

## Article

# A Particle Swarm Optimization Method for AI Stream Scheduling in Edge Environments

Ming Zhang <sup>1,2</sup>, Luanqi Liu <sup>1</sup>, Changzhen Li <sup>1</sup>, Haifeng Wang <sup>1,2,\*</sup> and Ming Li <sup>3</sup>

<sup>1</sup> School of Information Science and Engineering, Linyi University, Linyi 276002, China

<sup>2</sup> Shandong Provincial Network Key Laboratory, Linyi University, Linyi 276002, China

<sup>3</sup> School of Economics and Management, Anhui Polytechnic University, Wuhu 241000, China

\* Correspondence: gadfly7@126.com; Tel.: +86-135-6391-2296

**Abstract:** With the development of IoT and 5G technologies, edge computing has become a key driver for providing compute, network and storage services. The dramatic increase in data size and the complexity of AI computation models have put higher demands on the performance of edge computing. Rational and optimal scheduling of AI data-intensive computation tasks can greatly improve the overall performance of edge computing. To this end, a particle swarm algorithm based on objective ranking is proposed to optimize task execution time and scheduling cost by designing a task scheduling model to achieve task scheduling in an edge computing environment. It is necessary to fully understand the concept of symmetry of resource utilization and task execution cost indicators. The method utilizes nonlinear inertia weights and shrinkage factor update mechanisms to improve the optimization-seeking ability and convergence speed of the particle-to-task scheduling solution space. The task execution time and scheduling cost are greatly reduced. Simulation experiments are conducted using the Cloudsim toolkit to experimentally compare the proposed algorithm TS-MOPSO with three other particle swarm improvement algorithms, and the experimental results show that the task execution time, maximum completion time and total task scheduling cost are reduced by 31.6%, 23.1% and 16.6%, respectively. The method is suitable for handling large and complex AI data-intensive task scheduling optimization efforts.

**Keywords:** internet of things; edge computing; task scheduling; AI data-intensive computing tasks; particle swarm optimization algorithm



**Citation:** Zhang, M.; Liu, L.; Li, C.; Wang, H.; Li, M. A Particle Swarm Optimization Method for AI Stream Scheduling in Edge Environments. *Symmetry* **2022**, *14*, 2565. <https://doi.org/10.3390/sym14122565>

Academic Editor: Theodore E. Simos

Received: 31 October 2022

Accepted: 2 December 2022

Published: 5 December 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Over the past few decades, cloud computing architectures have been a key force in providing computing, networking and storage services to users, and these services have been used in a wide range of domains. The Internet is an emerging technology today and has been extended to the Internet of Things. The IoT is ubiquitous connectivity, interconnecting tangible objects and end devices through the Internet [1]. End devices are able to sense data from the environment and share the acquired information with their neighboring devices. IoT plays an important role in enhancing service performance and reducing computational costs [2]. At the same time, geographically distributed end devices generate large-scale and complex data at the edge of the network [3]. Although cloud service centers are rich in computational resources, end-to-end latency and energy consumption of edge devices will become increasingly high if all data transfers are uploaded to remote clouds [4]. Cloud computing architectures no longer meet the low-latency and high-performance requirements of AI data-intensive and latency-sensitive tasks. A better solution is to develop and design edge computing. This computing architecture has emerged and become fully popular to reduce the real-time nature of data processing in cloud service centers, and to effectively relieve network transmission pressure [5]. Edge computing is an extension of cloud infrastructure. It is the physical transfer of computing resources and data storage

facilities placed at the edge close to end users, thus reducing latency and bandwidth consumption [6]. Edge computing is different from distributed computing architectures, where information exchange between services is centralized in clusters of machines at a single location in a distributed data center architecture. Edge computing involves a large number of edge devices scattered in different geographical locations, and end-users are able to perform computations close to the edge devices to enable a new computing model where computations occur close to the data source side. The dramatic increase in the number and variety of end devices has fundamentally changed the traditional form of distributed computing architecture. Edge computing is widely popular in several AI fields, such as smart health care, smart manufacturing, autonomous driving and augmented/virtual reality [7]. The combination of edge computing and AI applications solves the arithmetic problem of complex computational models and high data intensity. Edge computing also has some problems similar to those of cloud computing, such as resource allocation and task scheduling. The scheduling of different types of computational tasks is more challenging because of the higher heterogeneity of computational resources within the edge cluster [8]. To handle AI data-intensive computational jobs in edge computing, an efficient edge service scheduling scheme needs to be found to determine the final assignment of each task to minimize execution time and scheduling cost.

Distributed scheduling of AI data-intensive jobs in edge computing environments is an NP problem, and the merit of the scheduling algorithm directly affects the overall performance of task execution. The main goal of task scheduling is to fully utilize system resources to achieve higher throughput to solve large-scale and highly complex computational problems [9]. To address the problem of handling multiple objective optimization, many researchers have developed multi-objective optimization (MOO) techniques [10], which are search methods used to find solutions for several conflicting optimization objectives. For example, it is important to improve the computational performance while taking into account the scheduling cost. A particle swarm algorithm based on objective ranking is proposed for the scheduling of AI data-intensive computation tasks in edge computing environments. The method takes task execution time, maximum completion time and scheduling cost as optimization objectives. A nonlinear inertia weighting strategy and a shrinkage factor update mechanism are added to the traditional particle swarm algorithm. This helps to improve the search ability of particles in the solution space and the balance between local search and global search. The main contributions of this paper are as follows: ① Establish a task scheduling optimization model to provide a theoretical basis for the task scheduling strategy. A particle swarm algorithm based on objective ranking is proposed to improve the convergence of the algorithm by using nonlinear inertia weights and introducing shrinkage coefficients. ② To meet the high requirements of computationally intensive AI tasks in terms of response time and computational performance, the algorithm is applied to intelligent job scheduling in edge environments. The algorithm takes task execution time, maximum completion time and scheduling cost as the optimal objectives, which effectively helps to improve the computing performance of AI job systems. The efficiency of its algorithm is verified through experimental comparison with the benchmark algorithm.

This paper is organized as follows: Section 2 introduces the work related to the task scheduling optimization problem; Section 3 details the formal representation and parameter analysis of the task scheduling model; Section 4 describes the particle swarm algorithm based on objective ranking in detail; Section 5 validates the method and proves the efficiency of its algorithm; Section 6 concludes the study.

## 2. Related Works

Currently, the development of the Internet of Things is changing to intelligent demand, and the data at the edge of the network is exponentially growing, on the basis of which a “large group” of network interaction space is formed. Each individual in the large group is an intelligent body with wisdom and experience. There is an interaction mechanism between individuals, which forms a powerful group intelligence to solve complex problems

through interaction. For the task scheduling optimization problem in the edge environment, the quality of scheduling algorithms has an important impact on the optimization results. The existing scheduling algorithms are mainly divided into two types: classical algorithms and intelligent algorithms. The classical algorithms are: first-in-first-out (FIFO), random selection (RS) and polling (RR) algorithms. Although these algorithms are simple to implement and efficient for simple task scheduling, they cannot adapt to the scheduling requirements of large-scale and high-complexity AI data-intensive computing tasks [11]. Numerous researchers have investigated the task scheduling optimization problem by swarm intelligence algorithms and their variants. The swarm intelligence algorithms, represented by the particle swarm algorithm and the ant colony algorithm, have better robustness, flexibility and distributivity, which are the superiority of swarm intelligence algorithms in solving NP complex problems [12]. The swarm intelligence optimization algorithm is a probabilistic search, which does not require information about the gradient of the problem. Compared to traditional optimization algorithms, individuals in group intelligence algorithms are distributed and do not affect the global situation due to the failure of an individual, which has a strong robustness. The individuals of this type of algorithm can only sense local information, and the individuals follow simple rules that are easy to implement. In addition, the system is used for low communication overhead and can be easily expanded. Particle swarm optimization algorithms (PSO) have been widely used to solve complex problems, such as finding optimal routes, scheduling, structural optimization, image analysis and data mining [13]. Therefore, this paper uses PSO as the research basis for this work.

PSO belongs to the group intelligence heuristic optimization algorithm of computational intelligence. The algorithm is inspired by the predatory behavior of bird flocks, which is very easy to simulate and very effective in dealing with optimization problems [14]. Based on the heterogeneity of edge cluster resources, the simplicity and efficiency of PSO can no longer meet the high-performance requirements of edge systems due to the increasing size and complexity of computational tasks and the diversity of optimization objectives. Therefore, many researchers have investigated the task scheduling problem by improving the PSO and combining it with multi-objective optimization techniques. Bi et al. [15] designed a non-locally convergent particle swarm optimization algorithm strategy for the computing intensive task scheduling problem. The problem of high delay in the processing of tasks is solved. Steenkamp et al. [16] studied the scalability of the polyguided particle swarm optimization algorithm on multi-objective optimization problems. This method uses different file balance coefficient updating strategies to help achieve the scalability of the algorithm. Verma et al. [17] proposed a hybrid particle swarm optimization algorithm based on non-dominated ranking to optimize two conflicting objectives of completion time and cost, and the method uses Pareto optimality to find the best solution. Sahar et al. [18] applied an improved multi-objective particle swarm algorithm to solve the workflow scheduling problem and considered four conflicting optimization objectives of reliability, cost, maximum duration and energy consumption, using four improvements to improve the ability of the algorithm to converge to an undominated solution, allowing a balance between exploration and exploitation during the scheduling process. Pawel et al. [19] realized an archive management method for the dynamic multi-objective optimization problem, and used six archive management methods to effectively track the changing Pareto optimal solution. This work is to use the weighted sum method for multiple optimization objectives and combine them into a single objective to find the optimal solution. Kalka et al. [20] designed a method combining chemical reaction optimization and particle swarm optimization to solve the mixed task scheduling optimization problem. Minimize the calculation cost, execution time and energy consumption under the deadline constraint, and improve the performance of PSO and various quality of service parameters. Huang et al. [21] proposed a particle-swarm-based discrete variant optimization algorithm to study the effects of four different update strategies, linear decreasing, Sigmoid function, simulated annealing and logarithmic decreasing, on inertia weights, respectively. The

results show that the PSO scheduler with a logarithmic decreasing strategy has the shortest task execution time. However, the algorithm introduces more parameters, which increases the complexity of the algorithm. Fakhouri et al. [22,23] developed an integer particle swarm optimization algorithm in order to study the resource efficiency and user satisfaction of edge cloud collaborative computing. The algorithm utilizes integer coding of the computational core to rationalize the position of each particle through rounding and modulo operations, while increasing the diversity of particles. The results show that the method has better performance in terms of SLA satisfaction and resource efficiency, and the encoding method and meta-heuristic of the method are integrated to expand the search space of the algorithm and facilitate the particles to find the optimal solution. Vindigni et al. [24] proposed a population descent swarm optimization algorithm for dynamically adjusting the absolute error of the control strategy by finding the minimum of the time integral. The conventional swarm optimization algorithm is not adaptive to the parameter changes of the controller system. In order to make the controller adaptive in speed, the gain scheduling adaptive tuning controller is used. This method increases the chattering bound of the system by 47%.

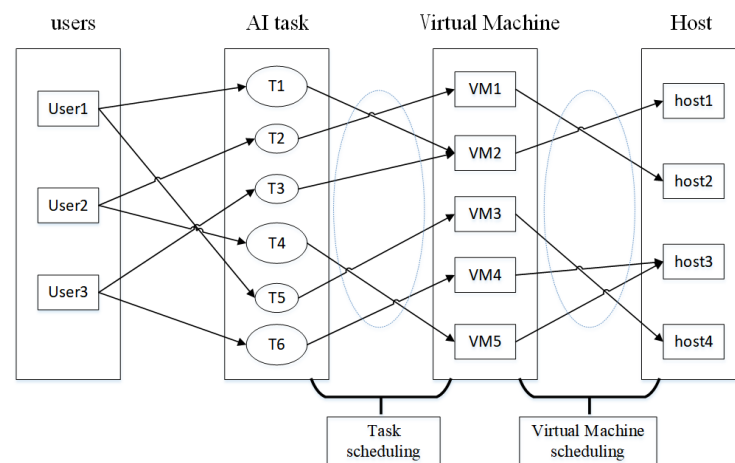
The above research shows that different improvement methods of PSO can effectively improve the performance of task scheduling in the edge environment. When facing AI computing tasks with large data scale and high model complexity, existing methods still need to be improved. In the edge computing environment, the research results for AI data-intensive computing task scheduling problems based on particle swarm optimization algorithms are not common. Therefore, on the basis of the original algorithm, PSO is improved to better realize the reasonable configuration of tasks and to effectively reduce the high latency and scheduling cost of edge AI data-intensive computing tasks in the execution process, and improve the overall efficiency of edge intelligent computing.

### 3. Task Scheduling Model for Edge Computing

Edge computing is a parallel distributed system, and edge service clusters use virtualization techniques to divide computing resources into multiple independent execution environments. These independent environments are called virtual machines (VMs) [25]. These virtual machines provide services such as compute and storage based on user demand. In an edge environment, virtualization technology allows flexible configuration of multiple VMs on the same physical machine. The number of CPU cores per host determines the number of VMs mapped. Task scheduling is actually a combinatorial optimization problem, where tasks are reasonably assigned to different VM nodes within the edge cluster by improving the allocation of VMs through task scheduling algorithms according to the set optimization goals [26]. A reasonable scheduling strategy can reduce task execution time while improving the overall performance of the edge system.

#### 3.1. Problem Description

Edge AI data-intensive computing tasks in IoT can be described as a combination of data-intensive jobs and common sensing tasks [27]. Users submit AI computation task queues to the edge network cluster, and the network edge side of the end device establishes a certain number of virtual machines through virtualization technology to form an edge network cluster. When the edge cluster receives a task request, it assigns the AI data-intensive computation tasks to  $n$  virtual machines for execution through a suitable intelligent scheduling algorithm, and finally, the distributed results are combined and returned to the user. For example, a certain task scheduling scheme  $a$  represents the mapping relationship from computational tasks to virtual machines, and then six computational tasks are assigned to five edge cluster virtual machines for execution when  $a = [2, 1, 2, 5, 3, 4]$ ; the mapping relationship between tasks and virtual machines is shown in Figure 1.



**Figure 1.** Task scheduling framework.

An edge service cluster consists of a service provider that provides resources such as compute and storage to users at different processing power and prices. The set of tasks  $T = \{T_1, T_2, T_3, \dots, T_m\}$  ( $i = 1, 2, 3, \dots, m$ ),  $m$  denotes the total number of AI computation tasks. The attributes of AI computation task can be denoted as  $TD_i = \langle cT_i, sT_i, AL_i, FL_i \rangle$ . AI task attributes are size of task  $i$ , start execution time, execution time and total task completion time. Heterogeneous resources are deployed in the edge cluster, where the physical hosts  $P = \{P_1, P_2, P_3, \dots, P_K\}$  are deployed in a data center and they are independent of each other. A collection of virtual machines  $VM = \{VM_1, VM_2, VM_3, \dots, VM_n\}$  ( $j = 1, 2, 3, \dots, n$ ),  $j$  represents the number of VM compute nodes in the edge cluster. The VM attributes are denoted by a quadruplet as  $VMA_j = \langle VM_{cpu_j}, VM_{mip_j}, VM_{ram_j}, VM_{bw_j} \rangle$ . The attributes of the virtual machine are the number of CPUs, processing performance, memory, and bandwidth. The data transmission volume of task  $i$  assigned to virtual machine  $j$  is expressed in  $VT_{ij}$ . The task assignment matrix  $X_{ij} = A$ . In Equation (1),  $a_{ij}$  denotes the correspondence between task  $i$  and virtual machine node  $j$ . The notation of the problem description is defined as shown in Table 1.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1j} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2j} \\ \vdots & & & & \\ a_{i1} & a_{i2} & a_{i3} & \dots & a_{ij} \end{bmatrix}, \quad (1)$$

**Table 1.** Symbol definition.

Symbol	Description
$m$	number of tasks
$n$	number of Virtual Machines
$k$	number of physical hosts
$T$	task sequence $T = \{T_1, T_2, \dots, T_i\}$
$VM$	virtual machine nodes $VM = \{vm_1, vm_2, \dots, vm_n\}$
$cT_i$	length of task $i$
$VM_{mip_j}$	processing speed of $VM_j$
$VM_{bw_j}$	bandwidth of $VM_j$
$VT_{ij}$	the size of the data transfer assigned to $VM_j$ by task $i$
$X_{ij}$	the assignment matrix, $a_{ij}$ , represents the task $i$ assigned to $VM_j$

### 3.2. Task Scheduling Mathematical Model

The following mathematical model can be made to illustrate the problem of scheduling AI data-intensive computing tasks within an edge cluster:

**Definition 1.** Total Task Execution Time  $T_{exe}$ .

The task execution time is the time consumed after the task is assigned to each different VM node for execution,  $t_{ij}$  denotes the execution time of task  $i$  on VM node  $j$ . Then, each task execution time can be represented by a matrix.

$$Time = \begin{bmatrix} t_{11} & t_{12} & t_{13} & \cdots & t_{1m} \\ t_{n1} & t_{22} & t_{23} & \cdots & t_{2m} \\ \vdots & & & & \\ t_{n1} & t_{n2} & t_{n3} & \cdots & t_{nm} \end{bmatrix}, \quad (2)$$

When a task  $T_i$  is assigned by the system to be processed on  $VM_j$ , the execution time of task  $t_{ij}$  is equal to the actual task size of  $T_i$  divided by the processing speed  $VM_{mipj}$  of that  $VM_j$ , which can be described by the formula:

$$t_{ij} = \frac{cT_i}{VM_{mipj}}, \quad (3)$$

Therefore, the total task execution time is defined as:

$$T_{exe} = \sum_{i=1, j=1}^{m, n} t_{ij}, \quad (4)$$

**Definition 2.** Maximum Completion Time Makespan.

The maximum completion time is the maximum execution time of a task, which reflects the real-time response time and the total completion time of the task. The maximum completion time is equal to the maximum of the sum of the execution time and communication time of each task. The size of the VM bandwidth affects the task communication time, and the communication time of task  $i$  can be expressed as the task data transfer volume  $VT_{ij}$  divided by the bandwidth of  $VM_j$ . Therefore, the communication time of task  $i$  is calculated as follows.

$$T_{comm_{ij}} = \frac{VT_{ij}}{VM_{bwj}}, \quad (5)$$

Therefore, the maximum completion time of the task is defined as:

$$Makespan = \max \{ T_{comm_{ij}} + t_{ij} \mid i \in m, j \in n \}, \quad (6)$$

**Definition 3.** Total Task Scheduling Cost  $T_{cost}$ .

Depending on the resource utilization in the edge service cluster, two different scheduling costs exist: task execution cost and communication cost. The task scheduling cost can be defined as the sum of task execution cost and communication cost. The task scheduling cost of task  $i$  on VM node  $j$  can be expressed in Equation (7).

$$T_{cost_{ij}} = \alpha_1 t_{ij} + \alpha_2 T_{comm_{ij}}, \quad (7)$$

Unit price of processing on different virtual machines in unit time  $\alpha_1$  represents: communication cost of different virtual machines per unit time  $\alpha_2$ . The total task scheduling cost can be expressed as:

$$T_{cost} = \sum_{i=1, j=1}^{m, n} T_{cost_{ij}}, \quad (8)$$

Objective optimization mathematical model:



- (1) Optimal total task execution time.

$$f_1(x) = \text{Min}(T_{exe}), \quad (9)$$

- (2) Maximum completion time optimal.

$$f_2(x) = \text{Min}(\text{Makespan}), \quad (10)$$

- (3) Optimal total task scheduling cost:

$$f_3(x) = \text{Min}(T_{cost}), \quad (11)$$

From the perspective of the entire edge cluster, the lower the total cost of task scheduling and the shorter the execution time, the higher the efficiency of the system; that is, the greater the throughput of the system. To a certain extent, this means that edge AI computing tasks need to consume less energy. Therefore, an effective task scheduling strategy can reduce the task execution time and scheduling cost of the entire system.

#### 4. Task Scheduling Algorithm Based on Goal Ordering

##### 4.1. Standard Particle Swarm Algorithm

PSO is a swarm intelligence algorithm inspired by the flocculation behavior of birds; similar to evolutionary algorithms. PSO is also a population-based heuristic search algorithm, the purpose of which is to seek solutions through mutual cooperation and sharing among individuals in a population. Suppose a population is initialized with  $N$  particles (solutions), the spatial extent of particle search is  $D$ -dimensional, and each particle may be a candidate solution to a particular problem. For a particle  $pi$ , it has two classical parameters, position and velocity. The best position found based on the particle's own motion search experience is called the local optimal solution, denoted by  $Pbest$ . The best position found by the particle in the whole population range through information sharing is called the global optimal solution, denoted by  $Gbest$ . In the iterative process of particle optimization, the velocity and position update of the particles are expressed by Equations (12) and (13); where  $c_1, c_2$  denotes the learning factor, the weight of the statistical acceleration direction that pushes each particle to the global and local optimal position.  $r_1, r_2$  is a random number, uniformly generated in the range  $[0, 1]$ , responsible for providing randomness to the particle flight.  $w$  denotes the inertia weight, which gives the tendency to extend the search space and regulates the search ability of the particles in the solution space.

$$v_i^n(k+1) = wv_i^n(k) + c_1r_1(Pbest_i^n(k) - x_i^n(k)) + c_2r_2(Gbest_i^n(k) - x_i^n(k)), \quad (12)$$

$$x_i^n(k+1) = x_i^n(k) + v_i^n(k+1), \quad (13)$$

##### 4.2. Adaptive Inertia Weights

The particle iteration process includes two processes, global and local search, and keeping a good balance between these two phases can make the algorithm converge to the global optimum in a reasonable time. In order to improve the convergence of PSO and prevent the particles from prematurely falling into the local optimum, this paper introduces an adaptive updating inertia weight strategy in the standard PSO. The parameter  $w$  controls the influence of the previous velocity on the current velocity and regulates the search ability of the particle reconnection space. At the beginning of the iteration, higher values of  $w$  help the global search for the best solution, while lower values help the particle local search for the best. All techniques based on swarm intelligence search rely on exploration and local development to achieve good performance. When the algorithm knows little about the search space, more exploration should be conducted in the initial stage. Since the inertia weight factor  $w$  of the standard particle swarm algorithm is a fixed value, it makes the algorithm easy to fall into local optimum in the early iterative stage. To avoid the above problem, Shi et al. used a linear decreasing function to dynamically improve the

inertia weights  $w$ , and achieved certain results in the convergence and performance of the algorithm. The linear decreasing inertia weight  $w$  is formulated as follows.

$$w = w_{\max} - (w_{\max} - w_{\min}) \times k / K_{\max}, \quad (14)$$

where  $w_{\max}$ ,  $w_{\min}$  are the maximum and minimum inertia weights,  $k$  is the current number of iterations, and  $K_{\max}$  is the maximum number of iterations.

The standard PSO has fewer parameters and is easy to implement. It is suitable for scientific research and engineering applications, and can quickly and efficiently search for solutions, and is well suited to solve task scheduling problems in edge environments. Since optimization problems usually involve more complex search spaces, the standard PSO uses a stochastic search process and results in easy convergence to local optima. In order to obtain more particles in the feasible solution space, it is necessary to improve the computational ability of the algorithm to find the optimal particles in the search space, so as to obtain the optimal scheduling results. The standard PSO is optimized and enhanced in two main ways. First, during the local convergence parameter adjustment, linear decreasing is a typical dynamic update strategy, but in the late iteration, the inertia weight gradually decreases to the minimum value with the number of iterations, which affects the global search accuracy. The nonlinear decreasing form is a dynamic method to prevent the particles from falling into local optimum, which can better improve the particle search ability in the solution space. Second, we improve the particle search accuracy and add some auxiliary operations to balance the local and global search performance of the algorithm by appropriately controlling the two learning factors. The combined two improvements allow the algorithm to find the optimal result faster and more accurately during the iterative process. In this paper, we design a new update method that adaptively updates the inertia weights using a nonlinear decreasing form with the following expression.

$$w = w_{\max} + (w_{\min} - w_{\max}) \times \ln(1 + \frac{\lambda e k}{K_{\max}}), \quad (15)$$

The value of inertial weight is [0.4–0.9],  $K$  is the current number of iterations, the maximum number of iterations. The value of  $e$  is taken to be approximately equal to 2.718 and the coefficient  $\lambda = 3.5$ .

#### 4.3. Contraction Factor Update Mechanism

In Equation (12), the setting of the  $c_1$ ,  $c_2$  parameters affects the trajectory of the particles, while facilitating the information transfer between particles. The velocity vector is effectively deflated by introducing the shrinkage factor  $\varphi$  to establish parameter restrictions on the two learning factors  $c_1$ ,  $c_2$ . It reduces the adverse effects on the algorithm caused by improper settings of learning factors and facilitates the cooperative search of particles in the local area. The expression of the shrinkage factor is given by:

$$\varphi = \frac{2}{|2 + 4c - \sqrt{2c^2 - 4c}|}, \quad (16)$$

Therefore, the optimized new velocity update Equation where  $c = c_1 + c_2$ , and  $c > 2$ .

$$v_i^n(k+1) = wv_i^n(k) + \varphi[c_1r_1(Pbest_i^n(k) - x_i^n(k)) + c_2r_2(Gbest_i^n(k) - x_i^n(k))], \quad (17)$$

On the basis of the new inertia weights, the size of the learning factor is adjusted by introducing the shrinkage factor, so as to control the local and global search ability of the particle in the solution space seeking, and effectively prevent the particle from falling into the local optimal solution state. Integrating the improvement of the particle velocity formula, the method can improve the convergence rate and performance of the algorithm.



#### 4.4. Adaptability Function

The fitness is a weighted composite index used to evaluate the time efficiency and time cost between the optimal implementation of the task optimization scheduling strategy. It represents how well the particles in the particle swarm algorithm model change their position states during the iterative optimization of the algorithm. The lower the fitness of the scheduling scheme, the lower the cost and execution time of the task scheduling, and the better the position state of the particles. In this paper, the adaptation degree calculation is set to include two parts: task execution cost and communication cost. According to the mathematical description of the task scheduling cost in Definition 3, the adaptation degree function is defined as:

$$fitness = Min(\alpha_1 T_{exe} + \alpha_2 T_{comm}), \quad (18)$$

#### 4.5. Particle Swarm Algorithm Based on Target Ranking

Considering the characteristics of AI data-intensive computing tasks in IoT with large scale and complex computational models, there is a need to efficiently implement AI data-intensive computing task scheduling in the edge environment to improve the overall performance of the edge cluster. In the optimization of multiple objectives with conflicting goals, there is no single optimal solution, and the interaction between different objectives finds a set of compromise solutions, called non-dominated or Pareto-optimal solutions. The Pareto-optimal solution set consists of multiple trade-off solutions, and these objective values are used to guide the non-dominated optimal position of the particles. A solution belongs to the Pareto-optimal set if there is no other solution that improves at least one objective without degrading any other objective. There are two basic approaches to handling multi-objective optimization problems [28]: one approach is to reduce the multi-objective optimization problem to a single-objective optimization problem. The optimization is performed mainly by weighting different optimization objectives and combining them into a single objective function. The complexity of this method lies in determining the weight value of each objective according to the importance of the objectives, and different weight values have a significant impact on the results. Therefore, this method is suitable for use when the optimization objectives do not conflict. Another approach is to separately evaluate each optimization objective and combine the concept of Pareto-optimality (Pareto) nature to specify the priorities or weights of the different optimization objectives. The Pareto-optimality approach involves many comparisons to determine the optimal value of the best solution (leader). The determination of the leader is complicated when there are more than two objectives and the objectives conflict with each other. This is because there may be many non-dominated solutions in the vicinity of a particle and it takes more time to find them. Therefore, instead of using the Pareto optimal method, a new method based on goal ranking is used in this paper to overcome these drawbacks by simplifying the search for the best scheduling solution.

Rational task scheduling for AI data-intensive jobs in edge computing environments, where the priority of the task depends on the size of the task itself. In this paper, we design a particle swarm optimization algorithm based on objective ranking to find the optimal solution. By adding a computational fitness function to PSO, the value of each optimization objective is separately calculated, and the solution in each swarm iteration is evaluated by applying a ranking strategy. The solutions are ranked according to the objective values in the PSO, and the particle with the lowest ranking among the corresponding three optimization objectives is used as the optimal solution, i.e., the virtual machine assigned to each task. The specific process is shown in Algorithm 1.

The algorithm consists of the main process of PSO. Based on the PSO algorithm, the process of selecting the best position of the particle by processing the objective function is changed. First, the execution time of each task to reach the available VMs, the maximum completion time and the scheduling cost are calculated according to Equations (9)–(11). The steps in line 3 to line 19 represent the main function of the PSO algorithm based on the goal ordering. The main loop traverses all particles to find the best scheduling solution, and

the PSO algorithm is improved by adding a sorting function to compute the fitness value. Specifically, in line 3, the particle population, velocity, position and task scheduling best solution  $P_{best}$  are initialized; lines 6 to 14 describe the main steps to evaluate the objective function using the goal-ranking approach. In line 6, the sorted list is initialized; lines 7 to 9 calculate the fitness values for each objective, sorting the execution time, scheduling cost and maximum completion time of each task in ascending order, respectively, and storing each result in a separate matrix to select the VM number with the smallest value. Row 11 calculates the sum of the objective function values of the corresponding VMs for each task in the three matrices by cycling through all VMs. Row 13 sorts the sums of the three optimization objectives in ascending order. Line 14 selects the VM with index 0 based on the ascending queue; i.e., the VM with the smallest value. Line 15 compares the fitness value of each particle with the individual particle best using the evaluation function and returns the particle best ( $p_{best}$ ). Line 16 updates the velocity and position of the particles in each iteration according to Equations (13) and (17). For a clearer understanding of the algorithm, assume that there are six tasks to be reasonably assigned to four virtual machines, and calculate the three optimized objective function values according to Equations (9)–(11), respectively. The process of task selection of VM is shown in Figures 2–5.

---

**Algorithm 1:** Particle swarm optimization algorithm based on objective ranking

---

**Input:** number of iterations of the algorithm  $K$ , task  $T$ , virtual machine  $VM$

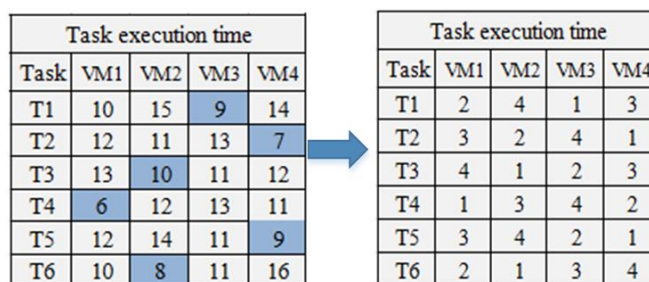
**Output:** the best scheduling solution  $p_{best}$  of tasks to VMs

```

1. while termination criterion not met do
2.   Calculate the three optimization objective values  $\rightarrow T_{exe}, Makespan, T_{cost}$  //Using
   Equations (9)–(11)
3.   Initial Population (Particle Swarm,  $vi, x_i, p_{best}$ )
4.   for Particle Swarm  $\in T$  do //iterate all particles
5.     for  $vm \in VMs$  do
6.       Initial (Objective ranking) //Initialize sort list
7.       Sort the execution time in ascending order  $\rightarrow vm1$ 
8.       Sort the maximum completion time in ascending order  $\rightarrow vm2$ 
9.       Sort the scheduling cost in ascending order  $\rightarrow vm3$ 
10.      for  $vm \in VMs$  do
11.        Objective ranking( $vm$ ) =  $vm1 + vm2 + vm3$  //Sum the ranking of each
        optimization goal
12.      end for
13.      Sum of the three objectives in ascending order  $\rightarrow vm_{best}$ 
14.      Return  $vm_{best}(0)$  //Returns particles with index 0
15.       $p_{best}(i)$  = Evaluate Particle Swarm( $i$ ) //Evaluation of particles
16.      Update ( $vi, x_i$ ) // Update the velocity and position of the particles, suing
        Equations (13) and (17)
17.    end for
18.  end for
19.  return  $p_{best}$ 
20. end while

```

---



Task execution time				
Task	VM1	VM2	VM3	VM4
T1	10	15	9	14
T2	12	11	13	7
T3	13	10	11	12
T4	6	12	13	11
T5	12	14	11	9
T6	10	8	11	16

Task execution time				
Task	VM1	VM2	VM3	VM4
T1	2	4	1	3
T2	3	2	4	1
T3	4	1	2	3
T4	1	3	4	2
T5	3	4	2	1
T6	2	1	3	4

**Figure 2.** Sort value of task execution time.

Maximum completion time				
Task	VM1	VM2	VM3	VM4
T1	10	12	8	14
T2	7	14	11	9
T3	12	9	10	17
T4	9	11	15	12
T5	13	8	10	9
T6	11	6	10	16

Maximum completion time				
Task	VM1	VM2	VM3	VM4
T1	3	2	1	4
T2	1	4	3	2
T3	3	1	2	4
T4	1	2	4	3
T5	4	1	3	2
T6	3	1	2	4

Figure 3. Sort value of maximum completion time.

Task Scheduling Costs				
Task	VM1	VM2	VM3	VM4
T1	11	14	10	9
T2	9	11	7	6
T3	12	9	16	11
T4	8	14	9	13
T5	10	14	12	9
T6	15	8	12	13

Task Scheduling Costs				
Task	VM1	VM2	VM3	VM4
T1	3	4	2	1
T2	3	4	2	1
T3	3	1	4	2
T4	1	4	2	3
T5	2	4	3	1
T6	4	1	2	3

Figure 4. Sort value of task scheduling cost.

Task	VM1	VM2	VM3	VM4	Best VM
T1	2+3+3=8	4+4+2=10	1+2+1=4	3+1+4=8	VM3
T2	3+3+1=7	2+4+4=10	4+2+3=9	1+1+2=4	VM4
T3	4+3+3=10	1+1+1=3	2+4+2=8	3+2+4=9	VM2
T4	1+1+1=3	3+4+2=9	4+2+4=10	2+3+3=8	VM1
T5	3+2+4=9	4+4+1=9	2+3+3=8	1+1+2=4	VM4
T6	2+4+3=9	1+1+1=3	3+2+2=7	4+3+4=11	VM2

Figure 5. Best Virtual Machine Selection.

## 5. Simulation Experiment Results and Analysis

### 5.1. Task Scheduling Environment Configuration

The high cost of designing and testing task scheduling models in a real edge computing environment is why many researchers choose simulation experiments. Using simulation mock-up experiments allows for flexible custom creation of entities and parameter configurations. The research in this paper focuses on the optimization problem of scheduling AI data-intensive computing tasks in an edge environment. The simulation experiments are carried out using CloudSim, a cloud computing simulation experiment platform developed and designed in-house at the University of Melbourne [29]. Given that the CloudSim platform is open source and widely used, and provides a virtual engine, it helps to create and manage multiple independent and collaborative virtual machine services on a single data center node. In this paper, we implement task scheduling algorithms on the CloudSim simulation platform by extending it. CloudSim platform is an open source simulation engine that can simulate the creation of entities in a variety of environments, including cloud data centers, physical hosts and virtual machines, messaging and clock management between components, etc. CloudSim serves as a general and scalable simulation framework that supports the simulation of emerging cloud computing infrastructure and management services. CloudSim allows various services to be modeled, simulated and experimented with, for example, management interface support for virtual machines, memory, storage and bandwidth; this layer implements a generic and scalable simulation framework based on user requirements for host-to-virtual machine mapping, managed application execution

and dynamic monitoring. Usually, the resources of one host in the data can be mapped to multiple virtual machines based on user requirements, therefore, there is competition between virtual machines for host resources. Cloudsim provides the detection of resources, host-to-virtual machine mapping capabilities. Understanding the working mode and workflow of Cloudsim, being familiar with the parameters and roles of each class, and extending the original classes and methods is a way for developers to implement new research methods and algorithms. A few core classes in the source code are outlined below.

- (1) Cloudlet: The task of building the environment; this paper studies AI data-intensive computing tasks.
- (2) DataCenter: Data center that provides virtualized grid resources and contains the allocation policy of virtual machines to resources.
- (3) Host: Extends the machine to virtual machine parameters allocation policy, such as bandwidth, memory, etc. A host can correspond to multiple virtual machines.
- (4) VMScheduler: Virtual machine scheduling policy that manages the execution of tasks.

To evaluate the performance of the algorithm and compute the task scheduling model, the algorithm in this paper is compared with three other improved particle swarm algorithms. The performance of the algorithm is tested by creating available virtual machines and tasks, and mapping the tasks to the virtual machines of the edge cluster. Before performing task scheduling, a task scheduling simulation experiment environment needs to be constructed. The simulation entity is created, which includes setting up the data center, virtual machines and physical hosts, as well as setting up the corresponding parameter information. The basic flow of the simulation is: initialize CloudSim, create a data center and its agents, and create feature objects to store the corresponding properties; create virtual machines and tasks, and set the relevant parameters; after the simulation environment is successfully built, invoke the improved task scheduling algorithm, start CloudSim for simulation, and finally, generate results and perform experimental analysis. Table 2 shows the configuration information about the experiment. There are four types of virtual machines with different CPUs, processing power and memory sizes, which lead to price differences in scheduling costs when tasks are processed. Tasks with different physical attributes are rationally assigned to virtual machines by specific task scheduling algorithms. AI computing tasks in IoT include intensive tasks and common sensor tasks with task lengths initialized to random integer values from 45–15,000.

**Table 2.** Experimental configuration parameters.

Hardware Configuration	configuration information	CPU	RAM	HD	OS
	parameters	Intel(R) Core(TM)i7-3930K	16.0 GB	1 TB	Windows 10
Mainframe Features	configuration information	CPU MIPS	RAM Size	Bandwidth	PE
	parameter	{2000, 2500, 3000, 3500}	{4096, 4096, 4096, 4096}	1,000,000	2
Virtual Machine Configuration	virtual machine number	1	2	3	4
	memory	256	512	256	512
	processing power	150	200	180	300
	CPU cores	1	1	1	1
	bandwidth	1500	2000	1000	3000
Mission Properties	task properties	task size	data transfer volume	number of tasks	
	fetch value	45–14,500	200–800	200–1000	

## 5.2. Results and Analysis

### 5.2.1. Algorithm Performance Analysis

In numerous studies of intelligent particle swarm algorithms applied to task scheduling, it has been demonstrated that the algorithm is suitable for complex scientific research and engineering applications, and can quickly and efficiently find the optimal solution. However, the PSO is a stochastic search process, which can easily converge to local extremes when the particles are searching for the optimal solution. Therefore, different types of methods to improve the PSO are proposed to avoid falling into local optima. To demonstrate the efficiency of the algorithm in this paper, the TS-MOPSO is experimentally compared with three other benchmark algorithms. The various improved PSO are differently implemented, but are based on the same PSO principle, so they work in a relatively close manner. In this case, a large number of AI data-intensive task nodes can distinguish the performance of the improved PSO well.

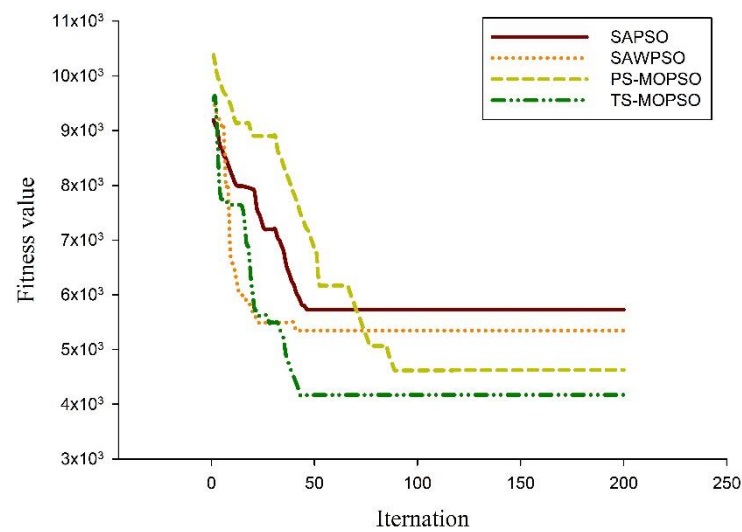
(1) PS-MOPSO [30]: This algorithm is a PSO with Pareto-optimal method by which to obtain the optimal solution set. A nonlinear adaptive inertia weighting strategy is used to update the particle velocity and position to improve the algorithm convergence and finishing computational performance. Finally, the optimal solution is selected from them by comparison.

(2) SAWPSO [31]: This algorithm combines a simulated annealing algorithm and an improved particle swarm algorithm applied to edge computing task scheduling. A new adaptive inertia weight adjustment strategy is used to balance the local and global search ability of particles. At the beginning of the iteration, the simulated annealing algorithm is embedded into the PSO, and the temperature is increased so that the population particles have a higher probability of accepting non-optimal solutions, and thus, jumping out of the local optimum; at the later stage of the iteration, the temperature is reduced and can converge to the global optimum and accurately locate the optimal position of the particles.

(3) SAPSO: This algorithm combines the simulated annealing algorithm and the standard particle swarm algorithm applied to edge computing task scheduling. The difference with the second method lies in the adjustment of inertia weights.

In order to better show the differences of various improved PSO, the experiments are set to 200 iterations and 1000 AI data-intensive task nodes are used as the task model. Figure 6 shows the convergence trends of the search results of the four algorithms, mainly analyzing the algorithm performance in terms of convergence speed and optimal solution. The convergence speed is the minimum number of iterations for the number of particles to reach the optimal position, and the optimal solution is the minimum value for the particles to reach all extreme positions. As shown in Figure 6, the final fitness values of the TS-MOPSO are all significantly smaller than the other three benchmark algorithms, indicating that the algorithm can obtain near-optimal solutions. The slow convergence of the fitness values of PS-MOPSO and SAPSO indicates that the particles of these two algorithms randomly search for solutions during the iterative process. The search result is closer to the optimal solution because the velocity and position update strategy of particles is used in TS-MOPSO, and each particle knows the direction of the optimal solution before searching. SAWPSO performs better than SAPSO because the former algorithm uses the adaptive inertia weighting strategy, which makes the particles jump out of the local extremes with certain probability. Both TS-MOPSO and PS-MOPSO introduce the standard. PS-MOPSO introduces the concept of Pareto-optimality, and it is more complicated to choose the best scheduling solution in the optimal set because there may be many non-dominated solutions in the neighborhood of the particles. Therefore, this algorithm is inferior to TS-MOPSO in both convergence speed and optimal solutions. Based on the above analysis, the TS-MOPSO outperforms other improved PSO in both convergence speed and fitness value, and the experiments prove that the algorithm is more suitable for solving multi-objective task scheduling optimization problems in the edge computing environment.

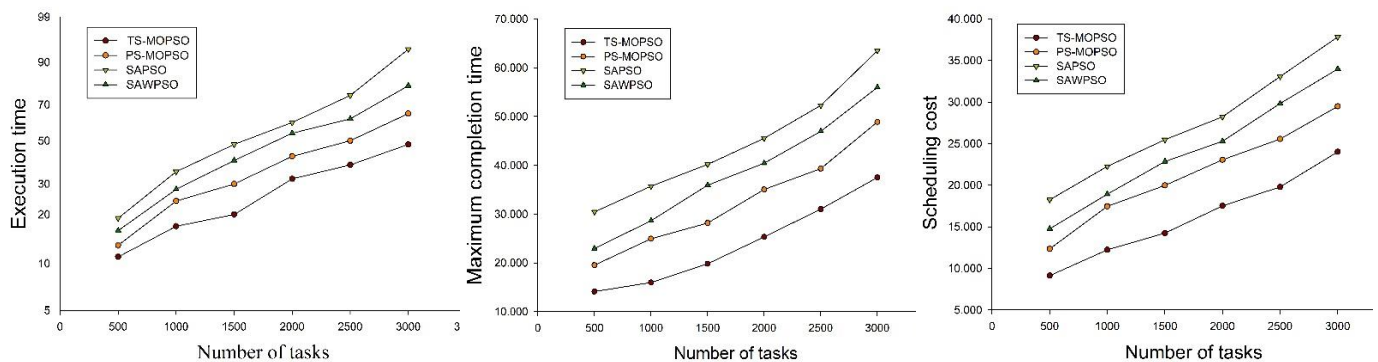




**Figure 6.** Optimization curve of four algorithms for task scheduling.

### 5.2.2. Simulation Experiment Comparison Analysis

Figure 7 shows the experimental comparison trends of the four different algorithms in terms of task execution time, maximum completion time and scheduling cost. It can be seen that the TS-MOPSO achieves the best performance in terms of execution time, maximum completion time and scheduling cost, and its superiority over other algorithms becomes more obvious as the number of tasks increases. In addition, experiments also demonstrate that the combination of multi-objective optimization techniques and particle swarm algorithms is more suitable for solving multi-objective task scheduling optimization problems. SAWPSO uses an adaptive inertia weight update strategy to prevent particles from falling into local optima during iteration, so the overall performance of this algorithm is better than that of SAPSO. Both PS-MOPSO and TS-MOPSO use a nonlinear inertia weight update method and introduce a shrinkage factor update mechanism to balance the performance of the algorithm. Both PS-MOPSO and TS-MOPSO use a nonlinear inertia weight update method, and introduce a shrinkage factor update mechanism to balance the local and global search ability of the algorithm, so that the optimal search ability of the particles in the solution space can be better optimized. The difference is that PS-MOPSO uses the Pareto-optimal method and finds the optimal scheduling solution by multiple comparisons in the Pareto-optimal set, and the number of comparisons increases with the number of optimization objectives, which leads to more time required for processing multiple objectives. In contrast, TS-MOPSO uses an objective-based ranking method to overcome this drawback and improve the overall performance of the algorithm by simplifying the lookup to select the best VM.



**Figure 7.** Optimization target comparison trend chart.

Since heuristic algorithms are stochastic optimization methods, they require at least 10 independent runs to produce meaningful statistical results. In this paper, each method is independently executed more than 20 times and the average results of each algorithm are given. Table 3 gives the best and worst values of the algorithms for each optimization objective during the iterative process, and the corresponding mean and variance are calculated. In terms of accuracy, the TS-MOPSO outperforms the other algorithms in all cases. This can also verify the capability of the combinatorial optimization algorithm compared to the rule-based simple algorithm. In terms of stability, the stability of the compared algorithms is measured by calculating the variance, and the smaller the variance, the more stable the algorithm is. TS-MOPSO has the smallest variance in all three optimization objectives, which indicates that the algorithm is more stable than the other algorithms. Based on the above analysis, TS-MOPSO outperforms the other three benchmark algorithms in terms of task execution time, maximum completion time and scheduling cost, and can well reflect the requirements of real-time, scheduling cost, and low energy consumption for AI data-intensive task scheduling in edge computing environments.

**Table 3.** Comparison of optimization target data.

	Algorithm	Worst Value	Best Value	Mean	Variance
Execution time	SAPSO	94	19	55.2	615
	SAWPSO	80	16	46.6	456.4
	PS-MOPSO	65	13	37.3	295.3
	TS-MOPSO	48	11	27.7	165.3
Maximum completion time	SAPSO	634	304	446	109
	SAWPSO	559	228	384	113
	PS-MOPSO	488	195	326	97.3
	TS-MOPSO	375	141	239	83
Scheduling cost	SAPSO	378	182	55.2	65
	SAWPSO	339	147	46.6	64.2
	PS-MOPSO	294	125	37.3	55
	TS-MOPSO	240	91	27.7	49

Three optimization objectives are selected for the multi-objective optimization problem of AI data-intensive tasks in edge computing environments. Deploying a large number of tasks on virtual machines in edge clusters by suitable task scheduling algorithms helps to save task scheduling costs, as well as execution time, in the edge environment. Therefore, task execution time and scheduling cost are important metrics to reflect the computational performance of edge computing. The following are the effects of the changes in the optimization objectives after different algorithms are applied to task scheduling. As shown in Figure 8, TS-MOPSO obtains the best performance in all three aspects. The TS-MOPSO task scheduling cost is smaller than the other three algorithms, and this algorithm can make good use of the edge cluster computing resources in the edge computing environment, thus significantly reducing the task execution time and scheduling cost.

Through the above sets of experiments, it can be proved that TS-MOPSO outperforms other improved PSO in terms of optimization objectives, fitness and convergence trend. The particle swarm optimization algorithm based on objective ranking proposed in this paper is an efficient algorithm that can provide better approximate optimal solutions for optimal scheduling of AI data-intensive tasks in edge environments.



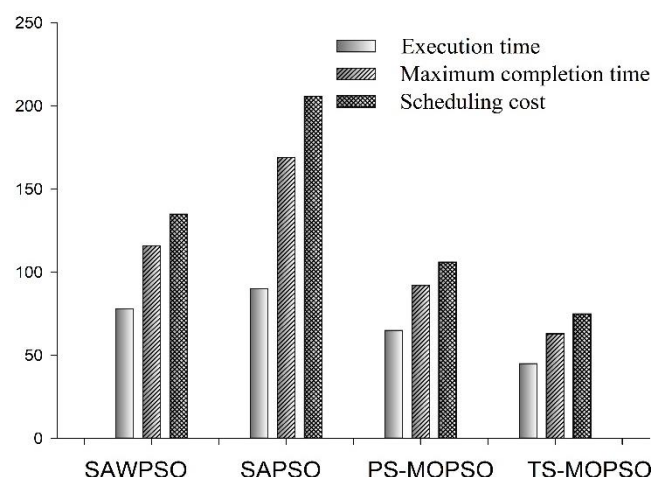


Figure 8. Optimize the effect of target changes.

## 6. Conclusions

Traditional scheduling models and algorithms are affected by the complexity and uncertainty of edge environments, which make it difficult to directly handle AI data-intensive computing tasks. To address the optimization problem of scheduling AI data-intensive computing tasks in IoT, this paper designs and implements a task scheduling optimization model, and considers three optimization objectives: task execution time, maximum completion time and scheduling cost. Furthermore, a particle swarm optimization algorithm based on goal ranking (TS-MOPSO) is proposed to optimize the proposed task scheduling model. TS-MOPSO works on the basis of a multi-objective particle swarm optimization algorithm and adopts nonlinear adaptive inertia weights and a shrinkage factor update mechanism to better balance the local and overall seeking ability of particles. Based on this, three objective functions are evaluated using an objective ranking strategy and the results are used for task scheduling to find the optimal virtual machine for each task. This improves approach speeds and simplifies the process of evaluating the optimization objectives of the algorithm, which improves the overall performance of the edge cluster while reducing the task execution time and scheduling cost by comparing with the benchmark algorithm experiments. Currently, task scheduling research has been limited in targeting data management in computational tasks. This paper does not consider the impact of data layer dependency and data privacy in computational tasks on task scheduling models, but only considers computational tasks and processing data as a whole. This research point will be a future research direction and is addressed in conjunction with the multi-objective optimization problem.

**Author Contributions:** Conceptualization, M.Z. and L.L.; methodology, L.L. and H.W.; formal analysis, H.W. and C.L.; writing—original draft preparation, L.L.; writing—review and editing, M.Z. and L.L.; visualization, M.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by Shan Dong Province Key Research and Development Program of China (No. 2019GGX101003), The funder: Ming Zhang; Shandong Provincial Natural Science Foundation (ZR2021QF078), The funder: Changzhen Li; Shan dong Major Scientific and Technological Innovation Project, China (No. 2019JZZY010134); Postgraduate Education and Teaching Reform Key Training Project (SDYJG19210).

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Abouzaid, L.; Sabir, E.; Elbiaze, H.; Errami, A. The meshing of the sky: Delivering ubiquitous connectivity to ground internet of things. *IEEE Internet Things J.* **2020**, *8*, 3743–3757. [\[CrossRef\]](#)
2. Din, I.U.; Bano, A.; Awan, K.A. LightTrust: Lightweight Trust Management for EdgeDevices in Industrial Internet of Things. *IEEE Internet Things J.* **2021**. [\[CrossRef\]](#)
3. Kim, Y.; Song, C.; Han, H.; Jung, H. Collaborative Task Scheduling for IoT-Assisted Edge Computing. *IEEE Access* **2020**, *8*, 216593–216606. [\[CrossRef\]](#)
4. Zhang, N.; Li, W.; Liu, Z.; Li, Z.; Liu, Y. A New Task Scheduling Scheme Based on Genetic Algorithm for Edge Computing. *CMC-Comput. Mater. Contin.* **2022**, *71*, 843–854.
5. Kang, L.; Chen, R.S.; Cao, W. Mechanism analysis of non-inertial particle swarm optimization for Internet of Things in edge computing. *Eng. Appl. Artif. Intell.* **2020**, *94*, 103803. [\[CrossRef\]](#)
6. Zhou, E.; Zhang, J.; Dai, K. Research on Task and Resource Matching Mechanism in the Edge Computing Network. *Int. Core J. Eng.* **2020**, *6*, 94–104.
7. Kh, A.; Iud, B.; Aa, C. Intelligent and Secure Edge-enabled Computing Model for Sustainable Cities using Green Internet of Things. *Sustain. Cities Soc.* **2021**, *68*, 102779.
8. Abdullahi, M.; Ngadi, M.A.; Dishing, S.I. An efficient symbiotic organisms search algorithm with chaotic optimization strategy for multi-objective task scheduling problems in cloud computing environment. *J. Netw. Comput. Appl.* **2019**, *133*, 60–74. [\[CrossRef\]](#)
9. Li, E.; Zeng, L.; Zhou, Z.; Chen, X. Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 447–457. [\[CrossRef\]](#)
10. Guang, P. Multi-objective Optimization Research and Applied in Cloud Computing. In Proceedings of the 2019 IEEE International Symposium on Software Reliability Engineering Workshops, Berlin, Germany, 27–30 October 2019; pp. 97–99.
11. SI, G. Towards Application-Driven Task Offloading in Edge Computing Based on Deep Reinforcement Learning. *Micromachines* **2021**, *12*, 1011.
12. Shang, J.; Tian, Y.; Liu, Y. Production Scheduling Optimization Method Based on Hybrid Particle Swarm Optimization Algorithm. *J. Intell. Fuzzy Syst.* **2018**, *34*, 955–964. [\[CrossRef\]](#)
13. Fang, J.; Ma, A. IoT Application Modules Placement and Dynamic Task Processing in Edge-Cloud Computing. *IEEE Internet Things J.* **2021**, *8*, 12771–12781. [\[CrossRef\]](#)
14. Milan, S.T.; Rajabion, L.; Darwesh, A.; Hosseinzadeh, M. Priority-based task scheduling method over cloudlet using a swarm intelligence algorithm. *Clust. Comput.* **2020**, *23*, 663–671. [\[CrossRef\]](#)
15. Bi, J.; Yuan, H.; Duanmu, S. Energy-Optimized Partial Computation Offloading in Mobile-Edge Computing with Genetic Simulated-Annealing-Based Particle Swarm Optimization. *IEEE Internet Things J.* **2020**, *8*, 3774–3785. [\[CrossRef\]](#)
16. Steenkamp, C.; Engelbrecht, A.P. A Scalability Study of the Multi-Guide Particle Swarm Optimization Algorithm to Many-objectives. *Swarm Evol. Comput.* **2021**, *66*, 100943. [\[CrossRef\]](#)
17. Verma, A.; Kaushal, R. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. *Parallel Comput.* **2017**, *62*, 1–19. [\[CrossRef\]](#)
18. Saeedi, S.; Khorsand, R. Improved many-objective particle swarm optimization algorithm for scientific workflow scheduling in cloud computing. *Comput. Ind. Eng.* **2020**, *147*, 106649. [\[CrossRef\]](#)
19. Paweł, J.; Beatrice, M.; Ombuki, B.; Andries, P. Multi-guide particle swarm optimisation archive management strategies for dynamic optimisation problems. *Swarm Intell.* **2022**, *16*, 143–168.
20. Kalka, D.; Sharma, S.C. A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing. *Sustain. Comput. Inform. Syst.* **2021**, *32*, 100605.
21. Huang, X.; Li, C.; Chen, H.; An, D. Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. *Clust. Comput.* **2020**, *23*, 1137–1147. [\[CrossRef\]](#)
22. Fakhouri, H.N.; Hudaib, A.; Sleit, A. Multivector particle swarm optimization algorithm. *Soft Comput.* **2019**, *24*, 11695–11713. [\[CrossRef\]](#)
23. Wang, B.; Cheng, J.; Cao, J. Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve SLA satisfaction. *PeerJ Comput. Sci.* **2022**, *8*, e893. [\[CrossRef\]](#)
24. Vindigni, C.R.; Orlando, C.; Milazzo, A. Computational Analysis of the Active Control of Incompressible Airfoil Flutter Vibration Using a Piezoelectric V-Stack Actuator. *Vibration* **2021**, *4*, 369–396. [\[CrossRef\]](#)
25. Bellendorf, J.; Dám, M. Classification of optimization problems in fog computing. *Future Gener. Comput. Syst.* **2020**, *107*, 158–176. [\[CrossRef\]](#)
26. Alsurdeh, R.; Calheiros, R.N.; Matawie, K.M. Hybrid Workflow Scheduling on Edge Cloud Computing Systems. *IEEE Access* **2021**, *9*, 134783–134799. [\[CrossRef\]](#)
27. Pishgoo, B.; Azirani, A.A.; Raahemi, B. A hybrid distributed batch stream processing approach for anomaly detection. *Inf. Sci.* **2021**, *543*, 309–327. [\[CrossRef\]](#)
28. Shi, Z.; Shi, Z.G. Multi-node Task Scheduling Algorithm for Edge Computing Based on Multi-Objective Optimization. *J. Phys. Conf. Ser.* **2020**, *1607*, 012017. [\[CrossRef\]](#)
29. Sahkhar, L.; Balabantaray, B.K. Scheduling Cloudlets to Improve Response Time Using CloudSim Simulator. *Lect. Notes Netw. Syst.* **2021**, *170*, 483–493.

- 
30. Xie, Y.; Zhu, Y.; Wang, Y. A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment. *Future Gener. Comput. Syst.* **2019**, *97*, 361–378. [[CrossRef](#)]
  31. Yang, L.; Hu, X.; Li, K. A vector angles-based many-objective particle swarm optimization algorithm using archive. *Appl. Soft Comput.* **2021**, *106*, 107299. [[CrossRef](#)]