A Task Scheduling Approach based on Particle Swarm Optimization for the Production of Remote Sensing Products

Yan-e Hou, Wenwen He, Xianyu Zuo, Lanxue Dang and Hongyu Han

Abstract—**The production of remote sensing products is very time-consuming, because it contains a large amount of remote sensing data to be processed. To increase the production efficiency of remote sensing products, a remote sensing processing job is usually divided into multiple tasks that are executed in parallel in the cluster system. So efficient task scheduling is the core problem of remote sensing products production system. This paper issued this problem and proposed a hybrid particle swarm optimization algorithm to improve the production efficiency of the remote sensing products. First, the task scheduling model of remote sensing products was established to minimize the maximum task execution time. Then, in the framework of particle swarm optimization, an inertia weight parameters adjustment strategy based on linear decreasing was adapted. For the solution obtained in each iteration, a local search procedure of task movement between the production nodes was employed to improve the quality of the solution. In addition, a probability acceptance rule was also introduced to allow the acceptance of poor solutions avoid trapping into local optimum prematurely. Finally, we carried out several experiments in the CloudSim simulation platform. The experimental results demonstrate that the proposed algorithm is effective and more competition when compared with the existing scheduling algorithms, such as ant colony optimization, tabu search and other classical scheduling algorithms.**

Index Terms—**Task scheduling, remote sensing products, particle swarm optimization, local search, probability acceptance.**

I. Introduction

WITH the rapid development of information technology and remote sensing technology [1], remote sensing related applications have been widely used in military, environment, agriculture, forestry, geology, ocean and other fields. Users can use a variety of thematic products based on massive remote sensing data in a more convenient and efficient way. In the remote sensing cluster system, the higher requirements for production task scheduling and allocation become very important, due to the large amount of remote sensing data [2,3], the diversification of user needs, increasing user scale and system resource constraints. To speed up the processing of remote sensing data, the production job of remote sensing products is usually divided into several tasks, which can be allocated to different resource nodes in the cluster system. How to schedule these tasks efficiently is the primary goal of remote sensing high-performance cluster system. Meanwhile, it also plays a key role in improving the production efficiency of remote sensing products.

Task scheduling is a kind of complex combinatorial optimization problem that needs to consider optimization objectives and constraints, which is also a NP-hard problem [4]. Scheduling problems have been widely used in many practical applications [5,6], and the typical applications include vehicle scheduling [7], program scheduling [8], and task optimization scheduling [9], etc. In the cluster system [10], optimizing tasks scheduling is one of the key issues to achieve high-performance computing. For massive remote sensing data processing, a reasonable task scheduling optimization algorithm can not only improve the efficiency of task execution, but also improve the performance of the whole cluster system.

Many relevant researches on task scheduling have been carried out. Early task scheduling methods focused on the scheduling of computing resources on single-core processors. These algorithms can solve small-scale task scheduling problems, but it is easy to cause many resource nodes to be idle for a long time, and resulting in low resource utilization rate. The commonly scheduling algorithms include round robin (RR) [11], minimum connection method, first come first serve (FCFS) algorithm [12], and so on. For FCFS, the tasks are executed in an orderly manner on a first-come, first-served basis, and its resource utilization rate is relatively low because of ignoring the difference of different tasks and the processing capability of physical resources. Further, Min-Min algorithm [13] and Max-Min algorithm [14] were proposed to solve the scheduling problem of independent tasks. The former executes the smallest task first, and the latter executes the larger task first and then the smaller task. These two algorithms can easily lead to the degradation of system performance when the task waits for a long time. In recent years, intelligent optimization algorithms [15] have been gradually applied in the task scheduling field due to their good performance. Some metaheuristic algorithms such as genetic algorithm [16], ant colony optimization algorithm [17], tabu search algorithm [18] and particle swarm optimization algorithm [19,20], have been proposed for a

Manuscript received August 07, 2022; revised December 07, 2022. This work has been supported by Science and Technology Development Plan Project of Henan Province (Grant No.212102210393) and Major Science and Technology Project of Henan Province (Grant No.201400210300).

Yan-e Hou is an associate professor of Henan Key Laboratory of Big Data Analysis and Processing, Henan University, Kaifeng, 475004, Henan, China. (e-mail: houyane@henu.edu.cn)

Wenwen He is a graduate student of college of Computer and Information Engineering, Henan University, Kaifeng, 475004, Henan, China. (e-mail:ww82323@henu.edu.cn)

Xianyu zuo is an associate professor of Henan Key Laboratory of Big Data Analysis and Processing, Henan University, Kaifeng, 475004, Henan, China (e-mail:xianyu_zuo@henu.edu.cn)

Lanxue Dang is a professor of Henan Key Laboratory of Big Data Analysis and Processing, Henan University, Kaifeng, 475004, China. (e-mail:danglx@vip.henu.edu.cn)

Hongyu Han is a lecturer of college of Computer and Information Engineering, Henan University, Kaifeng, 475004, China. (corresponding author,e-mail:hanhongyu@henu.edu.cn).

variety of task scheduling problems.

The successfully experience of intelligent optimization algorithms in the field of task scheduling brings new ideas for task scheduling in remote sensing products production system. Ge et al.[21] proposed a tabu-based remote sensing task scheduling algorithm to improve the effectiveness of task scheduling in the remote sensing cluster system. The proposed tabu algorithm comprehensively considered the resource requirements of remote sensing tasks and the production capacity of production nodes to improve the throughput of the cluster system, but it is unsuitable for the small scale cluster system. Li et al.[22] developed a fast simulate annealing algorithm for remote sensing data processing, which improved the efficiency of resource processing but requiring relatively long calculation time. Tang et al.[23] proposed a dynamic resource allocation scheduling strategy based on prior knowledge for dynamic scheduling problems. The strategy can dynamically determine the number of resources to be allocated by the job according to the running time of the unit resource and the current overall system load of the cluster, so as to shorten the total execution time and the average weighted turnover time of the job. Ding et al. [24] proposed a particle swarm optimization algorithm to solve cloud computing task scheduling problem for remote sensing data fusion processing. The best task allocation on each node is sought to increase computing efficiency of proposed algorithm. These researches promote the development of the task scheduling method for remote sensing products production to a certain extent. However, it is necessary to develop more effective task scheduling approach for remote sensing products production owing to the diversity of requirements of practical applications.

This paper aims to provide an effective task scheduling plan for the remote sensing products production system. Considering the characteristics of remote sensing production system, we first build a task scheduling model with the optimization goal of minimizing the maximum total execution time of tasks on resource nodes. And then, an improved hybrid particle swarm optimization algorithm (denoted as HPSO) is proposed. To improve the quality of the proposed algorithm, we improve the inertia weights parameter setting [25] and introduce task movement local search procedure. For worse solution, we also adapt a naive acceptance rule to keep the diversification of the algorithm. The proposed algorithm was tested on the simulation cloud platform and the results proved the effectiveness of the proposed algorithm.

The structure of this article is organized as follows. In Section II, the issued problem is described and modeled. Section III introduces the proposed particle swarm optimization algorithm. In Section IV, the validation and analysis of experiments are conducted. Section V gives the conclusion and the research direction in future.

II. Problem Statement and Model

A. Problem Description

In this paper, we address the production system of remote sensing products, which is a distributed processing system. In this distributed system, there are multiple production nodes that can execute tasks in parallel. For each task, it can be assigned to a production node for execution. The production system consists of three components: client, task scheduler and production nodes, as shown in Fig. 1. The client works on the personal computer of the users, which can provide the users with some functions such as creating production orders, monitoring production process, managing production algorithms, and a series of manual interactions. Task scheduler is a service program running on the task scheduler server. Its major work is to complete the allocation, management and monitoring of the tasks. The production node is responsible for executing task, completing production and verifying production. All the production nodes form a cluster system, where each production node can be executed the task at the same time.

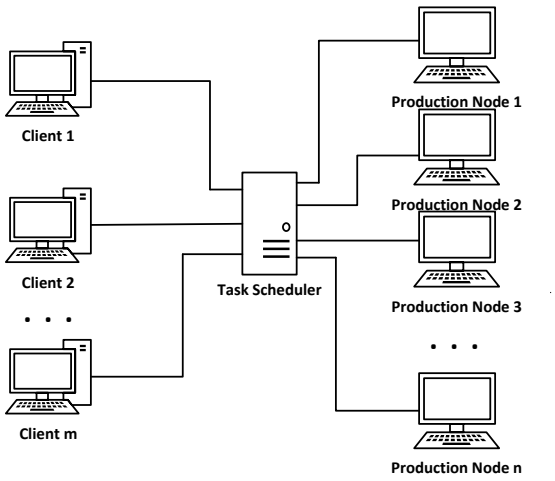


Fig. 1. Structure of Remote Sensing Product Production System

The production process of remote sensing product is described in the following. First, the user creates a remote sensing order containing multiple remote sensing tasks at the client by selecting different algorithmic processes and remote sensing image data. Second, the user submits the created remote sensing order to the task scheduling server via the client, and the task scheduler decomposes it into multiple remote sensing tasks after receiving the remote sensing order. Next, the task scheduler constructs the task pre-execution time (ETC) matrix through the correspondence between the set of remote sensing tasks and the set of production nodes in the cluster. Finally, the task scheduler produces a task scheduling plan by the scheduling algorithm, and then sends each task to the production node for processing.

As mentioned above, generation reasonable and highly effective task scheduling plan in a remote sensing production system is very important because it affects the performance of the whole system. In the view of the scheduling problem, the issued problem in this paper can be expressed as an optimization problem, which several tasks are assigned to some existing resource nodes in the cluster system and a certain optimization objective is achieved simultaneously.

B. Task Scheduling Model

This section, we give the formulation model of the remote sensing production task scheduling. Suppose that there are m tasks to be executed, $Tasks = \{T_1, T_2, ..., T_m\}$, and the list of task numbers is denoted as $T = \{1, 2..., m\}$. There

are n production nodes used to perform tasks, $Nodes = \{N_1, N_2, ..., N_n\}$, and $N = \{1, 2..., n\}$ is the set of node numbers. For each task, it maintains a set of information containing the task ID, the order ID of the task, the task type, and the task status. Each production node has its own IP address and other attributes. The relation between task and production node is that anyone task can be assigned to anyone production node, but one task is just only assigned to one production node. The tasks assigned to different nodes can be executed in parallel. The tasks located on the same production node are non-preemptive, that is, other tasks cannot start executing before one task is completed.

The task ETC matrix can be constructed by the number of tasks and the production nodes. Equation (1) gives the definition of ETC when there are m tasks and n production nodes. For example, etc_{11} represents the pre-execution time when task T_1 is assigned to the production node N_1.

$$ETC(i,j) = \begin{bmatrix} etc_{11} & \cdots & etc_{1n} \\ \vdots & \ddots & \vdots \\ etc_{m1} & \cdots & etc_{mn} \end{bmatrix} \quad (1)$$

The mapping relationship between tasks and nodes can be defined as the task allocation matrix (TA), shown in equation (2).

$$TA(i,j) = \begin{bmatrix} ta_{11} & \cdots & ta_{1n} \\ \vdots & \ddots & \vdots \\ ta_{m1} & \cdots & ta_{mn} \end{bmatrix} \quad (2)$$

Where i ($i \in T$) represents the number of the task, j ($j \in N$) represents the number of the production node, ta_{ij} indicates whether the task T_i is assigned to the production node N_j. When task T_i is assigned to production node N_j, $ta_{ij} = 1$; otherwise $ta_{ij} = 0$.

The execution time of each T_i on production node N_j is recorded as t_{ij}, and we suppose $t_{ij} = etc_{ij}$. The total time-consuming cost TC_j of executing a task on a certain production node j is defined as equation (3). The maximum value of total task time cost in all nodes is recorded as $Makespan$, which is defined in formula (4). The optimization goal of the problem is to find the minimum $Makespan$.

$$TC_j = \sum_{i \in T} ta_{ij} t_{ij}, j \in N \quad (3)$$

$$Makespan = \max\{TC_1, TC_2, ..., TC_n\}, n \in N \quad (4)$$

Therefore, the mathematic model of the task scheduling problem is described as follows. The objective function (5) minimizes the $Makspan$ defined in equation (4). Constraint (6) ensures that a task only is allowed to located one production node at any time. Equation (7) indicates that the decision variable ta_{ij} must be 0 or 1.

Minimize

$$Z = Makespan \quad (5)$$

Subject to:

$$\sum_{i \in T} ta_{ij} = 1, \forall j \in N \quad (6)$$

$$ta_{ij} \in \{0, 1\}, \forall i \in T, j \in N \quad (7)$$

III. Proposed Algorithm

A. Overview of Particle Swarm Optimization

Inspired by the phenomenon of birds foraging and simulating biological population activities, the particle swarm optimization (PSO) algorithm was firstly developed by Eberhart and Kennedy [26]. Due to its advantages of easy implementation, high precision and fast convergence, PSO has been widely used in many optimization problems, such as function optimization, vehicle routing problem, scheduling, and other fields. The candidate solution of optimization problem is defined as a particle in PSO. Each particle corresponds to an adaptation value determined by the optimization function. Each particle is described by position and velocity two attributes, and it can adjust its velocity and position according to its situation and the flight of other particles. The PSO algorithm is usually initialized by generating a random set of particles, and in each iteration, each particle adjusts its velocity and position by tracking the individual extreme value $Pbest$ and the global extreme value $Gbest$, as shown in equation (8) and (9), until a satisfactory solution is found.

$$V_i(t+1) = \omega V_i(t) + c_1 r_1 (Pbest_i(t) - X_i(t)) + c_2 r_2 (Gbest_i(t) - X_i(t)) \quad (8)$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \quad (9)$$

Where t is the number of current iteration and ω is the inertia weight. Assuming that particles search in S-dimensional space, $V_i = \{v_i^1, v_i^2, ..., v_i^S\}$ represents the flying speed of particle i, $Pbest_i = \{p_i^1, p_i^2, ..., p_i^S\}$ is the current individual optimal position of particle i, $Gbest_i = \{g_i^1, g_i^2, ..., g_i^S\}$ represents the global optimal position of all particles, $X_i = \{x_i^1, x_i^2, ..., x_i^S\}$ represents the current position of particle i. c_1 and c_2 are the individual learning factor and the population learning factor, respectively. Factor c_1 guides the particle to fly closer to the individual optimal position, and c_2 regulates the particle to fly closer to the population optimal. These two parameters are usually set to 2. r_1 and r_2 are random numbers between 0 and 1.

B. Encoding and Decoding

For task scheduling problem, the first job is to decide how to represent the result of task scheduling in terms of a particle. Assuming that the number of tasks is M and the number of production nodes is N, a matrix with one row and M columns is constructed. The length of the particle depends on the number of tasks, which is M. The elements at each position in the particle should be integers between 1 to N, which indicates the number of production node assigned to the task. When the particle swarm is created, each particle is randomly generated according by the number of production nodes. For a particle, decoding of the particle is to get the task scheduling plan.

For example, there are 10 tasks to be executed on four production nodes R1, R2, R3 and R4. The list of task numbers is {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, and the list of production nodes numbers is denoted as {1, 2, 3, 4}. If a possible particle X is set to {1, 2, 1, 4, 2, 2, 1, 3, 2, 1}, the task scheduling plan of this particle is described in Fig. 2. There are four tasks assigned to the production node R1,

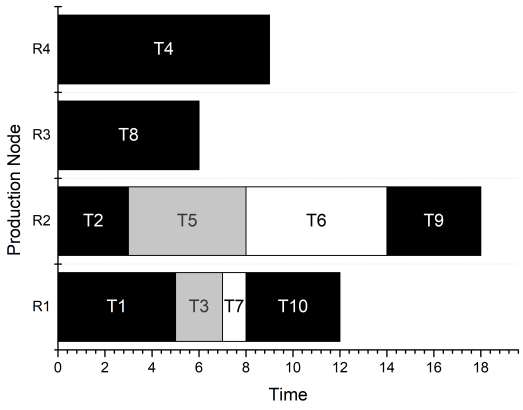


Fig. 2. An Example of a Task Scheduling Plan

which are tasks 1, 3, 7 and 10. In a similar way, tasks 2, 5, 6 and 9 are assigned to node R2, and task 8 and task 4 are allocated to the node R3 and node R4. As a whole, the task lists of the four production nodes is {1, 3, 7, 10},{2, 5, 6, 9},{8}and{4} respectively.

C. Fitness Function

In the PSO algorithm, each particle has a fitness function related to the optimization objective. For this issued problem, the estimated maximum completion time of all tasks can be calculated as shown in equation (4). Since the optimization goal of the problem is to find the minimum execution time of all tasks, the definition of the fitness function of particle X is shown in equation (10).

$$fitness(X) = \frac{1}{Makespan(X)} \tag{10}$$

D. Inertia Weights Adjustment

The change of inertia weight will affect the search ability and convergence speed of the algorithm. On one hand, increasing the inertia weight can jump out of the local optimum of the PSO algorithm, and avoid appearing the premature phenomenon. On the other hand, reducing the inertia weight can improve the search accuracy of the algorithm. Therefore, an effective inertia weights adjustment method is very important for the PSO algorithm.

In this paper, we used a typical linear decreasing strategy to modify the inertia weights, which is defined in equation (11). The value of inertia weights are dynamically updated during the iterative process.

$$\omega == \omega_{\max} - \frac{\omega_{\max} - \omega_{\min}}{t_{\max}} * t \tag{11}$$

Where ω_{max} represents the maximum inertia weight, ω_{min} represents the minimum inertia weight, t_{max} represents the maximum number of iterations, and t represents the current number of iterations.

It can be seen that the speed of the particle is relatively high at the start of the iteration, so a higher value of inertia weight can make the particle swarm algorithm perform more global search. As the number of iterations increases, the speed of the particle decreases, and the value of inertia weight is gradually reduced to make it have better local search ability.

E. Local Search Procedure

In order to improve the search ability of the PSO algorithm, we introduce a local search procedure to do task movement. For the particle, the task movement will apply and try to find better neighborhood solution. The task movement operator is to shift one task from one production node to another production node, which can change the arrangement of the task to find better scheduling plan.

Assume there are two production nodes R1 and R2 and the task lists of them are {1, 2, 3, 4} and {5, 6} respectively. If task 2 is moved from production node R1 to the production node R2 before task 6, the list of tasks of R1 and R2 will change to {1, 3, 4} and {5, 2, 6}. From the operation, we can find that the locations of tasks on different production nodes are changed, and the total execution time of each task list on the production nodes will also be updated.

In local search procedure of the HPSO algorithm, the task movement operator will executed many times until it cannot find better movement. If the algorithm iterates several times with a constant fitness value, we think the algorithm may fall into the local optimum. At this point, we will do the task movement local search procedure.

The process of task movement is described as follows. First, according to the ETC matrix and the task assignment matrix, we calculate the task completion time of each production node, and find the production nodes N_{max} and N_{min} with maximum task completion time and minimum task completion time. Then, we seek the task $Task_i$ in the task list of production node N_{max}, which can have the smallest pre-execution time when it is assigned to the N_{min} production node. Finally, the task $Task_i$ is tried to be shifted to the production node N_{min}. If the task $Task_i$ is assigned to the production node N_{min}, and the task completion time of N_{min} is less than the completion time of N_{max}, the task $Task_i$ is allocated to the production N_{min}. Meanwhile, the task completion time and task assignment matrix of the production nodes are also updated. Otherwise, the movement process is terminated.

To better illustrate the local search procedure, we give an example. There are an ETC matrix and a task assignment matrix for 10 tasks and 4 production nodes, and their definitions are shown in formulation (12). The Fig. 3(a) gives the scheduling plan before the local search.

$$ETC = \begin{bmatrix} 5 & 2 & 3 & 4 \\ 1 & 3 & 2 & 7 \\ 2 & 4 & 1 & 3 \\ 3 & 2 & 5 & 9 \\ 3 & 5 & 1 & 3 \\ 4 & 6 & 5 & 1 \\ 1 & 1 & 3 & 4 \\ 9 & 2 & 6 & 3 \\ 2 & 4 & 4 & 3 \\ 4 & 2 & 2 & 3 \end{bmatrix} \quad TA = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{12}$$

First of all, we can calculate the task completion times of R1, R2, R3 and R4 four production nodes, and their values are 12, 18, 6 and 9 respectively. Then, the production nodes with maximum task completion time and minimum task completion are R2 and R3. The task 5 on node R2 has

the minimum execution time when it is assigned to R3. From the ETC matrix, we can find the execution time of task 5 on R2 and R3 are 5 and 1. If the task 5 is shifted from R2 to R3, the task completion time of R3 is 7, which is smaller than that of R2. Thus, we do the movement of task 5, the new task completion time of all the production nodes are changed into 12, 13, 7 and 9. The maximum task completion time of four production nodes is 13. The task movement procedure will be continuously executed until no task movement operation occurs. After the local search procedure, the task assignment matrix and task assignment matrix are shown in equation (13). The task completion task times of R1, R2, R3 and R4 are 10, 10, 10 and 9. So, the maximum task completion time is 10.

$$ETC = \begin{bmatrix} 5 & 2 & 3 & 4 \\ 1 & 3 & 2 & 7 \\ 2 & 4 & 1 & 3 \\ 3 & 2 & 5 & 9 \\ 3 & 5 & 1 & 3 \\ 4 & 6 & 5 & 1 \\ 1 & 1 & 3 & 4 \\ 9 & 2 & 6 & 3 \\ 2 & 4 & 4 & 3 \\ 4 & 2 & 2 & 3 \end{bmatrix} \quad TA = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{13}$$

The above example depicts the process of task movement and the new task scheduling plan can be seen in Fig. 3(b). Seen from the Fig. 3, the task scheduling plan is more equilibrium after executing task movement local search procedure.

F. Naive Acceptance Rule

In the search process of the algorithm, the algorithm may reach the current local optimum because of lacking of the multifarious solutions. The fitness value may remain unchanged in multiple iterations, and it is necessary to change the update direction of the particles to jump out the local optimum.

For this purpose, we adapt a probability-based acceptance strategy, which is called naive acceptance rule, to accept the generated new particles. If new particle is better than the current particle, the new particle is always accepted. For the worse new particle, the new particle is accepted if $p \geq 0.5$ and p is a random number generated between 0 and 1, otherwise it is rejected.

G. Algorithm Framework of HPSO

The whole algorithm framework of the HPSO algorithm is described in the following.

Step 1: Initialize the population size M, the maximum number of iterations $t_{\max}$, weights $w_{\max}$ and $w_{\min}$, learning factors c_1 and c_2, and the values of other parameters. Set the two parameters α and β to indicate when local search and acceptance rule are used in the algorithm.

Step 2: Generate M initial particles and calculate the fitness value of every particle. The individual and global best optimum particles are denoted as p_{best} and g_{best}. The current number of iterations is $t = 0$.

Step 3: Update the location and speed of each particle by equation (8) and equation (9).

Step 4: Calculate num that is the number of g_{best} with continuously constant fitness value.

Step 5: If $num > \alpha$, the task movement local search procedure is used for each current particle x_i. Then, a new particle $x_i^{'}$ will be obtained.

Step 6: If $num > \beta$, for each new obtained particle $x_i^{'}$, we use naive acceptance rule to decide whether accept $x_i^{'}$ or not. If $x_i^{'}$ is better than x_i, $x_i^{'}$ is accepted; otherwise it will be accepted by 50%.

Step 7: Update the p_{best} and g_{best}.

Step 8: Execute $t = t + 1$. If $t \leq t_{\max}$, go to step 3.

Step 9: Output g_{best}.

IV. Experimental Study

This section, we evaluate the performance of the proposed algorithm by some experiments. First, the HPSO algorithm was compared with other task scheduling algorithms. Then, we tested the advantage of the improvement strategies of the HPSO algorithm. Finally, we analyzed the stability of the HPSO algorithm.

A. Experimental Environment and Parameter Setting

The HPSO algorithm was coded in Java language with JDK 14.0.2. The population size of HPSO is set to 25, and the iteration number is 500. ω_{max} and ω_{min} are set to 0.9 and 0.4 respectively. The learning factors c_1 and c_2 are both set to 2. The parameters α and β are set to 50 and 100 respectively. We also used Cloudsim simulation environment with cloudsim-3.0.3 to evaluate the performance of the proposed algorithm. In Cloudsim, twenty virtual machine nodes were configured with memory of 512MB, processing speed of 1000 MIPS and bandwidth of 1000 MB. All the experiments had been conducted using a computer running windows 10 64-bit operation system with 3.40 GHz CPU and 16 GB of RAM.

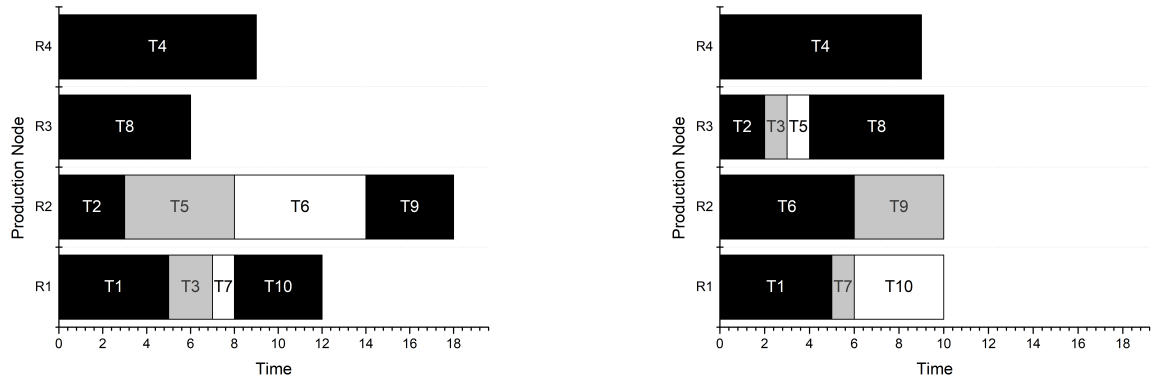
In additional, we used the instance whose ETC matrix is given directly, which can be downloaded from the website (https://github.com/AtheerAlgherairy/Cloudsim-FCFS-SJF-RR-PSO). And the instance was executed 10 times, and the best solution and average solution were calculated.

B. Experiment Results and Comparison

This section, we use the HPSO algorithm solve the instance with different task numbers and compare it with existing task scheduling algorithms. The comparison algorithms include RR, Short Job First (SJF), FCFS, the Max-Min algorithm (Max-Min), standard tabu search (Tabu) and basic ant colony optimization algorithm (ACO). All algorithms solved the same instance and were implemented in the CloudSim simulation platform.

TABLE I displays the results of these algorithms when the number of tasks is 50, 100, 200, 400 and 800 respectively. The column *TaskNum* represents the instance with different task numbers. Columns *Best* and *Avg* indicate the best solution and the average solution of instance running 10 times. The solutions are shown in integer seconds. The best solutions obtained by these algorithms are shown in Fig. 4.

As reported in TABLE I and Fig. 4, the HPSO outperforms the existing scheduling methods. In these methods, the early scheduling methods such as RR, SJF and FCFS have longer



(a) Before Task Movement

(b) After Task Movement

Fig. 3. An Example of Task Movement Local Search Procedure

TABLE I
COMPARISON HPSO WITH OTHER ALGORITHMS

TaskNum	RR		SJF		FCFS		Max-Min		Tabu		ACO		HPSO	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
50	2744	3428	2477	3943	2929	3581	967	967	2427	2796	1906	2062	820	874
100	5106	6315	4754	5529	4589	5716	1980	1980	4263	47746	3374	3571	1535	1657
200	8638	9810	7806	9861	8409	9451	3878	3878	7904	8714	5894	6318	2559	2802
400	14525	17342	13631	160572	14455	17017	8136	8136	14400	15559	11131	116549	5194	5468
800	28851	32894	28224	308514	27867	306424	17048	17048	27868	30626	21143	21493	11012	11490
Average	11973	13958	11372	13248	11650	13281	6401	6402	11372	12494	8690	9020	4224	4458

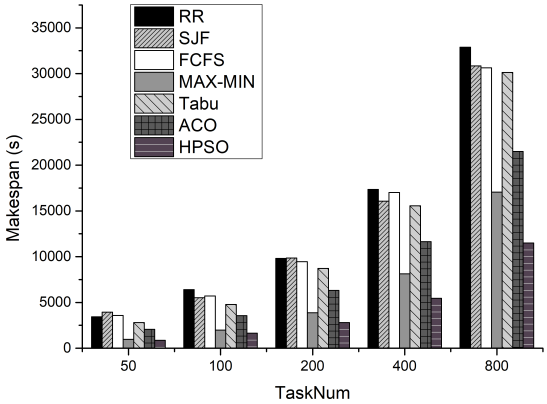


Fig. 4. Best Results Obtained by HPSO and Other Algorithms

execution time. Moreover, the HPSO algorithm is more competitive than Tabu and ACO. The Max-Min algorithm has the best performance among these six algorithms. But HPSO can find better solutions than the Max-Min algorithm. For the Max-Min algorithm and the HPSO algorithm, the average values of the best solutions are 6401 seconds and 4224 seconds. The improvement degree of HPSO is very high. The results show that the proposed algorithm can significantly decrease the execution time of tasks to enhance the production efficiency.

Further, we also calculate the execution time of HPSO and other algorithms and the results are shown in TABLE II. For every algorithm, the execution time is shown in milliseconds.

Seen from the TABLE II, the traditional task scheduling approaches have less execution time at different task scales. Among of three metaheuristics algorithm, tabu algorithm is fastest and ACO is lowest. Our proposed algorithm has a reasonable execution time. For the largest task, HPSO spent only not than 1 seconds. Although HPSO needs more computation time than traditional task scheduling approaches and tabu algorithm, HPSO has the best performance among of all the algorithms.

C. Performance Analysis of Different Improvement Strategies in HPSO

To evaluate the performance of improvement strategies in the HSPO algorithm, we used it to solve the instance with different task numbers. We compared the HPSO with four PSO algorithms, which include standard PSO, linear inertia weight improved PSO (IPSO), IPSO with NA rule algorithm (IPSO-NA) and IPSO having local search named IPSO-LS. All algorithms have the same parameters settings and test environment.

TABLE III shows the results of these five algorithms. The description of the columns in TABLE III is same as that of TABLE I. Fig. 5 gives the best solutions obtained by these algorithms.

From the TABLE III and Fig. 5, we find the HPSO algorithm outperforms other four PSO algorithms. Among other four PSO algorithms, when the local search strategy is adapted, the performance of the algorithm improves significantly. It can be explained that the task movement can find the better task scheduling combination. Additionally, the use of naive acceptance rule can enhance the quality of the algorithm. These results show that these hybrid strategies in the HPSO algorithm can improve its performance.

TABLE II
EXECUTION TIMES OF HPSO AND OTHER ALGORITHMS

TaskNum	RR	SJF	FCFS	Max-Min	Tabu	ACO	HPSO
50	35	36	46	42	39	317	127
100	44	43	47	50	46	1269	178
200	51	55	52	70	57	7970	277
400	59	76	62	150	83	37764	435
800	84	102	80	638	152	268987	805
Average	54.6	62.4	57.4	190	75.4	63261.4	364.4

TABLE III
COMPARISON HPSO WITH OTHER PSO ALGORITHMS

TaskNum	PSO		IPSO		IPSO-NA		IPSO-LS		HPSO	
	Best	Avg	Best	Avg	Best	Avg	Best	Avg	Best	Avg
50	2130	2366	1733	2076	1781	1982	856	978	820	874
100	3706	4388	3670	4102	3464	3743	1503	1662	1535	1657
200	8297	8972	7857	8693	7165	7685	2682	2778	2559	2802
400	15204	17227	15685	16527	13597	15117	5244	5540	5194	5468
800	26505	29470	27055	28097	26787	27346	11052	11518	11012	11490
Average	11168	12485	11200	11899	10559	11175	4267	4495	4224	4458

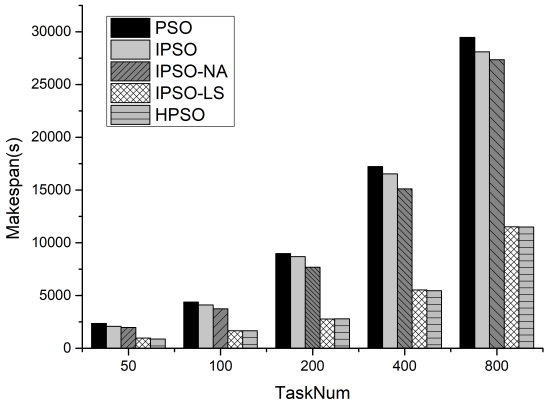


Fig. 5. Simulation Best Results of the Five PSO Algorithms

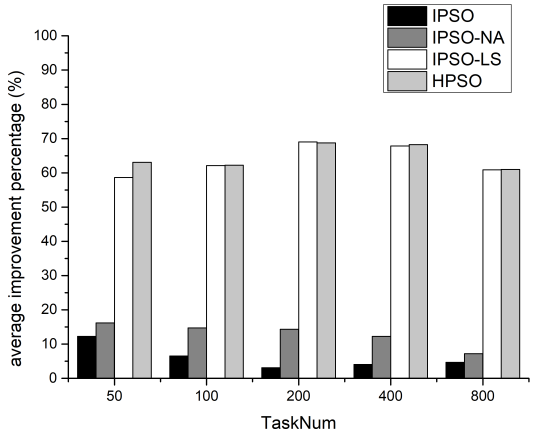


Fig. 6. Average Improvement Percentage of Four PSO Algorithms

Further, we regarded the PSO as the baseline method and calculated the average improvement percentage of the improved PSO algorithms. The results are shown in Fig. 6.

As shown in Fig. 6, the HPSO is still more competitive than other PSO algorithms. When the number of task increases from 50 to 200, the improvement percentage of IPSO-LS and HPSO algorithms increase obviously. With the increasing of the task numbers, the improvement percentages of them decrease a little. For the HPSO algorithm, it has the best whole average improvement. The results reveal that the hybrid strategies used in the HPSO algorithm is effective.

D. Stability Analysis of HPSO

We further analyze the stability of the HPSO algorithm in this section. For each task with different task numbers, we calculated the standard deviation(STD) and coefficient of variation(CV) of the HPSO algorithm and other comparison algorithms. TABLE IV gives the results of them. Due to the same result in each iteration of the Max-Min algorithm, its standard deviation and coefficient of variation do not shown in TABLE IV.

As shown in TABLE IV, we can find that the HPSO algorithm has lowest average standard deviation. Among of these algorithms, the traditional algorithms has bigger standard deviations and coefficient of variation values. For ACO and HPSO algorithm, they both have the smaller deviation values, which means that these algorithms have the stability. Although the ACO algorithm has the smallest coefficient of variation value, the HPSO algorithm is better than it from the view of the optimization objective.

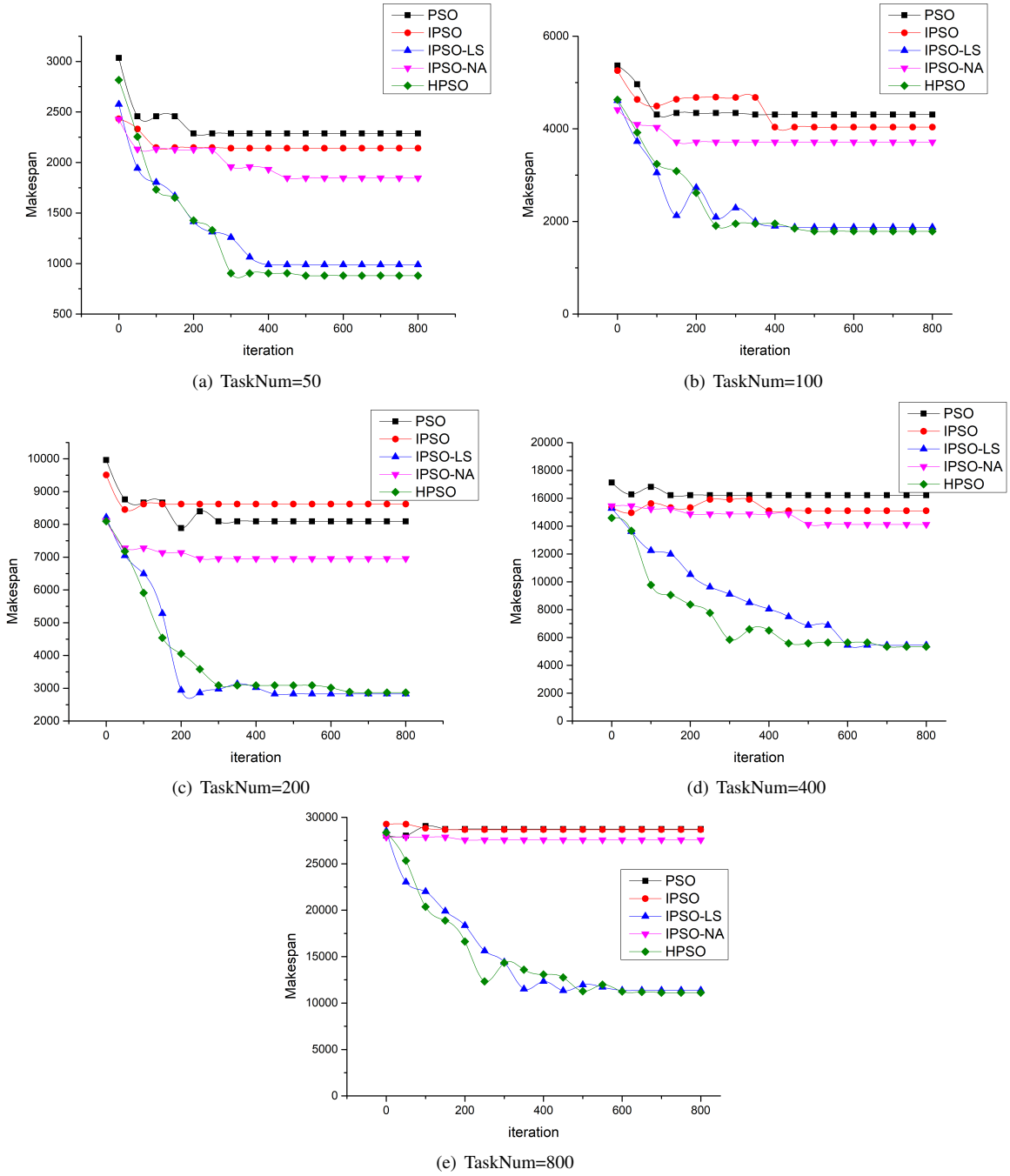
E. Convergence Analysis of HPSO

In this section, we analyse the convergence of the HPSO algorithm when the test instances have different task numbers. We calculated the objective values of the HPSO and other PSO algorithms mentioned in Section C at each iteration. The convergence results of them are shown in Fig. 7.

It can be seen from Fig. 7 that HPSO overcomes the shortcoming of trapping into local optima too early. HPSO has the best performance among all the PSO algorithms. There are three PSO algorithms including PSO, IPSO and IPSO-NA having worse solving quality. They converge too

TABLE IV
STANDARD DEVIATION AND COEFFICIENT OF VARIATION OF THE HPSO AND OTHER ALGORITHMS

TaskNum	RR		SJF		FCFS		Tabu		ACO		HPSO	
	STD	CV(%)	STD	CV(%)	STD	CV(%)	STD	CV(%)	STD	CV(%)	STD	CV(%)
50	550.02	16.05	901.73	22.87	508.08	14.19	177.62	6.35	108.57	5.26	57.70	6.60
100	1052.99	16.67	662.59	11.98	792.38	13.86	205.08	4.30	157.71	4.42	104.57	6.31
200	927.23	9.45	1354.23	13.73	1001.75	10.60	468.01	5.37	231.84	3.67	98.81	3.53
400	2168.31	12.50	1041.24	6.48	1682.37	9.89	686.37	4.41	247.33	2.12	214.63	3.93
800	2928.93	8.90	1347.36	4.37	1596.02	5.21	2733.74	8.93	208.19	0.97	409.09	3.56
Average	1525.50	12.72	1061.43	11.89	1116.12	10.75	854.17	5.87	190.73	3.29	176.96	4.79



(a) TaskNum=50

(b) TaskNum=100

(c) TaskNum=200

(d) TaskNum=400

(e) TaskNum=800

Fig. 7. Change trends of the Objective Values of Different PSO Algorithms

early when iteration number is smaller than 200. For IPSO-LS and HPSO, their performance are better due to their hybrid strategies with local search and acceptance rule. When task number is 50, HPSO can find best solution faster than IPSO-LS. With the task number increased from 100 to 400, HPSO obtaines better solution at almost 300 iterations, and then converges at almost 600 iterations. When the task number reaches 800 (shown in Fig. 7(e)), HPSO finds the

best solution at 500 iterations and it still converges about 600 iterations. In general, these figures show that the HPSO algorithm has good convergence.

V. Conclusion

This paper deals with the task scheduling problem in the production of remote sensing products, which is the key problem to improve the production efficiency. We built a task scheduling model and then developed a hybrid particle swarm algorithm. Several improvement strategies, such as the adjustment of inertia weights, the task movement local search procedure and a naive acceptance rule of inferior solution, were adapted to improve the quality of the algorithm.

The experiments were tested in the CloudSim simulation environment. The results reveal that the proposed algorithm is more effective than the basic PSO algorithm. Among these three improvement strategies, the local search procedure can most significantly improve the performance of the proposed algorithm. Additionally, the probability acceptance rule can make the algorithm jump out the local optima to prevent the premature convergence. Furthermore, we also compared our algorithm with other scheduling algorithms including classical task scheduling algorithm, ACO algorithm and tabu algorithm. The results proved that our presented HPSO algorithm is very effective and stable.

In future, we will do more work on the task scheduling algorithm. The initial effort is to extend our algorithm to solve the more complex task scheduling problem with limited resource constraints. Another work is to sequentially improve the quality of the proposed algorithm, and then apply it to the real remote sensing products production system.

References

[1] C. Elachi and J. J. Van Zyl, "Introduction to the physics and techniques of remote sensing," John Wiley & Sons, 2021.

[2] X. Zhu, F. Cai and J. Tian, "Spatiotemporal fusion of multisource remote sensing data: Literature survey, taxonomy, principles, applications, and future directions," *Remote Sensing*, vol. 10, no. 4, 527, 2018.

[3] A. Abdollahi, B. Pradhan and N. Shukla, "Multi-object segmentation in complex urban scenes from high-resolution remote sensing data," *Remote Sensing* vol. 13, no. 18, 3701, 2021.

[4] B. Mor, D. Shabtay and L. Yedidsion, "Heuristic algorithms for solving a set of NP-hard single-machine scheduling problems with resource-dependent processing times," *Computers & Industrial Engineering*, vol. 153, 107024, 2021.

[5] I. M. Ibrahim, "Task scheduling algorithms in cloud computing: A review," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 4, pp. 1041-1053, 2021.

[6] C. J. Duerden, L. K. Shark, G. Hall and J. M. Howe, "Minimisation of energy consumption variance for multi-process manufacturing lines through genetic algorithm manipulation of production schedule," *Engineering Letters*, vol. 23, no. 1, pp. 40-48, 2015.

[7] E. Yao, T. Liu and T, Lu, "Optimization of electric vehicle scheduling with multiple vehicle types in public transport," *Sustainable Cities and Society*, vol. 52, 101862, 2020.

[8] B. A. Mahafzah, R. Jabri and O. Murad, "Multithreaded scheduling for program segments based on chemical reaction optimizer," *Soft Computing*, vol. 25, pp. 2741-2766, 2021.

[9] W. Shu, K. Cai and N. N. Xiong, "Research on strong agile response task scheduling optimization enhancement with optimal resource usage in green cloud computing," *Future Generation Computer Systems*, vol. 124, pp. 12-20, 2021.

[10] Z. Wu, J. Sun, Y. Zhang, Z. Wei and J. Chanussot, "Recent developments in parallel and distributed computing for remotely sensed big data processing," *Proceedings of the IEEE*, vol. 109, no. 8, pp. 1282-1305, 2021.

[11] F. Alhaidari and T. Z. Balharith, "Enhanced round-robin algorithm in the cloud computing environment for optimal task scheduling," *Computers*, vol. 10, no. 5, 63, 2021.

[12] L. Sahkhar and B. K. Balabantaray, "Scheduling cloudlets to improve response time using cloudsim simulator," in *Proceedings of the International Conference on Computing and Communication Systems*, pp. 483-493, 2021.

[13] A. Hussain, M. Aleem, M. A. Iqbal and M. A. Islam, "Investigation of cloud scheduling algorithms for resource utilization using cloudsim," *Computing and Informatics*, vol. 38, no. 3, pp. 525-554, 2019.

[14] I. Syed, "A hybrid algorithm of min-min and max-min task scheduling algorithms in cloud computing," *International Journal of Recent Technology and Engineering*, vol. 9, pp. 209-218, 2020.

[15] J. S. Wang and J. D. Song, "A Hybrid Algorithm Based on Gravitational Search and Particle Swarm Optimization Algorithm to Solve Function Optimization Problems," *Engineering Letters*, vol. 25, no. 1, pp. 22-29, 2017.

[16] S. Kouidri and C. Kouidri, "Data-intensive scientific workflow scheduling based on genetic algorithm in cloud computing," in *International Conference on Artificial Intelligence and its Applications*, pp. 524-533, 2022.

[17] D. P. Mahato, R. S. Singh, A. K. Tripathi and A. K. Maurya, "On scheduling transactions in a grid processing system considering load through ant colony optimization," *Applied Soft Computing*, vol. 61, pp. 875-891, 2017.

[18] M. I. Alam, M. Pandey and S. S. Rautaray, "Study on proposed resource allocation management for cloud computing using tabu search algorithm," *Current Topics on Mathematics and Computer Science*, vol. 6, pp. 96-103, 2021.

[19] N. Mansouri, B. M. H. Zade and M. M. Javidi, "Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory," *Computers & Industrial Engineering*, vol. 130, pp. 597-633, 2019.

[20] B. A. Mahafzah, R. Jabri and O. Murad, "Multithreaded scheduling for program segments based on chemical reaction optimizer," *Soft Computing*, vol. 25, no. 4, pp. 2741-2766, 2021.

[21] Q. Ge, Y. Hu and L. Zhang, "Task scheduling algorithm for remote sensing based on task demand and service capability," *Remote Sensing Information*, vol. 32, no. 4, pp. 30-34, 2017.

[22] W. Li, Y. Chen, J. Li and F. Yao, "Approach to remotely sensed data processing task scheduling problem based on fast simulated annealing," *Systems Engineering & Electronics*, vol. 33, no. 2, pp. 334-338, 2011.

[23] C. Tang, D. Liu and Y. Ma, "A dynamical scheduling strategy of resources allocation based on priory knowledge," *Remote Sensing Information*, vol. 3, pp. 3-1280, 2011.

[24] Z. Ding, "Particle swarm optimization for cloud computing task scheduling for remote sensing data fusion processing," Masters thesis, Nanjing University of Science and Technology, 2018.

[25] M. M. Golchi, S. Saraeian and M. Heydari, "A hybrid of firefly and improved particle swarm optimization algorithms for load balancing in cloud environments: Performance evaluation," *Computer Networks*, vol. 162, 106860, 2019.

[26] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4, pp. 1942-1948, 1995.