1i}} \tag{4}$$

with D_{1i} being the average delay to transmit one data unit from an IIoT user to N_i. The execution time required by N_i to complete all tasks assigned to it is

$$\tau_{2i} = \frac{\sum_{T_k \in \mathcal{S}^i} L(T_k)}{D_{2i}}. \tag{5}$$

4) System Throughput: The throughput of instructions executed on all nodes of the cloud system per time unit is

$$\eta = \frac{\sum_{k=1}^{n} L(T_k)}{\sum_{i=1}^{m} \tau_{2i}}. \tag{6}$$

5) Overall Objective Function and Constraint:

$$F = w_1 \mathcal{P} + w_2 \mathcal{C} + w_3 \mathcal{M} + \frac{w_4}{\eta} \tag{7}$$

$$\text{s.t.} \quad \varepsilon(T_k) \le \Gamma(T_k) \quad \forall T_k \in \mathcal{S}^i, i \in \{1, 2, 3, \ldots, m\} \tag{8}$$

with w_1–w_4 being the balance coefficients among $\mathcal{P}$, $\mathcal{C}$, $\mathcal{M}$, and η, where $\sum_{i=1}^{4} w_i = 1$, $w_i \in [0, 1]$, $i \in \{1, 2, 3, 4\}$.

Equation (8) is a constraint function that each task T_k's completion time $\varepsilon(T_k)$ meets its completion deadline $\Gamma(T_k)$. Two more constraints are as follows.

1) A node can perform one task only at a time.
2) Each task is assigned to only one node.

A TSCC problem can be mathematically modeled as a constrained multiobjective minimization problem as follows.

Input:

$\mathbf{T} = \{T_1, T_2, T_3, \ldots, T_n\}$: a set of IIoT tasks.

$\mathbf{N} = \{N_1, N_2, N_3, \ldots, N_m\}$: a set of cloud nodes.

Output:

$\mathcal{S} = \{T_1^a, T_2^b, T_3^c, \ldots, T_n^p\}$: an assignment of all tasks to nodes.

Objective:

Minimize the objective function in (7), subject to all constraints.

C. Conventional Particle Swarm Optimization

The biological behavior of fish schooling and bird flocking has been analogized to propose the SI-based particle optimization algorithm for particularly solving continuous nonlinear problems [13]. In the PSO process, a swarm of particles initially forms as a set of random solutions scattered across search space. Then, each particle is updated in each iteration for its velocity and position, according to

$$v_{i,j}^{t+1} = \omega v_{i,j}^t + c_1 r_{1_{i,j}}^{t+1}\left(p_{\text{best}_i,j}^t - x_{i,j}^t\right) + c_2 r_{2_{i,j}}^{t+1}\left(g_{\text{best},j}^t - x_{i,j}^t\right) \tag{9}$$

$$x_{i,j}^{t+1} = x_{i,j}^t + v_{i,j}^{t+1} \tag{10}$$

where $v_{i,j}^t$ and $x_{i,j}^t$ stand for the jth dimension of, respectively, velocity and position vectors of particle i at current iteration t. c_1 and c_2 are positive acceleration coefficients to control the influence of $\mathbf{p}_{\text{best}}$ and $\mathbf{g}_{\text{best}}$ on the search process. ω is the inertia weight. $r_{1_{i,j}}$ and $r_{2_{i,j}}$ are two independent random numbers selected from samples uniformly distributed over the interval [0, 1]. $p_{\text{best}_i,j}^t$ denotes the personal best position historically found by particle i to t, whereas $g_{\text{best},j}^t$ represents the entire swarm's global best position ever to t. On each iteration, $p_{\text{best}_i,j}^t$ and $g_{\text{best},j}^t$ are updated according to exact function evaluations for all the particles. PSO typically terminates when a stop criterion is met, and the global best position $\mathbf{g}_{\text{best}}$ is finally exported.

D. Conventional Simulated Annealing

SA is a metaheuristic trying to find the globally optimal solution for an optimization problem, often used in conjunction with other MHAs to overcome its slow convergence problem [44]. The annealing process of metals is basically imitated in SA, where the heat or temperature T steadily reduces based on a cooling rate

$$\Delta T = \left(T_f/T_0\right)^{\frac{1}{t+1}} \tag{11}$$

with T_0 and T_f being, respectively, the initial and final temperatures.

SA is usually initialized with a random single solution and repeatedly searches through the neighborhood of current solution, pursuing a new, better solution. Far from that a neighbor solution is always accepted if it is fitter, if that solution incidentally has a fitness worse than the current solution, it is accepted by SA with a certain probability

$$p = e^{\frac{-\theta}{\Delta T}} \tag{12}$$

where θ is the absolute difference between fitness values of the current solution and the previous neighbor solution. Accepting a worse solution could boost an algorithm's capability to avoid entrapment into local optima. This procedure is performed continually until a specified termination criterion for T is hit.

E. Opposition-Based Learning

In the group SI optimization algorithms, both convergence acceleration and search efficiency are directly affected by swarm diversity. Some scholars prefer chaotic mapping to initialize the swarm to promote diversity [45], given that initializing the swarm with the chaotic algorithm helps it jump out of the local optima traps. However, this way has a disadvantage due to the close contact with the proximity and the high randomness and uncertainty faced, which impedes equilibrium if the iteration encounters an unstable point. Reverse (Opposition-based) learning in the early initialization of solution space makes it even random, ergodic, and regular [46]. Furthermore, in the actual search process, if the initial swarm generates opposite solutions for the whole individuals, then the iterative search will only take an extended period of time. Thus, this article proposes using biased (l-best) reverse knowledge for swarm initialization by applying the classical opposition-based learning (OBL) theory to only a proportion l of the initial swarm, where $0 < l < 1$. On swarm initialization, the forward (original) solution $x_{i,j}$ and reverse (opposite) solution $\bar{x}_{i,j}$ are mathematically formulated as

$$x_{i,j} \sim U\left(\check{j}, \hat{j}\right), \tag{13}$$

$$\bar{x}_{i,j} = \check{j} + \hat{j} - g_{\text{best},j} + r_j\left(g_{\text{best},j} - x_{i,j}\right) \tag{14}$$

where $\check{j}$ and $\hat{j}$ are, respectively, the demarcated lower and upper bounds of the values permitted in each dimension j. $r_j \sim U(0, 1)$ is a random value in the jth dimension.

III. Evolutionary Swarm Intelligence for TSCC

Our proposed algorithm is primarily concerned with mapping a huge number of incoming tasks into available limited-resource nodes in accordance with adaptable time. PSO generates a set of individuals (target vectors), which comply with different evolutionary operators to reaching a suitable (near-optimal) schedule, which enables that all tasks are intelligently scheduled to be executed within a minimized time interval. The final output to tackling a TSCC problem is often an array of solutions with assignment of tasks into proper nodes.

First, in PSO, the swarm is often initialized at random. This way is clearly disadvantageous to swarm diversity, particularly in the early stage of search, i.e., this blindness to search at the beginning does not ensure the quality of initial swarm and eventually diminishes the algorithm capacity to scrutinize whole search space. Furthermore, PSO does not fully utilize the search space and swarm information, and particle position is updated only around particles $\mathbf{p}_{\text{best}}$ and $\mathbf{g}_{\text{best}}$. If these

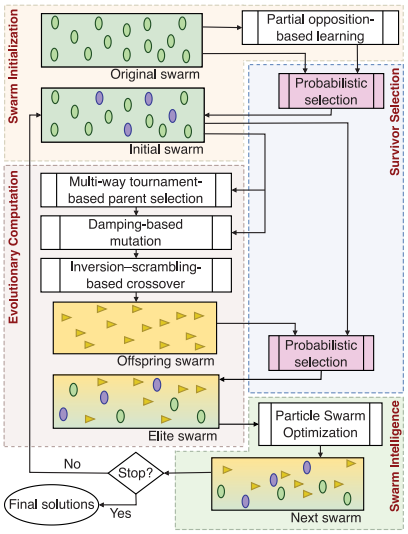


Fig. 2. Framework of OSAPSO.

two particles are trapped into local space extremes, it is highly likely that the remaining particles behave the same, causing the algorithm to experience "stop searching" behavior, hence premature convergence, due to the excessive loss of exploration and swarm diversity.

This work strives to tackle aforementioned PSO's issues by suggesting the following improvement strategies:

1) The swarm is initialized by applying partial reverse learning (i.e., POBL) to the top l-best particles, in order to reinforce the uniformity of swarm distribution, enhance swarm diversity, and alleviate search blindness at the beginning, thus boosting the overall performance of OSAPSO.
2) The evolutionary operators guide particles to move toward the feasible regions by capturing the optimal solution when individuals behave aggressively near the extremes, or jumping out of the local optima to explore solution space for more viable areas.
3) Particles exchange information about their own best positions found so far with the global optimal individual in each iteration, thus guiding the swarm to the optimum.
4) The survivor selection strategy of SA is incorporated to pursue an optimum within the global search regions identified by POBL and ES, providing OSAPSO with avaricious exploratory behavior.

The latter three strategies (i.e., evolution and survival) are repeated until a preset stopping criterion is met. Algorithm 1 in the Supplementary File realizes OSAPSO shown in Fig. 2.

A. Particle: Position Encoding and Velocity

The representation step is crucial to a PSO's successful design, which aims at finding a proper mapping between PSO particles (in its ordinary continuous form) and problem solutions. In OSAPSO, discrete representation is used for particle encoding, i.e., each particle is denoted an n-dimensional discontinuous vector, called a position vector, corresponding to n tasks. With m being the number of cloud nodes, the discrete position vector $\check{\mathbf{x}}_i$ of the ith particle is represented

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
Continuous	0.23	-0.45	-0.83	0.98	-1.00	-0.60	0.85	0.00	0.75	0.92
Normalized	0.62	0.28	0.09	1.00	0.00	0.20	0.93	0.51	0.88	0.97
Scaled	2.24	1.56	1.17	3.00	1.00	1.40	2.87	2.01	2.77	2.94
Discrete	2	2	1	3	1	1	3	2	3	3

Fig. 3. Example of a position vector in a succession of forms.

as $\check{x}_{i,j} \in \{1, 2, \ldots, m\}$, $\forall i \in \{1, 2, \ldots, N\}$, $j \in \{1, 2, \ldots, n\}$, where N is swarm size and each element j in the position vector contains an integer value g ranging from 1 to m, indicating that the corresponding task is allocated to node g. If $x_{i,j} = 3$, then task T_j is to be executed at node 3. A normalization and scaling technique is applied. First, each element j is normalized to a real number in the range [0, 1] as

$$\tilde{x}_{i,j} = \frac{x_{i,j} - \min_{j=1}^{n} x_{i,j}}{\max_{j=1}^{n} x_{i,j} - \min_{j=1}^{n} x_{i,j}}. \tag{15}$$

Then, the normalized vector $\tilde{\mathbf{x}}_i$ is scaled and rounded (discretized), for each element j, to one of $\{1, 2, \ldots, m\}$ as

$$\check{x}_{i,j} = \text{round}\big(1 + \tilde{x}_{i,j}(m - 1)\big). \tag{16}$$

For instance, if a set of ten tasks must be executed within a three-node cloud system, a potential schedule can be:

$$\mathcal{S} = \left\{T_1^2, T_2^2, T_3^1, T_4^3, T_5^1, T_6^1, T_7^3, T_8^2, T_9^3, T_{10}^3\right\}.$$

Node 1 is responsible for executing the task subsequence {3, 5, 6}, the task subsequence {1, 2, 8} is assigned to node 2, whereas node 3 executes {4, 7, 9, 10}. Since performing tasks in a different sequence at a node has no effect on the objectives to be optimized, the order of tasks to be processed is ignored. Fig. 3 shows an example of a particle's position vector (or a potential schedule) expressing the above solution in different representations.

Each particle utilizes its own velocity to move to a new location. The velocity vector's dimensions are identical to the continuous position vector (i.e., an n-dimensional vector), and its elements have real numbers clamped in $[\check{j}, \hat{j}]$. Overall, we must consistently ensures that $x_{i,j}$, $v_{i,j} \in [\check{j}, \hat{j}]$ $\forall i \in \{1, 2, \ldots, N\}$, $j \in \{1, 2, \ldots, n\}$.

B. Fitness Evaluation

An objective function is used to evaluate the quality of a schedule scheme that a particle expresses and its attained value is referred herein to as an exact objective value (EOV). In our objective function considers power consumption, monetary cost, service makespan, and system throughput. Its exact definition is given in (7). The lower the F's value, the more superior the solution obtained. In the following procedures, the swarm individuals' sorting via EOV is adopted in OSAPSO particularly to maintain its greedy behavior in order to consistently ensure obtaining the extremes.

1) Personal Best and Global Best: Various locations are explored by particles through many movements. $\mathbf{p}_{\text{best}_i}^t$ determines the personal best position that the ith particle has experienced to current iteration t, whereas $\mathbf{g}_{\text{best}}^t$ represents the global best location discovered by any of the swarm's individuals to t. Thus, both are n-dimensional.

$\mathbf{p}_{\text{best}}$ and $\mathbf{g}_{\text{best}}$ are two crucial solutions in the PSO process, which assist all particles, at every time step t, to effectively update their current position by searching around both their own best position and the global best one, respectively. Every particle i in the swarm regularly moves to a new location $\mathbf{x}_i^{t+1}$. If it achieves a less EOV, its personal best position $\mathbf{p}_{\text{best}_i}^{t+1}$ is substituted by $\mathbf{x}_i^{t+1}$. If $\mathbf{p}_{\text{best}}^{t+1}$ has an EOV less than $\mathbf{g}_{\text{best}}^t$, $\mathbf{g}_{\text{best}}^{t+1}$ is substituted by $\mathbf{p}_{\text{best}}^{t+1}$. This process is realized and solely imposed in Algorithm 5 in the Supplementary File.

2) Probabilistic Selection: In OSAPSO, the process of objective evaluation occasionally takes place based on the SA-based OES. The most striking feature in OES is that SA can, with a certain probability, accept poor solutions, provided that they learn later from superior ones in the mutation process. From (12), these poor solutions are highly acceptable at the early stage of the search, but their acceptance probability eventually diminishes by the end of the search, which could maintain the search direction out of the local optima's regions while making a good exploration–exploitation tradeoff. In ES strategy, OES is to select individuals from among initial and offspring swarms to form the elite swarm. Similarly, OES is used with POBL in the swarm initialization stage. Algorithm 2 in the Supplementary File realizes OES.

C. Swarm Initialization

Assuming an initial swarm of N particles, each particle has a position randomly generated to ensure extensive distribution of the swarm through the whole search space. To dynamic exploration, the particles' velocity vectors are also stochastically initialized. Furthermore, to further raise the initial swarm's quality, a POBL strategy is embedded into the swarm initialization of PSO. POBL is recommended as it helps PSO: 1) partially expand the space of feasible regions discovered by the initial swarm around $\mathbf{g}_{\text{best}}$ in the opposite direction, as seen in (14); 2) enhance the swarm diversity within these regions; and 3) avoid the high complexity in computational time that we would encounter if POBL is applied to the whole swarm. Thus, applying OBL partially to only a proportion l of the swarm ($\lfloor lN \rfloor$ search particles with the best EOVs ever) from the individuals obtained in the swarm initialization process, the chance that the quality of the initialized swarm approaches the optimal solution can increase [13]. Then, PSO's search performance can be improved with the help of swarm initialization via POBL, as realized in Algorithm 3 in the Supplementary File.

D. Particle Updating

Our proposed particle updating mechanism comprises evolutionary computing with the MWTS-based parent selection, SI and OES, as given in Algorithm 4 in the Supplementary File.

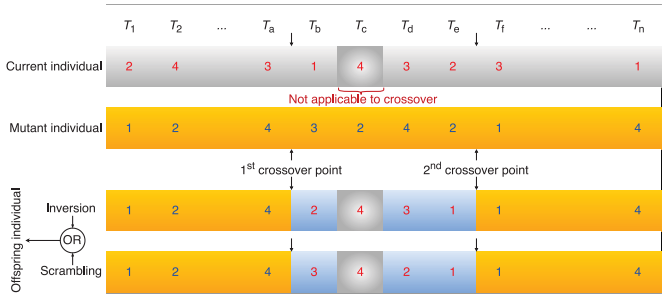


Fig. 4. All-point mutation operator and two-point inversion–scrambling-based crossover operator.

1) Multiway Tournament-Based Parent Selection: First, MWTS [16] is applied as follows. Excluding the underlying particle, a proportion (or rate of ways $q \geq 3/N \in]0, 1[$) are randomly selected from the current swarm. Their top three particles are selected as parents, say $\mathbf{x}_a^t$, $\mathbf{x}_b^t$, and $\mathbf{x}_c^t$, to participate in the mutation process. In such a manner, the swarm evolution is well guided since high-quality individuals with better EOVs have a larger chance of being picked as parents, guaranteeing that beneficial elements of particles are more likely to be retained in the swarm of the next generation.

2) Damping-Based Mutation: With the help of MWTS, a new mutant vector is presented by following one of the common mutation schemes of DE [15] and stemming from the idea of damping in SA (i.e., temperature eventually decreases over iterations in SA, as does the mutation step size) as

$$\ddot{\mathbf{x}}_i^{t+1} = \mathbf{x}_a^t + \sigma\left(\mathbf{x}_b^t - \mathbf{x}_c^t\right), \tag{17}$$

with $(a, b, c) \in \{1, 2, \ldots, \lfloor qN \rfloor\}$, $a \neq b \neq c \neq i$, and σ being a mutation step size is initially set to $\sigma_0 \in]0, 1[$ and linearly decreased (or dampened) by a given mutation damp rate $\lambda \in [0, 1]$, thus enabling the algorithm to closely search feasible regions to eventually possess accelerated decent convergent behavior. The mutation process is utilized to generate a new individual by inheriting good elements from parents (i.e., $\mathbf{x}_a$, $\mathbf{x}_b$, and $\mathbf{x}_c$) picked via MWTS.

3) Inversion–Scrambling-Based Crossover: This operation is adopted by switching alternately, based on a probability strategy, between executing two crossover mechanisms (i.e., inversion and scrambling), thus making the algorithm able to jump out of local optimization. Crossover is employed to create a new individual by blending mutant individual $\ddot{\mathbf{x}}_i^{t+1}$ with current individual $\mathbf{x}_i^t$. Fig. 4 shows this operation with particle encoded as an integer array. The two crossover's boundary elements are selected at random. The current individual shares an inverted or scrambled segment of applicable elements (selected via a crossover rate $C_r \in [0, 1]$) with the mutant individual, while the remaining elements are preserved, so as to produce a new offspring individual. That way, the tasks (elements) chosen in mutation and crossover are assigned to be processed at other nodes. Once the crossover operation completes, a probabilistic selection process is rendered to select survivors to the next elite swarm by comparing the current swarm's individuals to new offspring based on their EOVs.

4) Swarm Intelligence: Swarm information is updated by using the *gbest* PSO [13] model so as to maintain fine balance

between diversity and convergence, as a guide for exploring high-quality solutions. According to the minimization mechanism adopted, the better personal solution $\mathbf{p}_{\text{best}}$ of an individual is selected via the less EOV, whereas the global best one $\mathbf{g}_{\text{best}}$ is determined via the least EOV among all individuals. We recommend this greedy selection with PSO to maintain the swarm diversity and feasibility over iterations, especially when there is sluggishness in substituting for the elements of inferior individuals, which could still contribute to exploring new regions, while there is no frequency of good segments of elements in individuals in one swarm. *gbest* PSO is given in Algorithm 5 in the Supplementary File.

E. Computational Complexity Analysis

The computational complexity of MHAs generally depends on problem definition, population/swarm size (N), the maximum number of iterations ($\hat{T}$), the number of the problem's dimensions ($\hat{D}$), and the cost of fitness/objective evaluation ($\hat{C}$). Based on the detailed analysis shown in Table A in the Supplementary File, we conclude that OSAPSO's complexity can be actually reduced to $\mathcal{O}(\text{OSAPSO}) = \mathcal{O}(1) + \mathcal{O}(N\hat{D} + N\hat{C}) + \mathcal{O}(\hat{T}N\hat{D}) + \mathcal{O}(\hat{T}N\hat{D} + \hat{T}N\hat{C}) + \mathcal{O}(\hat{T}N\hat{D} + \hat{T}N\hat{C}) = \mathcal{O}(N(\hat{D}+\hat{C}) + \hat{T}N(\hat{D}+\hat{C})) = \mathcal{O}(\hat{T}N(\hat{D}+\hat{C}))$, which is equal to the original PSO's.

IV. Performance Evaluation

The simulation setup is presented in this section. Specifically, two typical, contradictory scenarios of TSCC problems are exemplified by using four multiscale cloud datasets of 20 different TSCC instances.

A. Experimental Setup

In our experiments, cloud nodes are constructed with the characteristics listed in Table B in the Supplementary File. Due to resource heterogeneity and diversity in cloud systems, the cloud nodes have diverse configurations and parameters. The node characteristics include: 1) processing capacity measured by average delay to process million instructions (s/MI); 2) data transmission delay (measured by time consumed to transmit one mega byte (s/MB)); and 3) monetary cost and power consumption from computation, storage, and bandwidth usage. Cost and power are, respectively, calculated according to Grid Dollars ($\mathbb{G}$\$) and Grid WattHour ($\mathbb{G}$Wh), that is, currency and power units used in experiments, respectively. To obey how a cloud system practically operates, two state types (i.e., idle and running) are considered for the cloud nodes. If a node does not transfer, store, or process data, it is deemed idle, and vice versa. The parameter values in Table B in the Supplementary File are generated uniformly in ranges following prior work [47], [48].

The cloud system is generally designed to execute all IIoT users' requests. Each request (or job) is broken down into a series of tasks, which are then examined and given an estimate of the resources they require. Each task is supposed to carry a few proprieties, such as the number of instructions, size of the input and output files, amount of memory required, and completion deadline. Depending on the workload of each request, there may be a wide variation in the size of task sets. Thus, in order to duly test the validity and authenticity of OSAPSO on the cloud systems developed, five synthetic groups (sets of tasks) with 200–1000 tasks are created. Tasks are assumed to be independent and nonprimitive, and each task is generated by using a uniform distribution with attributes in Table C in the Supplementary File. With such uniform randomness, various scenarios could be covered in the experiment due to the creation of many task types, some of them requiring a huge amount of bandwidth or memory whereas others requiring more processing capacity, and so on. In addition, we put forward three more realistic case studies, each with five task instances evenly covering from 500 to 2500 tasks. These case studies are altered from three commonly used realistic datasets, including GoCJ [49], as well as cleaned versions of the parallel workload logs of HPC2N and NASA Ames iPCS/860 (available from the "Parallel Workloads Archive" [50]). Attributes of the three realistic datasets are also reported in Table B and C in the Supplementary File. It is noteworthy that the benchmarks (i.e., Tables B in the Supplementary File) are proposed due to the lack of approved benchmarks in this research area.

OSAPSO is compared to some state-of-the-art standard (e.g., PSO and SA), well-known (e.g., CLPSO, CPSO, JADE, and LSHADE), and widely used recent (e.g., WOA and GWO) optimization algorithms. For the sake of fairness, all of the competing methods' parameter settings match those recommended in the respective publications, without any parameter tuning [51]. On the other hand, it is unquestionably true that OSAPSO's best parameters rely on the situation at hand and definitely have an impact on its overall effectiveness. However, many real-world trials are needed to fully understand the impact of parameter adjustment. So, its proper parameter values that could produce the greatest outcomes are determined according to our pilot runs. Standard values are set for some OSAPSO's parameters. Specifically, SA's initial temperature T_0 and final one T_f are fixed to 1000 and 1, respectively, [52]. PSO's inertia weight ω is linearly reduced from 0.9 to 0.4 for balancing global and local searches. Its acceleration constants, c_1 and c_2, are both fixed to 1.2 [53]. The initial mutation step size σ_0, mutation damp rate λ, and crossover rate C_r are set to 0.9, 0.3, and 0.2, respectively. The rate of ways q and the swarm proportion l are set to 0.5 (i.e., roughly one-half of the swarm size) and 0.3 (i.e., roughly one-third of the swarm size), respectively. In order to determine the proper values of σ_0, λ, C_r, q, and l, we employ grid search [54] in the experiments. For all algorithms, swarm size N is set to 50, the maximum number of generations (or iterations) $\hat{T}$ is adopted as the termination criterion and is set to 500, and lower ($\check{j}$) and upper ($\hat{j}$) bounds are, respectively, -1 and 1 in each dimension j of the position and velocity vectors. To reduce randomness impact on the final results, 30 trials of optimizations are operated independently on each set of tasks and the mean results and standard deviations are recorded.

In (7), w_1–w_4 indicate the priority of optimization among power, cost, time, and throughput, respectively. Since our primary goal is to diminish the energy consumption and cost–time bond, we simulate a typical scenario, in which power, cost, time, and throughout have disparate levels of

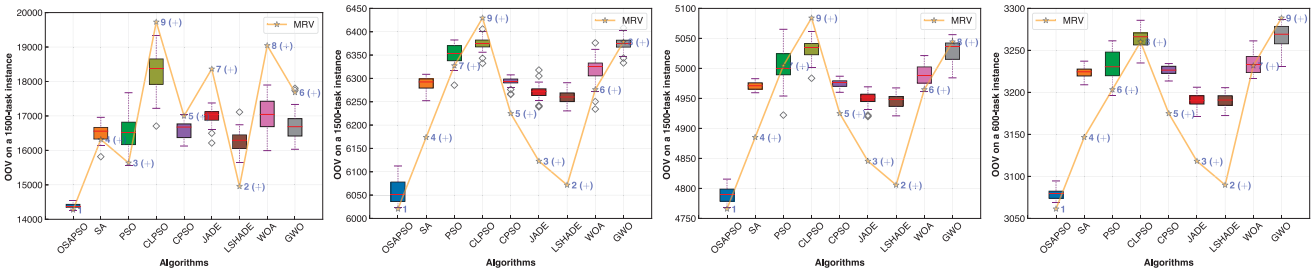


Fig. 5. Stability, ranking, and significance given 50 cloud nodes on the four cloud datasets. (a) GoCJ. (b) HPC2N. (c) NASA Ames iPCS/860. (d) Synthetic.

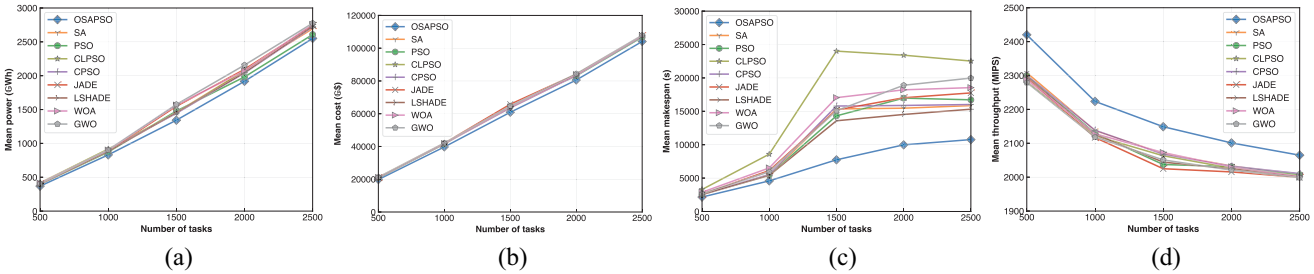


Fig. 6. MOV given 50 cloud nodes on the GoCJ dataset for the four desired objectives. (a) Power consumption. (b) Monetary cost. (c) Service makespan. (d) System throughput.

interest (i.e., $w_1 = 0.5$, $w_2 = 0.2$, $w_3 = 0.2$, and $w_4 = 0.1$).

All experiments are conducted on a server machine equipped with two Intel Xeon Silver-4214 2.20-GHz processors, 64-GB RAM, and 64-bit Windows Server 2022 Operating System. The simulation is conducted in Python version 3.11.0.

B. Performance Metrics

Of the 30 trials' independent experimental results, the mean objective value (MOV) and overall objective value (OOV) are reported to analyze the accuracy and stability of the optimization capabilities of all algorithms. To verify whether the results obtained by OSAPSO have a statistically significant difference from the competing methods,' we conduct a nonparametric test, called Wilcoxon's signed-rank test, at a confidence level 95%, where the symbols "+," "−," and "≈," respectively, indicate that OSAPSO's results are significantly better than, worse than, and similar to those achieved by the corresponding algorithm in comparison. Furthermore, to analyze the algorithms' comprehensive optimization performances, Friedman's test is adopted, in which the mean rank value (MRV) represents each algorithm's optimization capacity rank on average. The best results are shown in *bold*.

C. Results and Analysis

To investigate the effect of OSAPSO and its peers, two computational experiments are designed by using four multiscale datasets with five task sets for each, producing 20 different TSCC instances.

Task Scheduling With a Fixed Number of Nodes and Variable-Size Task Sets: Fig. 5 shows distribution of OOV and MRV of the competing algorithms on the four cloud datasets. In this simulation, the number of cloud nodes is fixed at 50, and the results are recorded over multiple runs of each algorithm. As demonstrated from Fig. 5, OSAPSO surpasses its rivals with significant effectiveness in most cases. In order to further estimate its impact on each single objective, MOV results on the four datasets are individually visualized for the four desired objectives in Fig. 6 and Figs. A–C. As seen, OSAPSO outperforms its peers on all datasets for the four objectives.

Task Scheduling With a Variable Number of Nodes and a Fixed-Size Task Set: Since it is practically difficult to determine the optimal number of cloud nodes. Thus, we further study the influence of the number of cloud nodes on the performance of OSAPSO by using a variable number of nodes from 10 to 100 stepped by 10 for each dataset. Results are shown in Fig. 7. As seen, the number of cloud nodes has high impact on the merit of final results, and MOV improves when a larger number of nodes are deployed. This indicates that the strategies adopted can elastically contribute to the final performance.

Summing up both scenarios, further findings can be concluded.

1) From Fig. 6 and Figs. A–C in the Supplementary File, when there exist numerous tasks submitted, our proposed OSAPSO performs better in terms of scheduling than the compared methods, especially on task sets at scale. For instance, taking the HPC2N dataset as an example, Fig. A in the Supplementary File shows that OSAPSO obviously achieves up to 4%, 21%, and 7% improvements over the second-best performers JADE, LSHADE, and CPSO in power consumption, service makespan, and system throughput objectives, respectively. This signifies our method's effectiveness, which

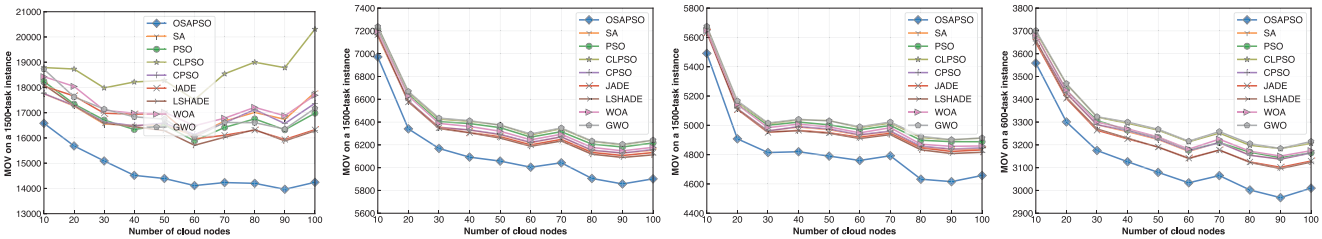


Fig. 7. MOV with a different number of cloud nodes on the four cloud datasets. (a) GoCJ. (b) HPC2N. (c) NASA Ames iPCS/860. (d) Synthetic.

may be primarily attributed to the damped mutation and inversion–scrambling crossover operators and the capability of the feasible solutions' sharing of PSO.

2) When the number of available cloud nodes is small, OSAPSO can, however, manage to achieve superior performance by using the crossover and mutation operators to substitute for inferior nodes, whereas most contemporary counterparts do not perform satisfactorily by preserving the base nodes. To put it another way, these algorithms frequently result in the ongoing requirement for more nodes in order to better the final results. For example, in Fig. 7, when the number of nodes is 10, OSAPSO statically preforms much better on the four datasets with their various TSCC instances. In terms of the other number of nodes, OSAPSO still outperforms its peers. When one hundred cloud nodes are deployed, OSAPSO improves on the four datasets by around 3%–13% over the second-best performer LSHADE. Another conclusion is that the increased number of nodes can reduce MOVs. In a word, it has been evident that OSAPSO is a promisingly scalable algorithm for task scheduling in the cloud thanks to its global exploration and local exploitation characteristics for searching high-quality schedules.

3) OSAPSO clearly outperforms other methods, such as CLPSO, CPSO, JADE, and LSHADE, based only on the mutation mechanism. As in MOV curves, OSAPSO continues to outperform its peers as the number of tasks grows. When a higher number of tasks are submitted, its peers do not pay enough attention to the mutation procedure, resulting in unsatisfactory performances. On the contrary, OSAPSO can handle any given number of tasks and produce desired results by exploiting viable regions that arise as a result of damping-based mutation and the probabilistic selection technique. In terms of stability and ranking, OSAPSO keeps ahead of its peers. For example, as seen in Fig. 5, OSAPSO ranks first on all four datasets with the selected TSCC instances. Additionally, it is evident that OSAPSO and its competitors vary significantly in almost all selected TSCC instances in favor of OSAPSO. This may be due to OSAPSO distinctively seeking for new decent solutions in multiple informative local areas by using POBL, thereby ensuring a more focused search around the better localities thanks to ES and OES, which reveals that OSAPSO has strong exploration and exploitation capabilities. As such, the advantages of OSAPSO have been validated.

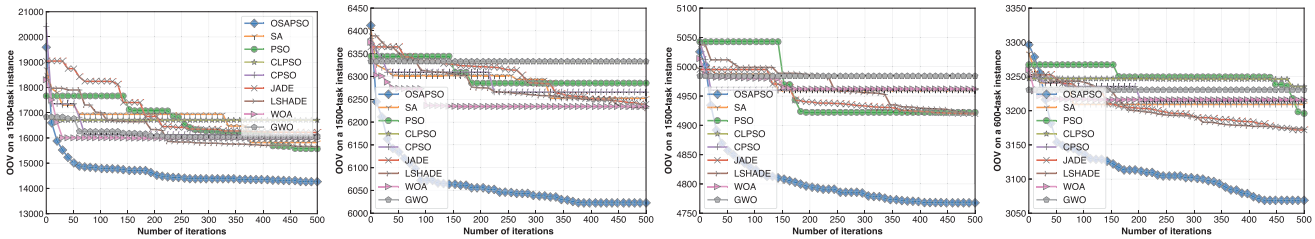
D. Parameter and Convergence Study

In (7), the objective function adopted has four weighting coefficients, and we assume that all its indices are normalized [55]. When weighting coefficients are equal, this states that the priority for optimizing power consumption, monetary cost, service makespan, and system throughput is the same. When $w_1 > w_2 > w_3 > w_4$, our mechanism prioritizes reducing power consumption above monetary cost, service makespan, and system throughput, implying that decision maker(s) (DMs) prefer to spend more money and consume longer time while tolerating a low system throughput, in exchange for environmental and energy conservation. On the contrary, with a throughput's weight coefficient held constant, when $w_1 < w_2 < w_3$, makespan and cost are more prioritized than power, i.e., consumers still need their requests to be executed fastest with a tight budget. Such an approach can assist DMs in giving the multiobjectives a priority order. This results in a broader concept of a utility function, also known as a preference function, which represents the DM's preferences.

Next, we experimentally verify OSAPSO's speedy convergent behavior. Fig. 8 demonstrates that all competing algorithms exhibit convergent behavior on all datasets. However, most of its peers converge more quickly initially, but as iterations progress, the diversity of their solutions decreases. Due to the synergy of damped evolution and probabilistic selection strategies, the inferior individuals in OSAPSO can leverage the discrete information of decision variables in a homogeneous search space and effectively learn from the whole swarm to achieve a potential tradeoff between swarm diversity and convergence rapidity, thus avoiding getting stuck into local optima, hence premature convergence. In summary, OSAPSO effectively converges more quickly than other rivals, and the strategies involved can help find a promising solution space in the early stage of optimization while becoming progressively more helpful in the later iterations so as for OSAPSO to jump out of a local minimum and search better schedules.

E. Analysis of Components of OSAPSO

OSAPSO has five essential hyperparameters that affect the final results noticeably, including the initial mutation step size σ_0, mutation damp rate λ, crossover rate C_r, rate of ways q in MWTS, and swarm proportion l in POBL. In order to



(a) (b) (c) (d)

Fig. 8. Convergence traces given 50 cloud nodes on the four cloud datasets. (a) GoCJ. (b) HPC2N. (c) NASA Ames iPCS/860. (d) Synthetic.

fine-tune them, we adopt a grid searching strategy. As apparently shown in Fig. D in the Supplementary File, a small σ_0 accompanied with a bit high λ gives decent results in the mutation process for all the four datasets. The reason is that such combination may promote the algorithm to exhibit accelerated convergent behavior while maintaining swarm diversity by POBL and OES over first few iterations. Furthermore, C_r and q are correlated as they are both incorporated into the evolution process and properly setting their values is highly expected to elevate OSAPSO's evolutionary capability. From Fig. E in the Supplementary File, C_r and q have strong impact on OSAPSO's performance for the selected TSCC instances on all datasets, and OOV improves when small C_r and high q values are assigned. This implies that inverting/scrambling a lower number of dimensions can boost exploitation, especially when a higher number of individuals are opted to participate in MWTS, thus escaping from local optima by promoting swarm diversity. As aforementioned, when smaller swarm proportions are conducted in POBL, greater enhancements can be delivered while mitigating computational burden. However, the optimal l is difficult to be determined, thus it is set experimental. From Fig. F in the Supplementary File, the final results are not so sensitive to l. Considering the time complexity and model stability, $l = 0.3$ is recommended in OSAPSO. Thus, our OSAPSO can be adapted to generate high-quality schedules in the cloud.

Next, the influence demonstrated by each single major component of OSAPSO is studied. Specifically, its four variants are investigated: OSAPSO\POBL, OSAPSO\ES, OSAPSO\OES, and OSAPSO\PSO. The symbol "\" indicates that a given component is ablated from OSAPSO. MOVs and MRVs of OSAPSO and the four variants are presented in Table D in the Supplementary File for saving space. As demonstrated, whereas OSAPSO and OSAPSO\ES both rank second by delivering best MOVs on only 5 of 25 TSCC instances, OSAPSO\POBL (i.e., ES included), which ranks first, greatly holds comparable advantages on 23 instances when compared to OSAPSO. In other words, OSAPSO behaves similarly to OSAPSO\POBL, OSAPSO\OES, and OSAPSO\PSO on 23, 22, and 23 instances, respectively. For example, whereas OSAPSO\OES has an outstanding performance with the NASA Ames iPCS/860 dataset on only 1500-task and 2000-task instances, it performs the worst with the GoCJ dataset on a 1000-task instance and comparably on the remaining instances, i.e., OES contributes to solving many TSCC instances. OSAPSO\POBL significantly outperforms OSAPSO on a 2500-task instance of the GoCJ dataset and a 1500-task instance of the NASA Ames iPCS/860 dataset; however, its performance is comparable when solving the remaining instances, i.e., POBL is crucial to solving many TSCC instances. Noteworthily, OSAPSO performs significantly better than OSAPSO\ES on all instances, implying that ES alone has considerable impact. Furthermore, Fig. G in the Supplementary File depicts that omitting any of OSAPSO's main components slows down its convergence, particularly in the late stages of optimization when these components, especially ES, can help shield OSAPSO against premature convergence. Thus, OSAPSO behaves quite different from its ablated variants over all instances, which leads to its ultimate benefit in terms of flexibility and trustworthiness for multiscale TSCC problems, provided that all variants statistically perform either worse than or comparable to OSAPSO over the majority of instances. As observed, the results obtained by OSAPSO without its major components can be further optimized. It oppositely confirms that the missing components can promote OSAPSO's performance. In summary, each injected component shows its unique favorable impact to every single TSCC instance and is advantageous to the overall functioning of OSAPSO.

V. Conclusion and Future Work

In this article, a novel task scheduling method is proposed, namely, OSAPSO. It is intended to elevate the optimization capacity of PSO and applied to TSCC problems. Thanks to the proposed POBL, ES, and OES strategies, OSAPSO surpasses the comparative methods, including various PSO-based algorithms, and comes out on top in performance evaluation. It strengthens the search capability of PSO by effectively utilizing the opposite elite space, neighborhood space, as well as swarm information. Furthermore, by optimizing the mapping of IIoT heterogeneous tasks into cloud nodes, an energy-efficient, cost–time-effective, deadline-constrained method is proposed for addressing TSCC problems in a heterogeneous CCP.

OSAPSO may have the potential to tackle more complicated TSCC problems in combination with real-world factors, such as cyberthreat, uncertainty, and cloud–fog architecture. Furthermore, it is worth investigating how to approach discrete and multiobjective high-dimensional optimization problems

by extending OSAPSO with some recently proposed concepts [56], [57]. One of its limitations lies in its many superparameters. Finding some effective hyperparameter tuning methods (e.g., Bayesian optimization) is worth studying.

ACKNOWLEDGMENT

Open Access funding provided by the Qatar National Library.

REFERENCES

[1] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 772–783, Apr. 2018.

[2] M. H. Ghahramani, M. Zhou, and C. T. Hon, "Toward cloud computing QoS architecture: Analysis of cloud systems and cloud services," *IEEE/CAA J. Automatica Sinica*, vol. 4, no. 1, pp. 6–18, Jan. 2017.

[3] E. Cao et al., "Energy and reliability-aware task scheduling for cost optimization of DVFS-enabled cloud workflows," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 2127–2143, Apr.–Jun. 2023.

[4] Q. Wu, M. Zhou, Q. Zhu, Y. Xia, and J. Wen, "MOELS: Multiobjective evolutionary list scheduling for cloud workflows," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 1, pp. 166–176, Jan. 2020.

[5] "U.S. energy information administration." U.S. Natural Gas Exports by Country. Accessed: Feb. 20, 2023. [Online]. Available: https://www.eia.gov

[6] J. Yarow. "Apples 100-acre solar farm is the biggest in the world, and its a thing of wonder to look." 2013. [Online]. Available: https://www.businessinsider.com/photos-apples-massive-solar-array-2013-4

[7] IHS Markit. "Number of connected IoT devices will surge to 125 billion by 2030, IHS markit says." 2017. [Online]. Available: https://news.ihsmarkit.com/prviewer/release_only/slug/number-connected-iot-devices-will-surge-125-billion-2030-ihs-markit-says

[8] X. Deng, D. Wu, J. Shen, and J. He, "Eco-aware online power management and load scheduling for green cloud datacenters," *IEEE Syst. J.*, vol. 10, no. 1, pp. 78–87, Mar. 2016.

[9] H. Yuan, J. Bi, M. Zhou, Q. Liu, and A. C. Ammari, "Biobjective task scheduling for distributed green data centers," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 2, pp. 731–742, Apr. 2021.

[10] A. Bouakkaz, A. J. G. Mena, S. Haddad, and M. L. Ferrari, "Efficient energy scheduling considering cost reduction and energy saving in hybrid energy system with energy storage," *J. Energy Storage*, vol. 33, Jan. 2021, Art. no. 101887.

[11] F. Shabestari, A. M. Rahmani, N. J. Navimipour, and S. Jabbehdari, "A YARN-based energy-aware scheduling method for big data applications under deadline constraints," *J. Grid Comput.*, vol. 20, no. 4, pp. 1–24, 2022.

[12] Z. Ma, G. Wu, P. N. Suganthan, A. Song, and Q. Luo, "Performance assessment and exhaustive listing of 500+ nature-inspired metaheuristic algorithms," *Swarm Evol. Comput.*, vol. 77, Mar. 2023, Art. no. 101248.

[13] E. H. Houssein, A. G. Gad, K. Hussain, and P. N. Suganthan, "Major advances in particle swarm optimization: Theory, analysis, and application," *Swarm Evol. Comput.*, vol. 63, Jun. 2021, Art. no. 100868.

[14] M. A. Strobl and D. Barker, "On simulated annealing phase transitions in phylogeny reconstruction," *Mol. Phylogenet. Evol.*, vol. 101, pp. 46–55, Aug. 2016.

[15] S. Das, S. S. Mullick, and P. N. Suganthan, "Recent advances in differential evolution—An updated survey," *Swarm Evol. Comput.*, vol. 27, pp. 1–30, Apr. 2016.

[16] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021.

[17] F. S. Gharehchopogh and H. Gholizadeh, "A comprehensive survey: Whale optimization algorithm and its applications," *Swarm Evol. Comput.*, vol. 48, pp. 1–24, Aug. 2019.

[18] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, "Grey wolf optimizer: A review of recent variants and applications," *Neural Comput. Appl.*, vol. 30, no. 2, pp. 413–435, Jul. 2018.

[19] N. Tran, T. Nguyen, B. M. Nguyen, and G. Nguyen, "A multivariate fuzzy time series resource forecast model for clouds using LSTM and data correlation analysis," *Procedia Comput. Sci.*, vol. 126, pp. 636–645, Aug. 2018.

[20] R. Majumder and D. Ghose, "A strategic decision support system using multiplayer non-cooperative games for resource allocation after natural disasters," *IEEE Trans. Autom. Sci. Eng.*, early access, Oct. 17, 2022, doi: 10.1109/TASE.2022.3213820.

[21] M. Agarwal and G. M. S. Srivastava, "Opposition-based learning inspired particle swarm optimization (OPSO) scheme for task scheduling problem in cloud computing," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 10, pp. 9855–9875, Oct. 2021.

[22] G. Singh, S. Kumar, and S. Prakash, "A performance improvement model for cloud computing using simulated annealing algorithm," *Int. J. Softw. Innov.*, vol. 10, no. 1, pp. 1–17, 2022.

[23] S. Mangalampalli, G. R. Karri, and M. Kumar, "Multi objective task scheduling algorithm in cloud computing using grey wolf optimization," *Clust. Comput.*, to be published, doi: 10.1007/s10586-022-03786-x.

[24] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, Mar. 2011.

[25] Y. Xue, X. Cai, and W. Jia, "Particle swarm optimization based on filter-based population initialization method for feature selection in classification," *J. Ambient Intell. Humanized Comput.*, vol. 14, no. 6, pp. 7355–7366, Oct. 2022.

[26] Y. Miao and B. Yang, "Multilevel reweighted sparse hyperspectral unmixing using superpixel segmentation and particle swarm optimization," *IEEE Geosci. Remote Sens. Lett.*, vol. 19, pp. 1–5, Sep. 2022.

[27] X. Xu, J. Li, M. Zhou, J. Xu, and J. Cao, "Accelerated two-stage particle swarm optimization for clustering not-well-separated data," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 11, pp. 4212–4223, Nov. 2020.

[28] F. E. Fernandes Jr. and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm Evol. Comput.*, vol. 49, pp. 62–74, Sep. 2019.

[29] D. Yu, Q. Lv, G. Srivastava, C.-H. Chen, and J. C.-W. Lin, "Multiobjective evolutionary model of the construction industry based on network planning," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 2173–2182, Feb. 2023.

[30] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm Evol. Comput.*, vol. 62, Apr. 2021, Art. no. 100841.

[31] Y. Wang and X. Zuo, "An effective cloud workflow scheduling approach combining PSO and idle time slot-aware rules," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 5, pp. 1079–1094, May 2021.

[32] H. Yuan, J. Bi, and M. Zhou, "Temporal task scheduling of multiple delay-constrained applications in green hybrid cloud," *IEEE Trans. Services Comput.*, vol. 14, no. 5, pp. 1558–1570, Sep./Oct. 2021.

[33] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "TTSA: An effective scheduling approach for delay bounded tasks in hybrid clouds," *IEEE Trans. Cybern.*, vol. 47, no. 11, pp. 3658–3668, Nov. 2017.

[34] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[35] Y.-C. Ho and D. L. Pepyne, "Simple explanation of the no-free-lunch theorem and its implications," *J. Optim. Theory Appl.*, vol. 115, no. 3, pp. 549–570, Dec. 2002.

[36] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4712–4721, Oct. 2018.

[37] A. G. Gad, "Particle swarm optimization algorithm and its applications: A systematic review," *Arch. Comput. Methods Eng.*, vol. 29, no. 5, pp. 2531–2561, Aug. 2022.

[38] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3774–3785, Mar. 2021.

[39] J. Bi et al., "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 1172–1184, Apr. 2017.

[40] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 281–295, Jun. 2006.

[41] B. Liu, L. Wang, Y.-H. Jin, F. Tang, and D.-X. Huang, "Improved particle swarm optimization combined with chaos," *Chaos Solitons Fractals*, vol. 25, no. 5, pp. 1261–1271, Sep. 2005.

[42] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, Oct. 2009.

[43] R. Tanabe and A. S. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, 2014, pp. 1658–1665.
[44] M. Tanha, M. H. Shirvani, and A. M. Rahmani, "A hybrid meta-heuristic task scheduling algorithm based on genetic and thermodynamic simulated annealing algorithms in cloud computing environments," *Neural Comput. Appl.*, vol. 33, no. 24, pp. 16951–16984, Dec. 2021.
[45] X. Xiang, X. Yan, C. Gao, S. Zhu, M. Xi, and H. Gao, "A circle chaos random search strategy particle swarm optimization with its application," *Comput. Elect. Eng.*, vol. 102, Sep. 2022, Art. no. 108219.
[46] J. Wang, D. Lin, Y. Zhang, and S. Huang, "An adaptively balanced grey wolf optimization algorithm for feature selection on high-dimensional classification," *Eng. Appl. Artif. Intell.*, vol. 114, Sep. 2022, Art. no. 105088.
[47] B. M. Nguyen, H. T. T. Binh, T. T. Anh, and D. B. Son, "Evolutionary algorithms to optimize task scheduling problem for the IoT based bag-of-tasks application in cloud–fog computing environment," *Appl. Sci.*, vol. 9, no. 9, p. 1730, May 2019.
[48] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of Internet of Things," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 46–59, Jan.–Mar. 2018.
[49] A. Hussain and M. Aleem, "GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, Dec. 2018, Art. no. 38.
[50] D. G. Feitelson. "Parallel Workloads Archive." 2007. [Online]. Available: https://www.cs.huji.ac.il/labs/parallel/workload
[51] I. Fister, J. Brest, A. Iglesias, A. Galvez, and S. Deb, "On selection of a benchmark by determining the algorithms' qualities," *IEEE Access*, vol. 9, pp. 51166–51178, Feb. 2021.
[52] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[53] R. C. Eberhart and Y. Shi, "Particle swarm optimization: Developments, applications and resources," in *Proc. Congr. Evol. Comput.*, vol. 1, 2001, pp. 81–86.
[54] P. Zhang, S. Shu, and M. Zhou, "An online fault detection model and strategies based on SVM-grid in clouds," *IEEE/CAA J. Automatica Sinica*, vol. 5, no. 2, pp. 445–456, Mar. 2018.
[55] H. Mausser, "Normalization and other topics in multi-objective optimization," in *Proc. Fields-MITACS Ind. Problems Workshop*, 2006, pp. 89–101.
[56] Q. Deng, Q. Kang, L. Zhang, M. Zhou, and J. An, "Objective space-based population generation to accelerate evolutionary algorithms for large-scale many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 27, no. 2, pp. 326–340, Apr. 2023.
[57] M. Cui, L. Li, M. Zhou, J. Li, and A. Abusorrah, "A bi-population cooperative optimization algorithm assisted by an autoencoder for medium-scale expensive problems," *IEEE/CAA J. Automatica Sinica*, vol. 9, no. 11, pp. 1952–1966, Nov. 2022.

Ahmed G. Gad (Member, IEEE) received the B.Sc. degree (Hons.) from the Faculty of Computers and Information, Mansoura University, Mansoura, Egypt, in 2013. He is currently pursuing the M.Sc. degree in information technology with the Faculty of Computers and Information, Minia University, Minia, Egypt.

Since 2017, he has been a Teaching Assistant with the Faculty of Computers and Information, Kafrelsheikh University, Kafrelsheikh, Egypt. His interests are computational intelligence, metaheuristic optimization, machine learning, data mining, cloud computing, scheduling, Blockchain, and Internet of Things. For more information, see https://ahmedgad.com.

Essam H. Houssein (Member, IEEE) received the Ph.D. degree in computer science from Minia University, Minia, Egypt, in 2012.

He is currently a Professor of Artificial Intelligence with the Faculty of Computers and Information, Minia University, Minia. He is the Founder and the Chair of Artificial Intelligence Research Group, Minia. He has more than 200 scientific research articles published in prestigious international journals. His research interests include metaheuristic optimization algorithms, WSN, bioinformatics, Internet of Things, artificial intelligence, image processing, and data mining.

MengChu Zhou (Fellow, IEEE) received the B.S. degree in control engineering from Nanjing University of Science and Technology, Nanjing, China, in 1983, the M.S. degree in automatic control from Beijing Institute of Technology, Beijing, China, in 1986, and the Ph.D. degree from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

He joined New Jersey Institute of Technology (NJIT), Newark, NJ, USA, where he is currently a Distinguished Professor. He has more than 1100 publications, including 14 books, more than 750 journal papers (more than 600 in IEEE Transactions), 31 patents, and 32 book-chapters. His interests are Petri nets, automation, robotics, big data, Internet of Things, cloud/edge computing, and AI.

Dr. Zhou is a recipient of Excellence in Research Prize and Medal from NJIT, the Humboldt Research Award for U.S. Senior Scientists from Alexander von Humboldt Foundation, and the Franklin V. Taylor Memorial Award and the Norbert Wiener Award from IEEE Systems, Man, and Cybernetics Society, and the Edison Patent Award from the Research & Development Council of New Jersey. He is a Life Member of Chinese Association for Science and Technology-USA and served as its President in 1999. He is a Fellow of International Federation of Automatic Control, American Association for the Advancement of Science, Chinese Association of Automation, and National Academy of Inventors.

Ponnuthurai Nagaratnam Suganthan (Fellow, IEEE) received the B.A. and M.A. degrees from the University of Cambridge, Cambridge, U.K., in 1990 and 1994, respectively, and the honorary doctorate (Doctor Honoris Causa) degree from the University of Maribor, Maribor, Slovenia, in 2020.

He served as a Research Assistant for the University of Sydney, Camperdown, NSW, Australia, from 1995 to 1996, and a Lecturer with the University of Queensland, Brisbane, QLD, Australia, from 1996 to 1999. He is currently a Research Professor with the KINDI Center for Computing Research, Qatar University, Doha, Qatar.

Yaser M. Wazery received the Ph.D. degree in information technology from Port Said University, Port Said, Egypt, in 2014.

He is currently an Associate Professor with the Information Technology Department, Faculty of Computers and Information, Minia University, Minia, Egypt. His research interests include network security, cryptography, biological inspired clustering, multimedia communication and processing, big data, cloud computing, and data mining.