

**RESEARCH ARTICLE**

# PECCO: A Profit and Cost-oriented Computation Offloading Scheme in Edge-Cloud Environment with Improved Moth-flame Optimisation

Jiashu Wu<sup>1,2</sup> | Hao Dai<sup>1,2</sup> | Yang Wang\*<sup>1</sup> | Shigen Shen<sup>3</sup> | Chengzhong Xu<sup>4</sup>

<sup>1</sup>Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China

<sup>2</sup>University of Chinese Academy of Sciences, Beijing 100049, China

<sup>3</sup>Shaoxing University, Shaoxing 312000, China

<sup>4</sup>University of Macau, Macau 999078, China

**Correspondence**

\*Corresponding Author: Yang Wang, Email: yang.wang1@siat.ac.cn

**Present Address**

1068 Xueyuan Avenue, Shenzhen University Town, Shenzhen 518055, Guangdong, P.R.China

**Summary**

With the fast growing quantity of data generated by smart devices and the exponential surge of processing demand in the Internet of Things (IoT) era, the resource-rich cloud centres have been utilised to tackle these challenges. To relieve the burden on cloud centres, edge-cloud computation offloading becomes a promising solution since shortening the proximity between the data source and the computation by offloading computation tasks from the cloud to edge devices can improve performance and Quality of Service (QoS). Several optimisation models of edge-cloud computation offloading have been proposed that take computation costs and heterogeneous communication costs into account. However, several important factors are not jointly considered, such as heterogeneities of tasks, load balancing among nodes and the profit yielded by computation tasks, which lead to the profit and cost-oriented computation offloading optimisation model *PECCO* proposed in this paper. Considering that the model is hard in nature and the optimisation objective is not differentiable, we propose an improved Moth-flame optimiser *PECCO-MFI* which addresses some deficiencies of the original Moth-flame Optimiser and integrate it under the edge-cloud environment. Comprehensive experiments are conducted to verify the superior performance of the proposed method when optimising the proposed task offloading model under the edge-cloud environment.

**KEYWORDS:**

Cloud Computing, Edge-Cloud Computation Offloading, Internet of Things, Moth-flame Optimiser

## 1 | INTRODUCTION

With the rapid prevalence of smart devices<sup>1</sup> such as mobile phone and Internet-of-Things (IoT) devices<sup>2,3,4</sup>, a vast amount of data has been generated<sup>5</sup> and the demand of computation resources has been boosted<sup>6,7</sup>. Due to the limited computation, storage and energy capacity of these smart devices<sup>8</sup>, the powerful *cloud computing* has been leveraged to provide elastic on-demand services to cope with limitations of smart devices<sup>9,10</sup>. With the support of resource-rich cloud servers, processing and storage-intensive applications such as Augmented Reality (AR)<sup>11</sup> and Virtual Reality (VR)<sup>12</sup> become feasible.

However, the fast growing of computation demands pose severe burden on cloud centres<sup>13,14,15</sup>, and tremendous amount of data generated<sup>16,5</sup> congests the network with limited bandwidth<sup>17,14,15</sup>, hence causing bottlenecks for the cloud-based computing

paradigm. To relieve the pressure on cloud centres, the *edge computing* concept has emerged<sup>18,14</sup>, which allows computation to be performed at the edge network. The *Edge network*<sup>15</sup> refers to the computing and network resources sit along the path between data sources and cloud centres. The rationale of *computation offloading*<sup>19,20</sup> is to let the computation happen at closer proximity to the data sources, so that not only the load pressure of cloud centres can be lessened, but also the Quality of Service (QoS) can be improved as the edge computing can provide more efficient responses<sup>21</sup>.

To fully excavate the potential of edge-cloud computation offloading, several past research efforts investigated performance-influencing factors and proposed optimisation models to maximise the performance gain while not causing significant costs. Wang et al.,<sup>22</sup> presented a Two-Phase Optimisation algorithm and an Iterative Improvement algorithm to jointly optimise the computation costs and latency under the mobile-edge setting. Li et al.,<sup>23</sup> constructed a cost graph to optimise the energy consumption of handheld computing devices during computation offloading to achieve considerable energy saving. Works in<sup>22,24,25,26</sup> all considered optimising the communication cost under the edge-cloud task offloading setting. Specifically, they adopted a homogeneous communication model with two assumptions: cloud-cloud and edge-edge communication cost were ignored, and the edge-cloud and cloud-edge communications were assumed to have symmetric costs, irrespective of the communication direction and distance between nodes. To address the over-simplicity of homogeneous communication models in these methods, Du et al.,<sup>19</sup> attempted the heterogeneous communication model and proposed HETO algorithm that can jointly minimise the computation, communication and migration costs during computation offloading. Their work was the first to propose a heterogeneous communication model so that the deficiencies in existing researches can be overcome.

Despite various attempts of past researches to optimise the edge-cloud computation offloading problem, these models still suffered from the following drawbacks:

- These methods were not fine-grained enough. Although some methods considered heterogeneous communication cost, these methods failed to leverage more fine-grained factors such as the distance between node pairs. These methods also only leveraged a homogeneous cost model for computation tasks, which did not reflect the task heterogeneity in real-world settings.
- During computation offloading, some methods did not pay attention to the load balancing, which can cause overloading on certain nodes.
- These methods were cost-oriented, which failed to jointly optimise the profit and cost during computation offloading.

To address these issues, we propose a novel edge-cloud computation offloading model which not only utilises the more realistic heterogeneous communication and computation cost model, but also considers the cost and profit heterogeneities of tasks. Hence, the proposed method jointly optimises the profit and cost yielded during computation offloading. The model is named as *PECCO*, which stands for “Profit and Cost-oriented Edge-Cloud Computation Offloading”.

Considering this optimisation problem is hard in nature and the objective of the *PECCO* model is not differentiable, we consider using the Moth-flame Optimisation (MFO) algorithm<sup>27</sup> to tackle the computation offloading problem. As a swarm-based algorithm, it is gradient-free and it balances exploration and exploitation. Besides, empirically it outperforms other swarm-based counterparts in terms of convergence speed<sup>27</sup>, etc., which make it suitable to be leveraged in this case. We therefore propose an improved Moth-flame optimiser (*PECCO-MFI*) that addresses several drawbacks of the original MFO and significantly boosts its optimisation effectiveness. Specifically, a density-aware Moth-flame initialiser is designed to fit under the edge-cloud computation offloading setting. A dynamic hierarchical flaming mechanism is applied to avoid the single flame matching which is more likely to cause local optima stagnation. Moreover, the lifetime of moths is introduced to promote exploration when the corresponding paired flame is eliminated.

In summary, this paper makes the following contributions:

- We construct a profit and cost-oriented edge-cloud computation offloading optimisation model *PECCO* that jointly optimises both the heterogeneous profit of computation tasks and the heterogeneous cost produced during computation offloading.
- We not only utilise the heterogeneous communication cost, but also consider the load balancing among nodes during optimisation.
- We realise the suitability to leverage the Moth-flame Optimiser, and propose an improved algorithm which tackles several deficiencies of the original MFO and hence boosts the effectiveness when solving the proposed computation offloading model.

The rest of the paper is organised as follows, Section 2 introduces some related works on both edge-cloud computation offloading models and optimisation algorithms. The research opportunities are then discussed. The background, suitability and room for improvements to the Moth-flame Optimisation algorithm are given in Section 3. Section 4 presents the details of the proposed model, as well as how the Moth-flame Optimisation algorithm is improved and integrated. Section 5 presents the experimental settings and results to verify the effectiveness of the proposed algorithm when tackling the proposed model. Section 6 concludes the paper.

## 2 | RELATED WORK

In this section, past research works on edge-cloud computation offloading models will be presented. Then, some well-known optimisers will be presented and compared. Finally, the research opportunities of our work are discussed.

### 2.1 | Offloading Model

As a promising technique that can relieve the burden posed on cloud centres, edge-cloud computation offloading has drawn huge attention from both industry and academic community<sup>8,28</sup>. Wu et al.,<sup>25</sup> formulated the edge-cloud computation offloading problem into a graph min-cost partitioning problem, in which computation tasks will be partitioned to be run on either the cloud side or the edge side. The proposed Min-Cost Offloading Partitioning (MCOP) algorithm took both the execution time and energy consumption into account when deciding an optimal task partitioning strategy. Li et al.,<sup>23</sup> put forward a partition scheme to offload computation tasks on handheld devices. A cost graph was constructed and the partition scheme was applied to split computation programs into server tasks and client tasks with the aim to reduce the energy consumption. Juttner et al.,<sup>24</sup> presented the Lagrange Relaxation based Aggregated Cost (LARAC) algorithm, which formulated a task graph and traversed the shortest path between nodes when considering the communication costs. The proposed algorithm was effective on delay-sensitive applications, justified by the simulation experiment they performed.

In works completed by Wang et al.,<sup>22</sup> and Dong et al.,<sup>26</sup>, they paid attention to the communication cost faced in the edge-cloud computation offloading problem. When modelling the communication cost, communications between nodes on the same side (cloud-cloud, or edge-edge) were assumed to be cost-free. Moreover, to simplify the model, communication costs were assumed to be symmetric, i.e., cloud-edge and edge-cloud communications have the same cost, irrespective of direction and the distance between nodes on different sides. The homogeneous communication model they leveraged is considered to be over-simplified and highly infeasible in real-world settings, as the cost can be asymmetric and distance-dependent. Therefore, Du et al.,<sup>19</sup> proposed a more fine-grained heterogeneous cost model, in which the symmetric assumption was relaxed, and the communication costs between nodes in a single side were no longer ignored. They then formulated the problem as a graph partitioning problem and designed the HETO algorithm to find a sub-optimal offloading strategy. Experiments on PageRank datasets testified to the excellent performance of the HETO algorithm when minimising the communication, computation and migration costs.

Despite that various research efforts have been drawn to optimise the edge-cloud computation offloading, they still suffered from some drawbacks which need to be addressed:

- Although the heterogeneous communication cost has been considered in some works, they failed to leverage more fine-grained factors such as distance between node pairs.
- When considering the cost during computation offloading, these methods utilised a homogeneous cost model for computation tasks, i.e., task heterogeneity was ignored.
- During computation offloading, some methods did not take load balancing into consideration, i.e., some node may be overloaded.
- These methods were cost-oriented, which failed to jointly optimise the profit and cost during computation offloading.

### 2.2 | Model Optimiser

A suitable optimiser is indispensable to tackle the edge-cloud computation offloading problem and find out an excellent offloading strategy. Some well-known individual-based optimisation algorithms were proposed<sup>29,30</sup>. They only optimised a single

search candidate, and hence enjoyed a lighter computation cost and required less function evaluations. For instance, Lawrence<sup>31</sup> presented the Hill Climbing (HC) algorithm which iteratively improved a single search candidate by changing its variables. The Iterated Local Search (ILS) algorithm proposed by Lourenco et al.,<sup>32</sup> was an improvement towards the HC algorithm. The best solution obtained in each iteration was perturbed and utilised as the starting point of the next iteration. Despite the efficiency enjoyed by these algorithms, they suffered a lot from the local optima stagnation. These algorithms may encounter the premature convergence, which prevents them from converging towards the global optima. Some more advanced algorithms such as gradient descent<sup>33</sup> have also been widely applied, especially for the optimisation in the field of deep learning<sup>34,35</sup>. However, these methods required gradient information of the objective function, which made them not applicable when the objective function is not differentiable.

In order to provide better local optima avoidance, some population-based optimisation algorithms have been proposed and became popular in the past few years. By utilising multiple search candidates and meanwhile balancing between exploration and exploitation, they provided higher possibilities to approach the global optima. As a sub-class of population-based algorithms, swarm-based algorithms<sup>36</sup> utilised multiple search candidates for the purpose of exploration. These search candidates then iteratively evolve, and eventually the healthier individuals will survive, making the exploitation become possible. Kennedy et al.,<sup>37</sup> presented the Particle Swarm Optimisation (PSO) algorithm that mimicked the behaviour of birds in a flock which keep track of their individual and global best positions. The PSO involved only primitive math operations and was computationally inexpensive. Yang<sup>38</sup> proposed the Firefly Optimisation algorithm (FFA), which was inspired by fireflies. During flying, fireflies are attracted by other fireflies with higher brightness. The effectiveness of the algorithm was verified on several test functions. A Whale Optimisation algorithm (WOA) was proposed by Mirjalili et al.,<sup>39</sup> which was inspired by the bubble-net hunting strategy of humpback whales. The WOA algorithm mathematically modelled this behaviour to guide optimisation. A Grey Wolf Optimiser (GWO) was also proposed by Mirjalili et al.,<sup>40</sup> which modelled the social hierarchy of grey wolves during hunting to guide the optimisation process. Besides, Mirjalili<sup>27</sup> put forward the Moth-flame Optimisation algorithm (MFO), which was one of the most famous swarm-based optimisers. The MFO utilised a population of moths to act as search candidates so that the probability of approximating the global optima was increased. Inspired by the transverse orientation characteristic of moths, the location of moths will be updated based on their transverse oriented path, with extra parameters controlling the exploration and exploitation. Each search candidate was iteratively assessed by a fitness function and hence the MFO algorithm was gradient-free. After several generation of evolvments, the fittest moth will be regarded as the optimised result. Experiments on several benchmarks<sup>27</sup> demonstrated that compared with several counterparts, the MFO algorithm can achieve better optimisation results with statistical significance, while also converge in a fast manner.

### 2.3 | Research Opportunity

Considering that past edge-cloud computation offloading models suffered from these aforementioned drawbacks, we find it promising to propose an optimisation model that is both profit and cost-oriented. In terms of costs, the heterogeneous communication cost should be considered in a fine-grained manner. Moreover, a heterogeneous cost model should also be applied for tasks to make the model more practical. Besides, during computation offloading, load balancing should be taken care of to avoid computation node overloading.

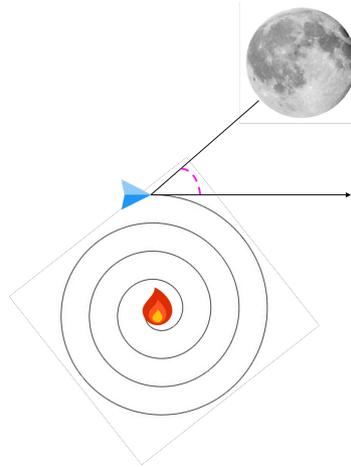
A comprehensive optimisation model and an excellent optimiser are both indispensable to produce a better task offloading strategy. Given the suitability of the MFO algorithm such as a higher chance to converge towards the global optima and its gradient-free merit, we propose an improved Moth-flame optimiser that addresses some design flaws of the original MFO, which can boost the effectiveness when working on the proposed computation offloading model.

## 3 | BACKGROUND

In this section, we firstly introduce the background of the Moth-flame Optimisation (MFO) algorithm, including how it works, its advantages and its suitability to be utilised to solve the *PECCO* model. Then, some deficiencies of the MFO algorithm are pointed out which provide room for improvements for its improved version.

### 3.1 | The Moth-flame Optimisation Algorithm

**Motivation and Rationale** As a nature-inspired optimiser, the Moth-flame Optimisation algorithm is population-based as it involves a population of moths. The moth has a special navigation mechanism called *transverse orientation*, which they use as a flight path maintaining method. As shown in upper portion of the conceptual Figure 1, the moth attempts to maintain a fixed angle (marked in pink) between its flying direction and the moon, so that they can fly in a relatively straight path since the moon is far away from the moth. However, as illustrated in the lower part of Figure 1, the moth can sometimes confuse the artificial light with the moon. Then, it will try to maintain the transverse orientation mechanism with the light which is much closer than the moon, leading to the entrapment towards the light and eventually hit it.



**FIGURE 1** Flight mechanism of the moth. The upper portion illustrates the *transverse orientation* mechanism. The lower portion illustrates the artificial light entrapment. The moth is represented using the blue arrow.

Inspired by this phenomenon, the Moth-flame Optimisation regards moths as search candidates, and treats the lights (flames) as potential optimal solutions. The Moth-flame Optimisation algorithm mimics the transverse orientation mechanism and hopes that the moth can reach the most optimal flame, which is regarded as the approximation to the global optima. By utilising a population of moths instead of a single one, the Moth-flame Optimiser possesses higher chance to avoid local optima entrapment and hence better approximates the global optima.

**General Framework** The Moth-flame Optimisation algorithm works under the general framework of swarm-based algorithm<sup>41</sup>. The species population will firstly be initialised, then they will keep evolving, eliminating individuals with bad fitness and updating until the termination criteria are reached. Eventually, the fittest individual will survive and will be treated as the optimal solution.

**Formulation** The Moth-flame Optimisation algorithm involves  $n$  moths, each is a search candidate wandering in the search space. Each moth  $M_n \in \mathbb{R}^d$  is a  $d$  dimensional vector, where  $d$  is the number of features to be optimised. Hence, it leads to the moth matrix  $M$  with dimension  $n \times d$ , represented as follows:

$$M = \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_n \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,d} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,d} \end{bmatrix} \quad (1)$$

A fitness function  $f$  is required to evaluate the fitness of each moth  $M_n$  by taking it as input, and returns its fitness, i.e., the objective value. The objective function has the following formulation:

$$f : \mathbb{R}^d \rightarrow \mathbb{R}, f(M_n) = OM_n \quad (2)$$

and hence, the corresponding fitness vector  $OM$  is defined as follows:

$$OM = \begin{bmatrix} f(M_1) \\ f(M_2) \\ \vdots \\ f(M_n) \end{bmatrix} = \begin{bmatrix} OM_1 \\ OM_2 \\ \vdots \\ OM_n \end{bmatrix} \quad (3)$$

In the Moth-flame Optimisation algorithm, the flames are not the real flames in the real world. Instead, they are set to be moths with top  $k$  highest fitness values that have the right to survive (as in line 9 in Algorithm 3), hence the flame matrix  $F$  has dimension  $k \times d$ . Without prior knowledge about which moth location is better, initially, the Moth-flame Optimisation algorithm randomly initialises the flame matrix  $F$  with  $k = n$ . During iterations, the  $k$  will be gradually decreased as the population evolves. The flame matrix  $F$  is represented as follows:

$$F = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_k \end{bmatrix} = \begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,d} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ f_{k,1} & f_{k,2} & \cdots & f_{k,d} \end{bmatrix} \quad (4)$$

and its corresponding fitness vector  $OF$  is defined as follows:

$$OF = \begin{bmatrix} f(F_1) \\ f(F_2) \\ \vdots \\ f(F_k) \end{bmatrix} = \begin{bmatrix} OF_1 \\ OF_2 \\ \vdots \\ OF_k \end{bmatrix} \quad (5)$$

The details on how the Moth-flame Optimisation algorithm is integrated in the *PECCO* model, i.e., what moth matrix  $M$  stands for, etc., will be explained in Section 4.5.2.

As is previously mentioned, if there is no knowledge about which initial position is better, then a random initialisation will be applied to generate both the moth matrix and the flame matrix using the following random generator:

$$m_{i,j} = (ub(i) - lb(i)) * random() + lb(i) \quad (6)$$

where  $ub$  and  $lb$  are the upper and lower bound of the range constraint, respectively. The  $random()$  function is a random number generator with range in  $[0, 1]$ .

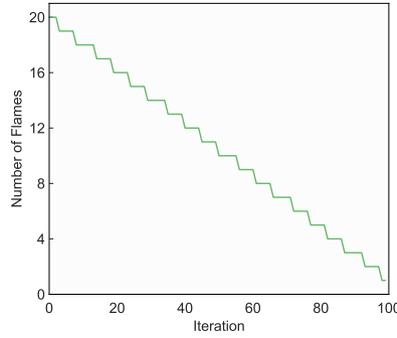
**Deficiency 1:** The Moth-flame Optimisation algorithm applies a random initialisation as it assumes there is no prior knowledge about which initial location is better. However, if prior knowledge presents, the random initialisation will degrade the performance. Besides, the random initialisation is not density-aware, i.e., the random initialisation may produce random vectors that are highly similar and hinder the diversity of the random population. An initial population with poor diversity will impair the benefit of population-based optimisers.

**Balancing Exploration and Exploitation** The Moth-flame Optimisation algorithm puts effort to balance between exploration and exploitation. Initially, there are  $n$  moths and  $n$  flames, each moth pursues its corresponding flame as illustrated by the solid arrows in Figure 3, which encourages exploration to avoid local optima stagnation as much as possible. During iterations, the moths will be sorted based on their fitness value in descending order, and the moths with top  $k$  highest fitness value will survive while other moths will be eliminated as shown in Algorithm 3. The value of  $k$  keeps decreasing based on the following formula during iterations so that the exploitation will be gradually emphasised:

$$k = round(n - CI * \frac{n-1}{MI}) \quad (7)$$

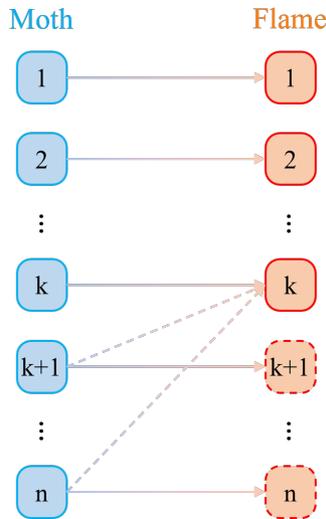
where  $n$  is the initial number of moth/flame,  $MI$  denotes the total number of iterations (Max iteration),  $CI$  stands for the current iteration. Eventually,  $k$  will decrease to 1, the last survived flame is regarded as the optimal solution produced by the Moth-flame Optimisation algorithm. The decreasing trend of  $k$  has been illustrated in Figure 2.

The number of flames  $k$  keeps decreasing while the number of moths  $n$  remains unchanged, the Moth-flame Optimisation algorithm therefore designs a moth-flame pairing mechanism as presented in Figure 3 so that the moths can decide which target flame is designated for them to pursue. At the beginning, the number of moths and flames are equal, i.e.,  $n$ , hence each moth will pursue its corresponding flame, i.e.,  $M_i \rightarrow F_i$ , as represented by the solid arrows in Figure 3. During iterations, the value of  $k$  will keep decreasing, hence the number of flames will be less than the number of moths. Under the MFO moth-pairing



**FIGURE 2** Illustration of the decreasing trend of the number of flames  $k$ , i.e., Equation 7.

mechanism, the moth  $M_i$  will still pursue its corresponding flame  $F_i$  if flame  $F_i$  still survives, or otherwise  $M_i$  will chase the last flame  $F_k$  as represented by the dashed arrows in Figure 3.



**FIGURE 3** The original moth-flame pairing mechanism. The moths are represented using blue boxes while the flames are represented using orange boxes.

Deficiency 2: During moth evolution, at any given time, moth  $M_i$  will always pursue a single designated flame, which increases the chance of being trapped in the local optima.

Deficiency 3: During moth evolution, those moths that have their corresponding flame being eliminated will always pursue the last surviving flame, which is the flame with the worst fitness. Neither pursuing the worst flame nor letting lots of moths pursuing a single flame is a reasonable design.

In terms of the moth updating mechanism, the Moth-flame Optimisation algorithm mimics the transverse orientation based on the following equation:

$$U(M_i, F_j) = D_{i,j} \times e^{bt} \times \cos(2\pi t) + F_j \tag{8}$$

where  $F_j$  is the paired flame designated for  $M_i$  to pursue,  $t$  is a random number in range  $[r, 1]$ ,  $r$  is a random number that will linearly decrease from  $-1$  to  $-2$  during iterations,  $b$  is the shape parameter, and  $D_{i,j}$  denotes the  $L1$ -distance between  $M_i$  and  $F_j$  which is defined as follows:

$$D_{i,j} = |F_j - M_i| \tag{9}$$



**FIGURE 4** Illustration of the exploration vs exploitation of the Moth-flame Optimisation algorithm.

The shape of an example updating path, i.e., the spiral shape, has been illustrated in Figure 1 and 4.

Specifically, the  $t$  parameter decides how close to the flame will the moth's terminal position be. As illustrated in Figure 4, a  $t$  value that is closer to 1 will let the moth ends up with a position that is farther from the flame (the blue shaded area), which emphasises exploration. On the other hand, a negative  $t$  value will draw the moth closer towards its target flame as indicated by the green shaded area in Figure 4, which encourages exploitation. Since  $t$  is within the range of  $[r, 1]$ , initially,  $r$  has value  $-1$  which promotes exploration by avoiding the moth to be too close to the flame. As the process evolves,  $r$  linearly decreases from  $-1$  to  $-2$ , which gradually encourages exploitation over exploration.

**Termination** The termination criterion is when there is only one flame remaining. It will be treated as the optimal solution.

**Advantage and Applicability** In summary, the Moth-flame Optimisation algorithm has the following advantages which make it applicable in our case:

- Since the *PECCO* model is hard in nature, applying this population-based algorithm with multiple search candidates while enabling the balance between exploration and exploitation will possess higher chance to approximate the global optima.
- Since the objective function in the *PECCO* model is not differentiable, the Moth-flame Optimiser becomes applicable as it evaluates each search candidate using the fitness function and therefore is gradient-free.
- Compared with its counterparts, the Moth-flame Optimiser achieves superior optimisation results and converges in an efficient manner.

**Room for Improvements** As is aforementioned, the Moth-flame Optimisation algorithm suffers from three deficiencies, which leave us with room for improvements. We propose three new mechanisms to tackle these deficiencies as follows:

- The profit, cost and density-aware initialiser → Deficiency 1
- The dynamic hierarchical flaming mechanism → Deficiency 2
- The lifetime-enabled moth-flame pairing strategy → Deficiency 3

Together, these mechanisms form the improved *PECCO-MFI* algorithm. The details will be presented in Section 4.5.1.

## 4 | MODEL AND METHOD

In this section, the problem formulation will be provided, followed by the presentation of the proposed *PECCO* optimisation model, in which the profit and cost component of the *PECCO* model will be explained. Then, we will explain how the Moth-flame Optimisation algorithm is improved and integrated to form our *PECCO-MFI* algorithm.

## 4.1 | Problem Formulation

In the edge-cloud environment, there are cloud nodes and edge devices (nodes), with a connection topology to form a connected graph. Hence, we formulate the problem as a graph  $G = (V, E)$  where  $V$  stands for a set of cloud/edge nodes and  $E$  represents a set of communication links. There are in total  $N$  computing nodes, in which it contains  $I$  cloud nodes and  $J$  edge nodes, hence we have

$$\begin{aligned} V^C &= \{V_1^C, V_2^C, \dots, V_I^C\}, V^E = \{V_{I+1}^E, V_{I+2}^E, \dots, V_{I+J}^E\} \\ V &= V^C \cup V^E, V^C \cap V^E = \emptyset, N = I + J \end{aligned} \quad (10)$$

Note that we can simplify the notation  $V_n^X$ ,  $X \in \{C, E\}$  to be  $V_n$  as the range of subscript  $n$  can tell whether the node belongs to the cloud or the edge.

For each computing node  $V_n$ , it has the following properties. Firstly, each computing node is capable of handling certain capacity of computation tasks. Hence  $Cap_{V_n\_max}$  denotes the maximum number of units of computation workload that node  $V_n$  is capable of handling, while  $Cap_{V_n\_min}$  stands for the minimum workload of node  $V_n$  when it is idle. We assume that no node can be overloaded by computation tasks. By considering the capacity of each node, it can also indirectly model other performance factors such as power consumptions.

As for edges  $E$  in the graph  $G$ , there are in total  $Q$  edges, denoted as  $E_q$ , or interchangeably  $E_{\langle V_s, V_t \rangle}$ , which stands for edge  $E_q$  is an edge that starts from node  $V_s$  and points to node  $V_t$ . To make the heterogeneous model more generalisable, the length of each edge is also considered instead of being ignored as in<sup>19,24</sup>, and is denoted as  $L_{E_q}$  (or  $L_{E_{\langle V_s, V_t \rangle}}$  using the interchangeable notation).

In terms of tasks to be executed, there are in total  $K$  of them, each task  $T_k$  has a property  $WL_{T_k}$  that represents how many units of computation workload does task  $T_k$  have. Each task can only be allocated on either a cloud node or an edge node, and a task is allowed to stay if its initial allocation is good enough. Hence, we define  $\mathcal{A}_{T_k}^I$  and  $\mathcal{A}_{T_k}^O$  to be the initial and offloaded allocation of task  $T_k$ , which satisfies

$$\mathcal{A}_{T_k}^I, \mathcal{A}_{T_k}^O \in V^X, X \in \{C, E\} \quad (11)$$

Moreover, vector  $\mathcal{A}^O$  is defined to represent the offloaded allocations of all  $K$  tasks as follows:

$$\mathcal{A}^O = \begin{bmatrix} \mathcal{A}_{T_1}^O \\ \mathcal{A}_{T_2}^O \\ \vdots \\ \mathcal{A}_{T_K}^O \end{bmatrix} \quad (12)$$

Besides, the *PECCO* model applies a heterogeneous cost model for tasks. Instead of applying a homogeneous task cost as in<sup>19</sup>, for each task, it has different costs  $C_{T_k}^C$  if it is executed on the cloud, or  $C_{T_k}^E$  if being allocated to the edge. By utilising a heterogeneous cost model for each task, it can reflect that different tasks can have different costs when being allocated to different sides, which makes the model more realistic. To make the model profit-oriented, each task is also associated with two profits, i.e.,  $P_{T_k}^C$  and  $P_{T_k}^E$ , which stand for the profit gained of completing task  $T_k$  on the cloud and edge, respectively. By jointly considering profit and cost, it makes the proposed *PECCO* model become profit and cost-oriented.

## 4.2 | PECCO Cost Model

The *PECCO* model is a multi-factored model that jointly considers both the generalised heterogeneous communication cost and the heterogeneous computation cost.

### 4.2.1 | Generalised Heterogeneous Communication Cost Model

Considering that previously proposed communication cost models in past researches suffered from several drawbacks (e.g., applied the unrealistic symmetric and cost-free assumption, failed to consider communication distance, applied homogeneous communication costs between node pairs, etc.) which made them become hardly generalisable in practice, it naturally leads to the rationale of our generalised heterogeneous communication cost model.

There are in general four types of communication costs, i.e.,  $w^{CC}$ ,  $w^{EE}$ ,  $w^{CE}$  and  $w^{EC}$ . The  $CE$  here for instance represents the communication from a cloud node to an edge node. Inside each type of communication cost, it can also have different costs between different nodes, which is denoted as  $w_{\langle V_s, V_t \rangle}^{XX}$ . For example,  $w_{\langle V_s, V_t \rangle}^{EC}$  denotes the communication cost from edge node

$V_j^E$  to the cloud node  $V_i^C$ . As different nodes may work under different conditions like being operated by different service providers, the communication cost between node pairs can be different even if they are situated on the same side. Therefore, the model is more realistic in practice. This kind of generalisation also offers convenience to represent communication failures for instance, by setting the edge-wise communication cost to be a large value. The communication cost function  $C_{E_{\langle V_s, V_t \rangle}}$  is formulated as follows:

$$C_{E_{\langle V_s, V_t \rangle}} = \text{sum}(L_{E_{\langle V_s, V_t \rangle}} \times [1_{V_s, V_t \in V^C}, 1_{V_s, V_t \in V^E}, 1_{V_s \in V^C, V_t \in V^E}, 1_{V_s \in V^E, V_t \in V^C}] \odot [w_{\langle V_s, V_t \rangle}^{CC}, w_{\langle V_s, V_t \rangle}^{EE}, w_{\langle V_s, V_t \rangle}^{CE}, w_{\langle V_s, V_t \rangle}^{EC}]) \quad (13)$$

inside it, for instance,  $1_{V_s \in V^C, V_t \in V^E}$  will return 1 if  $V_s$  is a cloud node and  $V_t$  is an edge node, and will return 0 otherwise, other indicator functions carry the similar meaning. The  $\odot$  represents the element-wise multiplication operator and  $\times$  is the scalar multiplication operator. Hence, the communication cost function  $C_{E_{\langle V_s, V_t \rangle}}$  will return the length of the inputted edge times the corresponding cost of that type of communication so that the heterogeneous communication costs between node pairs can be considered.

After defining the generalised heterogeneous communication cost model, the optimal cost path between any pairs of computing nodes can be pre-computed using the shortest path algorithm. The optimal cost path between node  $V_i$  and  $V_j$  is denoted as  $OCPP_{\langle V_i, V_j \rangle}$ , and therefore we can define the optimal communication cost from node  $V_i$  to node  $V_j$  as follows:

$$COMM_{\langle V_i, V_j \rangle} = \sum_{E_{\langle V_s, V_t \rangle} \in OCPP_{\langle V_i, V_j \rangle}} C_{E_{\langle V_s, V_t \rangle}} \quad (14)$$

and therefore, the total communication cost is defined as follows:

$$COMM(G, T, \mathcal{A}^I, \mathcal{A}^O) = \underset{\mathcal{A}^O}{\text{argmin}} \left\{ \sum_{k=1}^K COMM_{\langle \mathcal{A}_{T_k}^I, \mathcal{A}_{T_k}^O \rangle} \right\} \quad (15)$$

The optimisation algorithm should find an optimal offloading strategy  $\mathcal{A}^O$  to offload task  $T_k$  so that it can achieve a communication cost  $COMM(G, T, \mathcal{A}^I)$  as low as possible. In summary, the proposed generalised heterogeneous communication cost model overcomes the drawbacks of previously proposed communication models and has the following benefits:

- It no longer ignores the communication cost between nodes on the same side (i.e., cloud-cloud, edge-edge).
- The asymmetry between communication costs is considered, cost from cloud to edge and from edge to cloud communication can be heterogeneous.
- It considers distances when modelling communication cost between two nodes.
- It allows different node pairs to have heterogeneous communication costs.

## 4.2.2 | Heterogeneous Computation Cost Model

Next, the heterogeneous computation cost model is defined which also takes the heterogeneities between computing tasks into account. Generally, for each task  $T_k$ , it possesses cost  $C_{T_k}^C$  and  $C_{T_k}^E$ , which is the cost of executing task  $T_k$  on the cloud and edge, respectively. Due to the diversity of tasks,  $C_{T_k}^C$  is not necessarily lower than  $C_{T_k}^E$ . The previously proposed homogeneous computation cost model for tasks is infeasible, it is unreasonable to assume that all tasks share exactly the same computation cost when being executed on a single side. Hence, the computation cost model for tasks we considered in the *PECCO* is more generalisable. The overall computation cost is formulated as follows:

$$COMP(G, T, \mathcal{A}^O) = \underset{\mathcal{A}^O}{\text{argmin}} \left\{ \sum_{k=1}^K (1_{\mathcal{A}_{T_k}^O \in V^C} \times C_{T_k}^C + 1_{\mathcal{A}_{T_k}^O \in V^E} \times C_{T_k}^E) \right\} \quad (16)$$

where the indicator function  $1_{\mathcal{A}_{T_k}^O \in V^C}$  will return 1 if the allocation for task  $T_k$   $\mathcal{A}_{T_k}^O$  is a cloud node, and will return 0 otherwise, similar for  $1_{\mathcal{A}_{T_k}^O \in V^E}$ .

## 4.3 | PECCO Profit Model

Different from previously proposed works, the proposed *PECCO* model is not only cost-oriented, but also profit-oriented. For each task  $T_k$ , it has profit  $P_{T_k}^C$  and  $P_{T_k}^E$  when being executed on the cloud and edge, respectively. The overall profit is formulated

as follows:

$$PROFIT(G, T, \mathcal{A}^O) = \operatorname{argmin}_{\mathcal{A}^O} \left\{ \sum_{k=1}^K (1_{\mathcal{A}_{T_k}^O \in V^C} \times P_{T_k}^C + 1_{\mathcal{A}_{T_k}^O \in V^E} \times P_{T_k}^E) \right\} \quad (17)$$

#### 4.4 | Overall Optimisation Objective

Finally, the *PECCO* optimisation model will integrate the aforementioned cost and profit model to become profit and cost-oriented. The objective function is defined as follows:

$$Obj(G, T, \mathcal{A}^I) = \operatorname{argmin}_{\mathcal{A}^O} \{ (COMM(G, T, \mathcal{A}^I) + COMP(G, T, \mathcal{A}^O)) + \lambda \times PROFIT(G, T, \mathcal{A}^O) \} \quad (18)$$

where  $\lambda$  is a ratio parameter being set to a negative value to integrate the profit into the objective to be minimised. By having  $\lambda$ , the objective function can minimise the cost and simultaneously maximise the profit.

By jointly optimising the profit and cost-oriented optimisation model *PECCO*, we can find a solution that can jointly optimise costs and the profit as much as possible.

#### 4.5 | The *PECCO-MFI* Optimiser

In this section, we will introduce the proposed improved Moth-flame Optimiser with detailed explanations to the improvements we made. Then, how the improved Moth-flame Optimisation algorithm is integrated to optimise the *PECCO* model is explained, i.e., what moths stand for in the *PECCO-MFI* algorithm, how are tasks offloaded based on the optimised allocation strategy  $\mathcal{A}^O$ , etc.

##### 4.5.1 | Algorithm Improvement

To tackle the deficiencies mentioned in Section 3.1 and therefore boost the performance, we propose an improved Moth-flame Optimiser called *PECCO-MFI* with three improvements to tackle three deficiencies, respectively.

**Improvement 1 (Profit, Cost and Density-aware Moth Initialiser):** To tackle the Deficiency 1: trivial random initialisation, we design a new moth initialiser that is profit, cost and density-aware, as shown in Algorithm 1.

Profit and Cost-awareness: The newly designed moth initialiser will allocate tasks (elements in each moth) to cloud or edge side based on their profit and cost. It is natural to allocate services to the side in which they possess a lower profit-cost objective than the other side. Conversely, if the allocation is done in the reversed way, then this task is likely to be migrated during the optimisation process, which therefore incurs unnecessary costs. The profit and cost-aware moth initialisation mechanism has been shown in line 7 - 13 in Algorithm 1. By utilising this prior knowledge, the improved Moth-flame Optimisation algorithm is reasonable to outperform its knowledgeable random counterpart.

Density-awareness: The rationale of applying the population-based paradigm is to maximise the chance of approximating the global optima as close as possible. However, if some moths are initialised to have a close proximity, the benefit of the population-based paradigm will be greatly hindered. To ensure the initialised moths have rich diversity, the newly designed initialiser will generate more moth vectors than required, then it will iteratively remove the closest pair of moths and keep the average of them. The procedure has been given in line 15 - 19 in Algorithm 1. The removal will be continued until the number of moth vectors is satisfied as required. By leveraging this mechanism, moth vectors that are initialised to be too close will be merged into a new one to prevent the performance degradation from happening. Therefore, the proposed moth initialiser is density-aware.

**Improvement 2 (Dynamic Hierarchical Flaming Mechanism):** To deal with the Deficiency 2: Single moth-flame pairing, a dynamic hierarchical flaming mechanism is applied. As pursuing a single flame will lead to higher risk of being trapped in local optima, inspired by the social hierarchy possessed in the moth species, the moths with top 3 highest fitness values will be regarded as leaders, which will provide guiding reference for other moths to pursue. Hence, instead of pursuing a single flame, in the newly designed algorithm, each moth will chase the linear combination of its designated flame and the leader flames, as shown in line 3 - 8 in Algorithm 2.

Exploration and Exploitation Balance: At the beginning of the training process, putting too much dependency on the top 3 flames will incur risk of local optima stagnation as the performance of flames at the beginning is not promising enough. Hence, an adjusting factor  $\omega$  is introduced which linearly increases from 0 to 1 during the training process as indicated in line 2 in

---

**Algorithm 1** The profit, cost and density-aware moth initialiser  $initialiser(nsa, G, T, ub, obj, \mathcal{A}^I)$  of the *PECCO-MFI* Algorithm

---

**Input:**

Number of search candidates (moths)  $nsa$ ,  
 Edge-cloud graph  $G$ ,  
 Tasks  $T_k \in T$ ,  
 Allocation upper bound  $ub$ ,  
 Objective function  $Obj()$  as defined in Equation 18,  
 Initial allocation  $\mathcal{A}^I$

**Output:** Profit, cost and density-aware moth initialisation with dimension  $nsa \times N$

```

1: for  $T_k$  in  $T$  do
2:   Calculate costs based on Equation 15 and 16
3:   Calculate the profit based on Equation 17
4:   Calculate the profit and cost-oriented objective based on Equation 18
5: end for
6: for  $i$  in  $range(nsa \times 1.5)$  do
7:   for  $k$  in  $range(K)$  do
8:     if allocate task  $T_k$  to the cloud side yield a lower objective value then
9:       Store random number in range  $[0, \frac{ub}{2})$  into  $\mathcal{A}_i^I$ 
10:    else
11:      Store random number in range  $[\frac{ub}{2}, ub]$  into  $\mathcal{A}_i^I$ 
12:    end if
13:  end for
14: end for
15: while  $len(\mathcal{A}^I) \neq nsa$  do
16:   Find pair  $(\mathcal{A}_i^I, \mathcal{A}_j^I)$  with minimum intra-pair L2 distance
17:   Add  $\frac{\mathcal{A}_i^I + \mathcal{A}_j^I}{2}$  into  $\mathcal{A}^I$ 
18:   Remove both  $\mathcal{A}_i^I$  and  $\mathcal{A}_j^I$  from  $\mathcal{A}^I$ 
19: end while
20: return  $\mathcal{A}^I$ 

```

---

Algorithm 2. After applying the adjusting factor  $\omega$  as in line 5 and 7 in Algorithm 2, exploration will be encouraged at the beginning by putting less emphasis on the top 3 flames since initially the value of  $\omega$  is small. As the training progresses,  $\omega$  will gradually increase, which will emphasise more exploitation, since the guiding reference of top 3 flames will be gradually reinforced as the  $\omega$  keeps growing. As such, the utilisation of adjusting factor  $\omega$  in the newly designed dynamic hierarchical flaming mechanism will balance between exploration and exploitation.

**Improvement 3 (Lifetime-enabled Moth-Flame Pairing Strategy):** Finally, to solve the Deficiency 3: naive moth-flame pairing, we design a fairer pairing strategy as indicated in line 9 - 21 in Algorithm 2. Instead of letting all moths whose corresponding flames are eliminated to chase the last surviving flame, we introduce a lifetime parameter  $\tau$  with lifetime threshold set as 0.8. Due to the elimination of their unpromising flames, these moths are not promising themselves and hence the lifetime parameter  $\tau$  is used to decide whether certain moth will be re-initialised, i.e., starting a new lifetime. Hence, as indicated in line 11 - 12 in algorithm 2, if the randomly generated lifetime parameter  $\tau$  is higher than the lifetime threshold, the moth will start a new lifetime by pairing with a newly initialised flame. Otherwise, the moth will continue its lifetime, and the algorithm will let it to pair with a randomly selected survived flame to promote a fairer exploration. By randomly pairing with a survived flame, these moths will fairly explore all possible survived flames instead of all exploiting the worst-fitted flame. During the re-pairing process, the aforementioned dynamic hierarchical flaming mechanism will be utilised again to provide better exploration and exploitation balancing while enabling the guiding reference of top 3 flames as in line 16 - 20 in Algorithm 2. By utilising this enhanced moth-flame pairing strategy, the exploration of the algorithm will be further encouraged and hence leading to a higher chance to approach the global optima.

---

**Algorithm 2** The dynamic hierarchical flaming mechanism and the lifetime-enabled moth-flame pairer *enhanced\_pairer(CI, MI)* of the *PECCO-MFI* Algorithm

---

**Input:**

Current iteration  $CI$ ,  
Max iteration  $MI$

- 1: Define  $F_1, F_2, F_3$  as the flames with top 3 fitness values, respectively
- 2:  $\omega \leftarrow \frac{CI}{MI}$
- 3: **if** moth  $M_i$ 's corresponding flame  $F_i$  still survives **then**
- 4:   **if** there are  $\geq 3$  flames survive **then**
- 5:      $M_i$  will pursue  $\frac{F_i + \omega \times F_1 + \omega \times F_2 + \omega \times F_3}{1 + 3 \times \omega}$
- 6:   **else if** there are 2 flames survive **then**
- 7:      $M_i$  will pursue  $\frac{F_i + \omega \times F_1 + \omega \times F_2}{1 + 2 \times \omega}$
- 8:   **end if**
- 9: **else**
- 10:    $\tau \leftarrow \text{random}(0, 1)$ , where  $\tau$  is the lifetime parameter
- 11:   **if**  $\tau > 0.8$  **then**
- 12:      $F_I \leftarrow$  a randomly initialised flame
- 13:   **else**
- 14:      $I \leftarrow \text{random}(0, k)$
- 15:   **end if**
- 16:   **if** there are  $\geq 3$  flames survive **then**
- 17:      $M_i$  will pursue  $\frac{F_I + \omega \times F_1 + \omega \times F_2 + \omega \times F_3}{1 + 3 \times \omega}$
- 18:   **else if** there are 2 flames survive **then**
- 19:      $M_i$  will pursue  $\frac{F_I + \omega \times F_1 + \omega \times F_2}{1 + 2 \times \omega}$
- 20:   **end if**
- 21: **end if**

---

#### 4.5.2 | Integration of the Moth-flame Optimiser

We apply the improved Moth-flame Optimiser *PECCO-MFI* to optimise the *PECCO* model. The pseudocode of the algorithm has been presented in Algorithm 3. In the *PECCO-MFI* algorithm, each moth vector  $M_i$  is a  $1 \times K$  vector, where  $K$  is the number of tasks waiting to be allocated. The values in the moth vector  $M_i$  are within the range of  $[0, ub]$ , where  $ub$  is a constant. If the value is in range  $[0, \frac{ub}{2})$ , it indicates that this task will be executed to the cloud side, otherwise, this task will be offloaded to the edge side. Then, the algorithm will find the computing node with the cheapest communication cost in the designated side and allocate the task to that computing node. If the computing node will be overloaded by taking this task, the algorithm will find the node at the designated side with the second cheapest communication cost and so on. Eventually, the task will either be allocated to a computing node without causing overloading, or it will not be satisfied due to workload unavailability.

## 5 | EXPERIMENT

In this section, we will introduce our experimental setup and the dataset we utilised during experiments. Then, experimental results will be presented and explained to testify to the superiority of the proposed method. Specifically, objective values yielded by different methods are compared, followed by the comparison of profit, cost and profit-cost ratio to demonstrate the effectiveness of the *PECCO-MFI* algorithm in terms of joint profit and cost optimisation. Finally, to demonstrate the *PECCO-MFI* algorithm offloads computation tasks wisely, the task allocation and the resource utilisation are compared and analysed.

### 5.1 | Dataset, Parameter and Experimental Setup

**Dataset.** The dataset we use to simulate the edge-cloud environment<sup>42</sup> is the Sydney train station parking dataset obtained from the Open Data Portal provided by the New South Wales Government Department of Transportation<sup>43</sup>. The dataset contains the

**Algorithm 3** Workflow of the *PECCO-MFI* Algorithm**Input:**

Shape parameter  $b$ ,  
 Number of search candidates (moths)  $nsa$ ,  
 Allocation upper bound  $ub$ ,  
 Objective function  $Obj()$  as defined in Equation 18

**Output:** Resource allocation strategy  $\mathcal{A}^O$  of computation tasks, which is the best flame  $F$

```

1: Initialise moth matrix  $M \leftarrow \text{initialiser}(nsa, G, T, ub, obj, \mathcal{A}^I)$  (in Algorithm 1)
2:  $OM \leftarrow Obj(M)$ 
3: while  $len(F) \neq 1$  do
4:   Update  $k$ 
5:    $OM \leftarrow obj(M)$ 
6:   if  $CI = 1$  then
7:      $F \leftarrow M.sortBy(OM)$ 
8:   else
9:      $F \leftarrow M.sortBy(OM)[0 : k]$ 
10:  end if
11:  Update moth-flame pairing using  $enhanced\_pairer(CI, MI)$  (Algorithm 2)
12:  for  $i$  in  $range(nsa)$  do
13:    for  $j$  in  $range(K)$  do
14:      Update  $r$  and  $t$ 
15:      Calculate  $D$  with respect to the paired moth and flame using Equation (9)
16:      Update moth position using Equation (8)
17:    end for
18:  end for
19: end while
20:  $OF \leftarrow obj(F)$ 
21: return  $F, OF$ 

```

parking lot availability information at each train station in Sydney, Australia. Train stations in City of Sydney are treated as cloud nodes, and suburban train stations act as edge nodes. Each station has a parking with limited available parking lots, which represents the capacity of the node. The length of connected edges are the length of the shortest road between train stations. The dataset contains communication heterogeneity as paths between different nodes can have different charges due to factors such as toll roads, etc. Parking requests are simulated as tasks which will be allocated to nodes. We pick 20 stations from the City of Sydney to act as cloud nodes, and 30 stations from Sydney suburban areas as edge nodes, numbered from 1 to 20, and from 21 to 50, respectively. We utilise 200 tasks, each with certain amount of parking requests that will be treated as workloads. Parking in the city or in the suburban area can yield different parking fares, which serves as the cost of the task when being executed on the cloud side and edge side, respectively. Finally, successfully allocating each parking task will earn certain profit.

**Parameter Setting.** We now introduce parameter settings in two parts: *PECCO* optimisation model parameter settings, as well as improved Moth-flame Optimiser parameter settings.

*PECCO Optimisation Model Parameter Setting.* To reflect the communication heterogeneity, we set the communication cost  $w^{CC}$ ,  $w^{CE}$ ,  $w^{EC}$  and  $w^{EE}$  to have an average of 1, 2, 4 and 6, respectively. Thanks to the powerful network infrastructure equipped in cloud centres, the intra-cloud communication should be the cheapest among communication directions. The download cost should be cheaper than the upload cost and hence the cost is set to reflect this pattern. Finally, due to the limited network capacity between edge devices, it is natural to possess the highest communication cost.

The ratio parameter  $\lambda$  that balances between cost and profit is set to be  $-8$  so that the algorithm can jointly minimise the cost and profit times this negative ratio.

*PECCO-MFI Parameter Setting.* We set the default allocation upper bound  $ub$  to be 1, and the Moth-flame shape parameter  $b$  to be 1 to comply with  $ub$ . If the shape parameter  $b$  is set to be too small, then the shape of the spiral will be very tight and it will never allocate tasks to some nodes. On the other hand, if  $b$  is too large, then the spiral will be too wide and it may generate

offloading strategy which does not make sense. The default number of search candidates, i.e., moths, is set to be 30, and the number of iterations is set to be 100 to make the algorithm efficient. The default value of the threshold of the lifetime parameter  $\tau$  is set to be 0.8.

**Experimental Setup and Hardware Configuration.** We compare the *PECCO-MFI* algorithm with two edge-cloud computation offloading algorithms, including the LARAC algorithm, which traverses the shortest path during communication and optimises the computation cost, as well as GREEDY, which allocates each task to the side that will yield lower objective value, then greedily select the node which has the shortest distance from the initial location of the task. The effectiveness of the *PECCO-MFI* is also compared with 9 swarm-based optimisers, including Bat Algorithm (BAT)<sup>44</sup>, Sine Cosine Algorithm (SCA)<sup>45</sup>, Whale Optimisation Algorithm (WOA)<sup>39</sup>, Cuckoo Search Algorithm (CS)<sup>46</sup>, Firefly Algorithm (FFA)<sup>38</sup>, Particle Swarm Optimisation (PSO)<sup>37</sup>, Grey Wolf Optimisation (GWO)<sup>40</sup>, Differential Evolution (DE)<sup>47</sup> and the original Moth-flame Optimisation Algorithm (MFO)<sup>27</sup>. To make the experimental results concrete, all experiments are repeated 10 times and the average results are reported.

We implement the method using python 3.8, and conduct all experiments on a server equipped with Intel Core i9 9900K CPU and 32GB of memory.

Value \ Method	Method					
	LARAC	Greedy	BAT	SCA	WOA	CS
Overall Objective	-19253.93	-22615.6	-41317.38	-41496.97	-42489.74	-44088.8
Profit	2650.29	3054.72	5388.28	5407.45	5531.59	5737.87
Cost	1948.38	1822.16	1788.83	<b>1762.64</b>	1762.98	1814.15
Profit/Cost Ratio	1.36	1.67	3.02	3.07	3.14	3.17
Value \ Method	Method					
	FFA	PSO	GWO	DE	MFO	MFI
Overall Objective	-44741.63	-45546.13	-46494.95	-46598.59	-45953.91	<b>-48069.48</b>
Profit	5814.99	5915.65	6039.85	6050.26	5969.91	<b>6229.31</b>
Cost	1778.32	1779.07	1823.88	1803.52	1805.37	1765.0
Profit/Cost Ratio	3.28	3.34	3.32	3.36	3.32	<b>3.53</b>

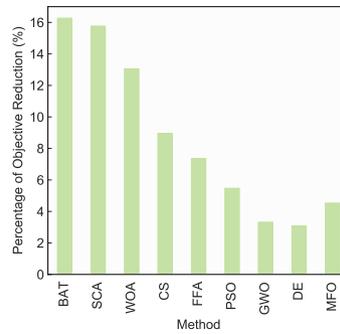
**TABLE 1** Objective value, profit, cost and the profit-cost ratio of different algorithms. Note the MFO stands for the original MFO algorithm, while the MFI stands for the improved *PECCO-MFI* algorithm.

## 5.2 | Comparison of Objective Optimisation between Algorithm

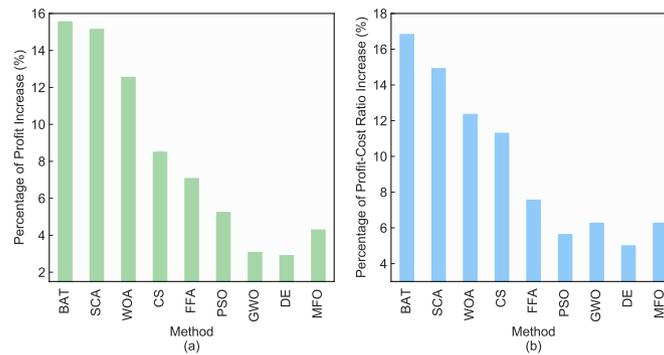
To verify the effectiveness of the *PECCO-MFI* algorithm in terms of objective optimisation, the objective results of the *PECCO-MFI* and 11 compared methods are listed in Table 1. As we can notice, the proposed *PECCO-MFI* algorithm achieves the lowest objective value among all compared methods. As we can observe from Figure 5, the *PECCO-MFI* algorithm produces a 4.6% and 3.16% objective value reduction compared with the original Moth-flame Optimisation algorithm and the best-performed Differential Evolution algorithm when tackling this optimisation problem, demonstrating the effectiveness of the *PECCO-MFI* algorithm. The significant performance improvement achieved by the *PECCO-MFI* over the original Moth-flame Optimiser also verifies the effectiveness of improvements we made on the MFO.

## 5.3 | Comparison of Profit and Cost between Algorithms

After investigating the total objective, we now look at the profit and cost component. As indicated in Table 1 and Figure 6(a), the *PECCO-MFI* algorithm achieves the highest profit among all comparing methods. Specifically, the profit achieved is 4.35% and 2.96% higher than the original MFO algorithm, as well as the best-performed Differential Evolution, respectively. Besides, the *PECCO-MFI* achieves the second lowest cost during computation offloading, which is only 0.1% higher than the SCA algorithm, who has the lowest cost. However, when it comes to the profit-cost ratio, the *PECCO-MFI* yields the best performance. As indicated in Figure 6(b), the *PECCO-MFI* algorithm achieves significant profit-cost ratio boost compared with all other methods.



**FIGURE 5** The percentage of objective value reduction achieved by the *PECCO-MFI* algorithm compared with other methods. Since the objective value reduction yielded by the *PECCO-MFI* is 149.7% and 112.6% compared with LARAC and GREEDY, respectively. Therefore, for better visualisation, these two methods are omitted from the plot.



**FIGURE 6** The percentage of increase on profit and profit-cost ratio achieved by the *PECCO-MFI* algorithm compared with other methods are presented in sub-figure (a) and (b), respectively. Since the profit achieved by the *PECCO-MFI* is 135% and 103.9% higher than LARAC and GREEDY, respectively, and the profit-cost ratio is 159.6% and 111.4% higher than LARAC and GREEDY, respectively. Therefore, for better visualisation, these two methods are omitted from plots.

The higher the profit-cost ratio is, the more profit will be yielded by spending one unit of cost, i.e., the computation offloading is wiser as it can achieve higher profit by spending unit amount of cost. According to Figure 6(b), the *PECCO-MFI* has a profit-cost ratio that is 6.33% and 5.06% superior than the original MFO and Differential Evolution counterparts, respectively, which demonstrates the effectiveness of the *PECCO-MFI* in terms of profit and cost-oriented offloading optimisation. It also indicates that the *PECCO-MFI* can draw a computation offloading strategy that can utilise cost wisely to produce excellent profit.

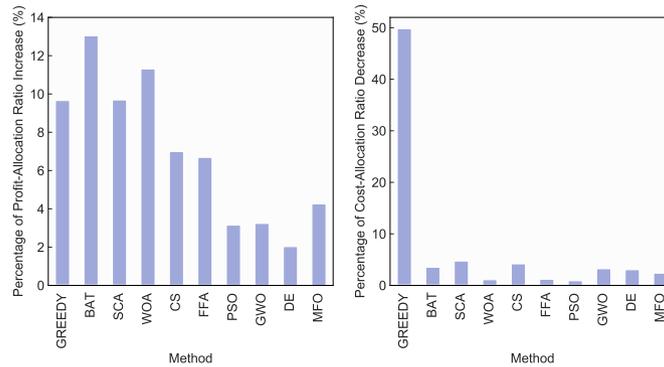
#### 5.4 | Comparison of Task Allocation between Algorithms

We now focus on task allocation done by different algorithms. If offloading in an unwise manner, some computing nodes will be overloaded, causing some tasks failed to be allocated. As we can see from Table 2, the proposed *PECCO-MFI* algorithm achieves the highest task allocation number. A high number of tasks being allocated after computation offloading indicates that the *PECCO-MFI* algorithm can offload tasks wisely without causing severe overloading. Hence, most number of tasks can be successfully allocated and completed instead of being stuck on overloaded computing nodes.

In terms of the profit-allocation ratio, both Table 2 and Figure 7 show that except for the extreme case LARAC due to poor task allocation, the *PECCO-MFI* algorithm produces the highest profit-allocation ratio. The higher the profit-allocation ratio is, the more profit will be yielded by completing each task. As visualised in Figure 7(a), the *PECCO-MFI* algorithm achieves 4.26% and 2.03% higher profit-allocation ratio compared with the original MFO and the best-performed Differential Evolution,

Method	LARAC	Greedy	BAT	SCA	WOA	CS
#Allocation	64.9	92.9	175.3	170.7	177.2	176.7
Profit/Allocation Ratio	<b>40.96</b>	31.69	30.74	31.68	31.22	32.48
Cost/Allocation Ratio	30.02	19.61	10.20	10.33	9.95	10.27
Method	FFA	PSO	GWO	DE	MFO	MFI
#Allocation	178.6	179.2	179.3	177.6	179.1	<b>179.3</b>
Profit/Allocation Ratio	32.57	33.69	33.66	34.06	33.33	34.75
Cost/Allocation Ratio	9.96	9.93	10.17	10.15	10.08	<b>9.84</b>

**TABLE 2** Number of computation tasks being allocated, the profit-allocation ratio and the cost-allocation ratio of different algorithms.



**FIGURE 7** The percentage of profit-allocation ratio increase and cost-allocation ratio decrease achieved by the *PECCO-MFI* algorithm compared with other methods are shown in (a) and (b), respectively. The extreme case LARAC is omitted for better visualisation.

respectively. Besides, according to Table 2, the *PECCO-MFI* yields the lowest cost-allocation ratio, which indicates the *PECCO-MFI* causes the lowest cost to satisfy each task, demonstrating its cost-effectiveness. Hence, by achieving the highest number of task allocation, a high profit-allocation ratio and a low cost-allocation ratio, the effectiveness of the offloading strategy yielded by the *PECCO-MFI* algorithm is verified.

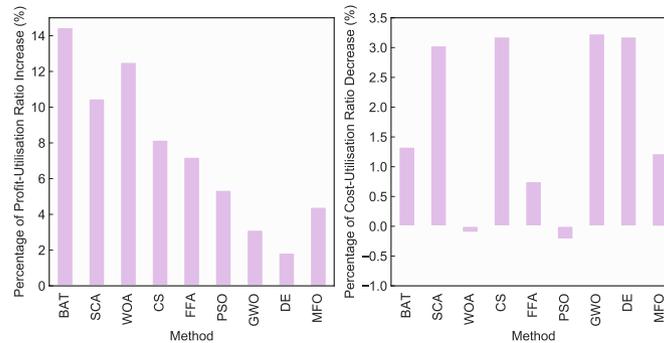
Method	LARAC	Greedy	BAT	SCA	WOA	CS
Utilisation	121%	112%	95%	92%	95%	95%
Profit/Utilisation Ratio	22.12	32.51	56.97	59.02	57.95	60.28
Cost/Utilisation Ratio	<b>16.10</b>	16.27	18.83	19.16	18.56	19.10
Method	FFA	PSO	GWO	DE	MFO	MFI
Utilisation	95%	96%	95%	94%	96%	95%
Profit/Utilisation Ratio	60.82	61.89	63.23	64.02	62.45	<b>65.2</b>
Cost/Utilisation Ratio	18.72	18.53	19.20	19.19	18.81	18.58

**TABLE 3** Average computing node workload utilisation, the profit-utilisation ratio and the cost-utilisation ratio of different algorithms.

## 5.5 | Comparison of Resource Utilisation between Algorithms

Finally, the computing node workload resource utilisation, the profit-utilisation and cost-utilisation ratio are indicated in Table 3. As we can observe, except the LARAC and GREEDY algorithm which overloads some computing nodes, all other methods produce offloading strategy that is free from overloading.

As we can see from Figure 8, the profit-utilisation ratio produced by the *PECCO-MFI* algorithm is significantly higher than all other compared methods. Specifically, the *PECCO-MFI* algorithm achieves a 4.4% and 1.8% increase in terms of profit-utilisation ratio compared with the original MFO and Differential Evolution, respectively. The higher the profit-utilisation ratio is, the more profit will be yielded by utilising one unit of computing node resource. On the other hand, the *PECCO-MFI* yields a relatively low cost-utilisation ratio, which means the algorithm will not incur a high cost by utilising one unit of computation resource. Hence, the excellent profit-utilisation and cost-utilisation ratio indicate the effectiveness of the *PECCO-MFI* algorithm, i.e., producing a computation offloading strategy that can utilise computation resource wisely to achieve a high profit and a low cost, without overloading any computing nodes.



**FIGURE 8** The percentage of profit-utilisation ratio increase and cost-utilisation ratio decrease achieved by the *PECCO-MFI* algorithm compared with other methods are shown in (a) and (b), respectively. The extreme cases LARAC and GREEDY are omitted from the plot.

## 6 | CONCLUSION

In this paper, we propose a profit and cost-oriented edge-cloud computation offloading model *PECCO* which jointly considers the heterogeneous communication and computation cost, as well as the profit yielded after computation offloading. An improved Moth-flame Optimisation algorithm with three improvements is proposed which addresses several deficiencies of the original MFO and is then integrated to produce an optimised edge-cloud computation offloading strategy, forming the *PECCO-MFI* algorithm. Comprehensive experiments are conducted and the *PECCO-MFI* algorithm is compared with several other baseline methods to testify to the effectiveness of the *PECCO-MFI* algorithm when optimising the edge-cloud computation offloading model, as well as the effectiveness of the improvements made over the original MFO.

## ACKNOWLEDGEMENT

This work is supported in part by Key-Area Research and Development Program of Guangdong Province (2020B010164002) and Zhejiang Provincial Natural Science Foundation of China (LZ22F020002).

## DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

## References

1. Shafique K, Khawaja BA, Sabir F, Qazi S, Mustaqim M. Internet of things (IoT) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5G-IoT scenarios. *Ieee Access* 2020; 8: 23022–23040.
2. Shen S, Huang L, Zhou H, Yu S, Fan E, Cao Q. Multistage Signaling Game-Based Optimal Detection Strategies for Suppressing Malware Diffusion in Fog-Cloud-Based IoT Networks. *IEEE Internet of Things Journal* 2018; 5(2): 1043-1054. doi: 10.1109/JIOT.2018.2795549
3. Zhang K, Tian J, Xiao H, Zhao Y, Zhao W, Chen J. A Numerical Splitting and Adaptive Privacy Budget Allocation Based LDP Mechanism for Privacy Preservation in Blockchain-Powered IoT. *IEEE Internet of Things Journal* 2022: 1-1. doi: 10.1109/JIOT.2022.3145845
4. Li T, Wang H, He D, Yu J. Blockchain-based Privacy-preserving and Rewarding Private Data Sharing for IoT. *IEEE Internet of Things Journal* 2022: 1-1. doi: 10.1109/JIOT.2022.3147925
5. Wu J, Wang Y, Fan X, Ye K, Xu C. Toward fast theta-join: A prefiltering and amalgamated partitioning approach. *Concurrency and Computation: Practice and Experience*; n/a(n/a): e6996. doi: <https://doi.org/10.1002/cpe.6996>
6. Marjani M, Nasaruddin F, Gani A, et al. Big IoT data analytics: architecture, opportunities, and open research challenges. *IEEE Access* 2017; 5: 5247–5261.
7. Li M, Wu J, Dai J, et al. A self-contained and self-explanatory DNA storage system. *Scientific Reports* 2021; 11(1): 1–15.
8. Shakarami A, Ghobaei-Arani M, Masdari M, Hosseinzadeh M. A survey on the computation offloading approaches in mobile edge/cloud computing environment: a stochastic-based perspective. *Journal of Grid Computing* 2020; 18(4): 639–671.
9. Li Q, Zhang Q, Huang H, Zhang W, Chen W, Wang H. Secure, Efficient and Weighted Access Control for Cloud-assisted Industrial IoT. *IEEE Internet of Things Journal* 2022: 1-1. doi: 10.1109/JIOT.2022.3146197
10. Shen Y, Shen S, Wu Z, Zhou H, Yu S. Signaling game-based availability assessment for edge computing-assisted IoT systems with malware dissemination. *Journal of Information Security and Applications* 2022; 66: 103140. doi: <https://doi.org/10.1016/j.jisa.2022.103140>
11. Ren J, He Y, Huang G, Yu G, Cai Y, Zhang Z. An edge-computing based architecture for mobile augmented reality. *IEEE Network* 2019; 33(4): 162–169.
12. Zhang W, Chen J, Zhang Y, Raychaudhuri D. Towards efficient edge cloud augmentation for virtual reality mmogs. In: SEC '17. Association for Computing Machinery; 2017: 1–14.
13. Mao Y, You C, Zhang J, Huang K, Letaief KB. Mobile edge computing: Survey and research outlook. *arXiv preprint arXiv:1701.01090* 2017.
14. Yu W, Liang F, He X, et al. A survey on the edge computing for the Internet of Things. *IEEE access* 2017; 6: 6900–6919.
15. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: Vision and challenges. *IEEE internet of things journal* 2016; 3(5): 637–646.
16. Zhao Y, Chen J. A Survey on Differential Privacy for Unstructured Data Content. *ACM Comput. Surv.* 2021. Just Accepteddoi: 10.1145/3490237
17. Shi W, Dustdar S. The promise of edge computing. *Computer* 2016; 49(5): 78–81.
18. Khan WZ, Ahmed E, Hakak S, Yaqoob I, Ahmed A. Edge computing: A survey. *Future Generation Computer Systems* 2019; 97: 219–235.
19. Du M, Wang Y, Ye K, Xu C. Algorithmics of cost-driven computation offloading in the edge-cloud environment. *IEEE Transactions on Computers* 2020; 69(10): 1519–1532.
20. Wang J, Pan J, Esposito F, Calyam P, Yang Z, Mohapatra P. Edge cloud offloading algorithms: Issues, methods, and perspectives. *ACM Computing Surveys (CSUR)* 2019; 52(1): 1–23.
21. Mach P, Becvar Z. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials* 2017; 19(3): 1628–1656.

22. Wang W, Zhou W. Computational offloading with delay and capacity constraints in mobile edge. In: IEEE. ; 2017: 1–6.
23. Li Z, Wang C, Xu R. Computation offloading to save energy on handheld devices: a partition scheme. In: CASES '01. Association for Computing Machinery; 2001: 238–246.
24. Juttner A, Szviatovski B, Mécs I, Rajkó Z. Lagrange relaxation based method for the QoS routing problem. In: . 2. IEEE. ; 2001: 859–868.
25. Wu H, Knottenbelt W, Wolter K, Sun Y. An optimal offloading partitioning algorithm in mobile cloud computing. In: Springer. ; 2016: 311–328.
26. Dong L, Wang F, Shan J. Computation offloading for mobile-edge computing with maximum flow minimum cut. In: CSAE '18. Association for Computing Machinery; 2018: 1–5.
27. Mirjalili S. Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-based systems* 2015; 89: 228–249.
28. Liu J, Wang X, Shen S, Yue G, Yu S, Li M. A Bayesian Q-Learning Game for Dependable Task Offloading Against DDoS Attacks in Sensor Edge Cloud. *IEEE Internet of Things Journal* 2021; 8(9): 7546–7561. doi: 10.1109/JIOT.2020.3038554
29. Shi H, Liu S, Wu H, et al. Oscillatory Particle Swarm Optimizer. *Applied Soft Computing* 2018; 73: 316–327. doi: <https://doi.org/10.1016/j.asoc.2018.08.037>
30. Lai X, Zhou Y. Analysis of multiobjective evolutionary algorithms on the biobjective traveling salesman problem (1, 2). *Multimedia Tools and Applications* 2020; 79(41): 30839–30860.
31. Davis L. Bit-climbing, representational bias, and test suit design. In: ; 1991: 18–23.
32. Lourenço HR, Martin OC, Stützle T. Iterated local search. In: Springer. 2003 (pp. 320–353).
33. Ruder S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* 2016.
34. Goodfellow I, Bengio Y, Courville A. *Deep learning*. MIT press . 2016.
35. Li S, Xie B, Wu J, Zhao Y, Liu CH, Ding Z. Simultaneous Semantic Alignment Network for Heterogeneous Domain Adaptation. In: Association for Computing Machinery; 2020: 3866–3874.
36. Keerthi S, Ashwini K, Vijaykumar M. Survey paper on swarm intelligence. *International Journal of Computer Applications* 2015; 115(5).
37. Kennedy J, Eberhart R. Particle swarm optimization. In: . 4. IEEE. ; 1995: 1942–1948.
38. Yang XS, He X. Firefly algorithm: recent advances and applications. *International journal of swarm intelligence* 2013; 1(1): 36–50.
39. Mirjalili S, Lewis A. The whale optimization algorithm. *Advances in engineering software* 2016; 95: 51–67.
40. Sm A, Smm B, Al A. Grey Wolf Optimizer. *Advances in Engineering Software* 2014: 46–61.
41. Mirjalili S. Genetic algorithm. In: Springer. 2019 (pp. 43–55).
42. Huang D, Fan X, Wang Y, He S, Xu C. DP\_Greedy: A Two-Phase Caching Algorithm for Mobile Cloud Services. In: IEEE. ; 2019: 1–10.
43. Transportation N. Commuter Carparks TfNSW Open Data Hub and Developer Portal. <https://opendata.transport.nsw.gov.au/dataset/commuter-carparks>; . Accessed: 2021-07-01.
44. Yang XS, Gandomi AH. Bat algorithm: a novel approach for global engineering optimization. *Engineering computations* 2012.
45. Mirjalili S. SCA: a sine cosine algorithm for solving optimization problems. *Knowledge-based systems* 2016; 96: 120–133.
46. Yang XS, Deb S. Engineering optimisation by cuckoo search. *International Journal of Mathematical Modelling and Numerical Optimisation* 2010; 1(4): 330–343.
47. Price KV. Differential evolution. In: Springer. 2013 (pp. 187–214).