Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Bi-Objective simplified swarm optimization for fog computing task scheduling

Wei-Chang Yeh[a], Zhenyao Liu[a*] and Kuan-Cheng Tseng[a]

[a]*Integration and Collaboration Laboratory, Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan*

C H R O N I C L E

Article history:
Received March 1 2023
Received in Revised Format May 10 2023
Accepted July 31 2023
Available online July, 31 2023

Keywords:
Fog Computing
Task Scheduling
Local Search
Simplified Swarm Optimization
Multi-Objective
Non-Dominated Sorting

A B S T R A C T

In the face of burgeoning data volumes, latency issues present a formidable challenge to cloud computing. This problem has been strategically tackled through the advent of fog computing, shifting computations from central cloud data centers to local fog devices. This process minimizes data transmission to distant servers, resulting in significant cost savings and instantaneous responses for users. Despite the urgency of many fog computing applications, existing research falls short in providing time-effective and tailored algorithms for fog computing task scheduling. To bridge this gap, we introduce a unique local search mechanism, Card Sorting Local Search (CSLS), that augments the non-dominated solutions found by the Bi-objective Simplified Swarm Optimization (BSSO). We further propose Fast Elite Selecting (FES), a ground-breaking one-front non-dominated sorting method that curtails the time complexity of non-dominated sorting processes. By integrating BSSO, CSLS, and FES, we are unveiling a novel algorithm, Elite Swarm Simplified Optimization (EliteSSO), specifically developed to conquer time-efficiency and non-dominated solution issues, predominantly in large-scale fog computing task scheduling conundrums. Computational evidence reveals that our proposed algorithm is both highly efficient in terms of time and exceedingly effective, outstripping other algorithms on a significant scale.

1. Introduction

As the era of the Internet of Things (IoT) unfolds, the staggering volume of data generated from smart devices such as mobile phones, automobiles, wearable devices, and more, is set to create a seismic shift in the data landscape. According to the International Data Corporation (IDC), by 2025, an estimated 80 billion interconnected devices will have produced an astronomical 180 trillion gigabytes of fresh data. This proliferation of data poses a significant challenge for traditional cloud computing services, which may struggle to handle the sheer volume and, subsequently, face increasingly extended response latencies. Many IoT applications demand real-time or low latency responses (Yannuzzi et al., 2014), thus amplifying the urgency to address these constraints. Conventional cloud computing (Perera et al., 2017), may no longer be able to accommodate the massive influx of data from multitudes of IoT devices and respond within acceptable latency timescales. Therefore, the advent of fog computing represents a new computing paradigm, decentralizing the cloud structure, and offering a promising solution to the latency challenges posed by this unprecedented data deluge.

The concept of fog computing was first clearly articulated by Bonomi et al. (2012). They characterized the fog computing paradigm, painting a vision where it functions as an intermediary computing power positioned between the cloud and the users. It aids the cloud in shouldering the hefty computational demands emerging from an extensive array of smart devices. However, it's essential to understand, as pointed out (Matt & Engineering, 2018), that fog computing is not replacing cloud computing. Instead, it is viewed as a complementary structure that extends the cloud computing services right to the network

* Corresponding author
E-mail: liuzhenyao49@gmail.com (Z. Liu)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)

doi: 10.5267/j.ijiec.2023.7.004

edge, as detailed by Bitam et al. (2018). Furthermore, Vaquero et al. defined fog computing as a situation typically comprising numerous ubiquitous, heterogeneous, and decentralized devices (Vaquero & Rodero-Merino, 2014). These devices communicate and potentially cooperate amongst themselves, often without the need for third-party intervention. Chiang et al. explored the challenges and opportunities of fog, highlighting how fog computing addresses the inherent issues within the IoT framework (Chiang & Zhang, 2016). They also underlined the critical issue of End-to-End architectural tradeoffs, a significant topic our study seeks to discuss.

The architectural landscape of fog computing is indeed diverse and various models have been proposed in recent research. Here, we present a broad overview of the general structure, as depicted in Fig. 1. Fundamentally, this architecture consists of three distinct layers: the cloud layer, the fog layer, and the device layer.

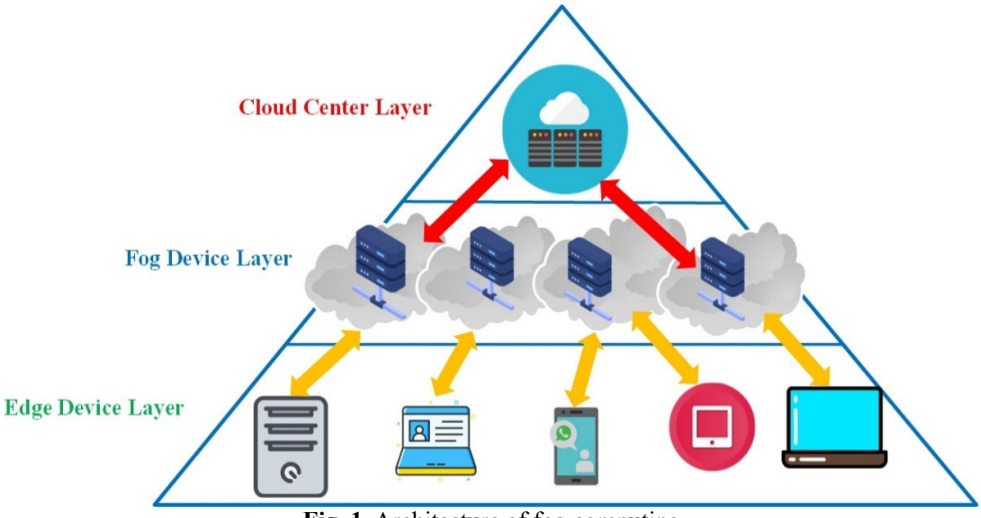


Fig. 1. Architecture of fog computing

The top of fog computing architecture, cloud center layer, is responsible for storing data, analyzing and decision making. Massive amounts of data are transmitted through this layer and sent to appropriate fog devices according to the results of the scheduling algorithm. Fog device layer consists of network devices such as routers, access points, gateways and switches. They are distributed among edge devices and cloud centers, and responsible for collecting raw data from edge devices or analyzing information from the cloud center. In addition, fog servers can store sensed data and process real-time analyses. Moreover, fog servers can preprocess raw data before transmitting to the cloud center. Edge devices contain a variety of IoT devices and devices with CPU processors, e.g., laptops, smart vehicles, smart phones. Edge devices are distributed geographically and usually not fixed. They receive data when a specified event occurs, then collect and send data to the upper layer server, fog server for immediate response or storage. There are some features of fog computing which made it different from cloud computing, such as low latency and location awareness, wide-spread geographical distribution, mobility, very large number of nodes, predominant role of wireless access, strong presence of streaming, real time applications and heterogeneity (Bonomi et al., 2012). Some of these features have changed some properties of task scheduling in fog computing. First and foremost, low latency and location awareness facilitate the broker to assign the tasks to the nearby fog devices. Besides, a vast number of nodes made the problem large and complex. A study compared the processing cost and transmission cost between fog and cloud computing paradigms against different numbers of terminal nodes (Sarkar & Misra, 2016), and it shows that fog computing costs significantly less than cloud computing. Moreover, the impact on cost reduction becomes more obvious when the number of terminal nodes rises. Finally, heterogeneity created the conflict between makespan and cost. That is, if a task is arrived at, the scheduler will assign it to the cloud for a shorter timespan or it will violate Service Level Agreement (SLA) and the service provider will be penalized. However, it will lead to the cost being high instead.

Though fog computing is a simple concept that distributes the computation load to local areas, much research hasn't been done yet. For instance, algorithm simulation time is too long for a large distribution system. Most of the research is utilizing linear optimization methods to schedule in the fog computing paradigm to acquire high-quality solutions. Nevertheless, the linear optimization algorithms require considerably much more heavy computation burden than any other machine learning algorithms, e.g., Simplified Swarm Optimization (SSO), Genetic Algorithm (GA), Particle Swarm Optimization (PSO), to converge and lack of flexibility when the objective functions were changed. In addition, the number of tasks in the fog computing paradigm is usually tremendous, that is, the computation burden will increase exponentially. Moreover, nearly all research applying linear optimization approaches has a critical constraint, the size of solution dimension, many of them set up a little cloud and fog device to acquire high-quality solutions within a limited time constraint. However, some fog computing applications are time sensitive. Such approaches are not applicable in such cases.

Recently, more and more efforts have been devoted to machine learning algorithms in fog computing task scheduling problem, e.g., Bee Swarm (Bitam et al., 2018), GA (Han et al., 2018), Evolutionary Algorithm (EA) (Binh et al., 2018), All of them focused on the algorithm application level but didn't pay attention to the simulation delay. Simulation delay is a critical issue in time-sensitive applications which is caused by the required time of the optimization algorithm to converge. Such problems will be worsened when the number of tasks and fog nodes increases. Though machine learning algorithms require relatively less time than linear optimization methods, the execution time is still time-consuming especially in large scale problems. Hence, we devote the efforts in algorithm time complexity reduction and parallel computation to reduce the impact of simulation delay.

Up to present, Bee swarm, GA and EA, algorithms which are good at discrete problems have been studied in this problem (Deng et al., 2016; Fard et al., 2012). However, these algorithms are time-consuming in multi-objective problems, due to the high time complexity of non-dominated sorting skills. In contrast, SSO based algorithms are not only strong in discrete problems but also run fast for it requires only one front to update, such as BSSO (Yeh, Zhu, et al., 2023). Furthermore, the structure of SSO is flexible, it could be varied to adapt to any kind of problem. In addition, SSO has demonstrated its powerful performance in cloud computing task scheduling problems (Yeh, Zhu, et al., 2023). For the shorter required time in converging and high flexibility in structure, we develop an efficient and effective strategy by means of SSO.

To the best of our understanding, the previous research in fog computing task scheduling problem usually combines two evaluators as one fitness value. One researcher set a predetermined balance coefficient α between makespan and cost then optimized the fitness value as close to 1 as possible (Binh et al., 2018). Another research set weights for memory and CPU execution time then combined them as a fitness evaluator (Bitam et al., 2018). Although these strategies have the advantage of shorter simulation time (because of the lower time complexity), the weights are not easy to determine at first. Moreover, one-fitness strategies produce one solution at a time. On the other hand, multi-objective optimization strategies produce a group of solutions that provide the decision maker with a variety of choices. However, it requires more time for its higher time complexity. Both two ways of strategy have their benefits and flaws. In this paper, we devote ourselves to a multi-objective optimization strategy to explore the future road of this strategy in this problem.

This research aims at accelerating the simulation time by minimizing time complexity and utilizing parallel computing. On top of that, we devote ourselves to developing a local search method to assist the algorithm to converge faster in this problem. In this study, we aimed to shorten the simulation delay in three ways. Firstly, time complexity, we proposed a one-front non-dominated sorting technique, Fast Elite Selecting, for the multi-objective algorithms that require only one front in solution updating. In such way, the time complexity is reduced from to to (N_F is the number of first front solutions among N solutions which is always less or equal to than N), and the speed of simulation could be raised. Secondly, for effectiveness, we proposed a novel local search method, Card Sorting Local Search, that helps the algorithm search highly potential areas where some non-dominated solutions might exist. Finally, we distribute the computation on four CPU threads, and we let each thread execute one independent optimization algorithm. In this way, the simulation delay is reduced by parallel computing.

The content of this research is organized in section 1, we introduce the background of fog computing including the reason for its emergence and some explicit definitions and concepts. Besides, motivation and purposes are depicted. Then we review the papers of task scheduling problems in fog computing, multi-objective algorithms and SSO in section 2. In section 3, the problem statement is presented. Section 4 illustrates the methodologies, Card Sorting Local search, Fast Elite Selection and BSSO. We evaluate the performance of the proposed algorithm against other multi-objective algorithms in section 5. Finally, we summarize the contribution and point out the future work in section 6.

2. Related Work

2.1. Task Scheduling Problem in Fog Computing

Deng et al. (2016) tackled the challenge of balancing power consumption and computational latency by breaking the problem down into three sub-issues. The first of these was finding an optimal compromise between computational latency and power consumption, achieved through the use of convex optimization techniques (He et al., 2014). The second sub-problem involved identifying the best tradeoff between power consumption and computational delay in cloud computing, where a nonlinear integer programming approach was applied (Li & Sun, 2006). The third and final issue aimed at minimizing communication delay in the WAN subsystem, treated as an assignment problem and addressed using the Hungarian method (Kuhn, 1955). However, a primary limitation of this study was the use of a centralized approach for optimization, reducing the delay and power consumption, which is a poor fit for a fog computing infrastructure. This approach could lead to a performance bottleneck at the central node during workload allocation, subsequently degrading the overall system performance.

Recognizing the performance bottleneck issue of the centralized optimization approach, alternative strategies have been explored. Bitam et al. proposed a bio-inspired optimization method known as the Bees Life Algorithm (BLA) to handle the job scheduling challenge in a fog computing environment (Bitam et al., 2018). This involved breaking jobs down into tasks

and allocating them across different fog devices considering factors such as CPU execution time and allocated memory size. During the foraging step, they employed a greedy local search process aiming to identify the optimal solution amongst various options. Despite demonstrating impressive performance in handling large-scale problems, this approach failed to consider specific attributes of the fog computing paradigm. For example, the tradeoff dilemma of whether to send tasks to the cloud, the communication costs incurred due to the distance between two distinct fog devices, and the penalties arising from Service Level Agreement (SLA) violations were overlooked.

Han et al. (2018) introduced an enhanced genetic algorithm for a hybrid cloud and fog computing infrastructure (Han et al., 2018). Here, the cost - incorporating the operational cost of virtual machines and the penalties from SLA violations - was considered as a performance evaluation metric. As a result, the impact of the makespan was simultaneously considered along with the penalty. However, like previous research, this study neglected to consider the tradeoff issue of whether to dispatch tasks to the cloud, creating a conflict between the makespan and cost that was not fully addressed.

2.1. Multi-Objective Algorithm in Task Scheduling Problem

In our understanding, the application of multi-objective algorithms in task scheduling within the realm of fog computing has been quite limited. Nevertheless, finding an optimal balance between makespan and cost within the fog computing paradigm is of paramount importance.

In this regard, Fieldsend et al. introduced a Multi-Objective Algorithm (MOA) to address the issue of conflicting metrics, applying a non-dominated tree to determine the global best for each particle (Fieldsend & Singh, 2002). Colleo et al. brought forward a multi-objective particle swarm optimization. Differing from other proposals that extended PSO to resolve multi-objective optimization problems, their algorithm employed an external repository of particles which subsequently guided the flight of other particles (Coello et al., 2004). Further, Zhou et al. proposed a Multi-Objective Evolutionary Algorithm (MOEA) to tackle the task scheduling problem in grid computing (Zhou et al., 2011). Liu et al. suggested a multi-objective genetic algorithm to resolve the task scheduling issue in cloud computing (Liu et al., 2013). Jena introduced Task Scheduling multi-objective nested Particle Swarm Optimization (TSPSO) for task scheduling, employing two performance evaluation metrics: power consumption and cost (Jena, 2015). Fard et al. proposed a Multi-Objective List Scheduling (MOLS) approach for workflow application scheduling in heterogeneous systems like Grids and Clouds (Fard et al., 2012). Basing on the Bi-objective Dynamic Level Scheduling algorithm (BDLS) aimed at maximizing reliability and minimizing execution time (Doğan & Özgüner, 2005), Yin (2018) proposed a Multi-Objective Simplified Swarm Optimization to address the conflict in cloud computing, taking into account both makespan and power consumption (Yin, 2018). Most recently, Yeh et al. suggested a Bi-Objective Simplified Swarm Optimization (BSSO) (Yeh, Zhu, et al., 2023), eliminating the *gBest* updating mechanism to encourage convergence in multi-objective cloud computing task scheduling problems. In our study, we likewise adopt BOSSO to address the task scheduling issue in the fog computing paradigm.

2.3. Simplified Swarm Optimization

Yeh's proposition of Simplified Swarm Optimization represents a novel, population-based stochastic optimization method. As a member of the swarm optimization family, it's recognized for its simplicity and efficiency, garnering significant interest from researchers. It has been effectively employed to solve discrete problems in numerous studies (Huang & Yeh, 2019; Yeh, 2009; Yeh, 2012, 2017; Yeh et al., 2011; Yeh, 2014; Yin, 2018). These instances of successful application demonstrate its potential in addressing complex optimization problems. And SSO has been applied to various problems, such as the redundancy allocation problems and reliability redundancy allocation problems(Jiang et al., 2023; Yeh, 2019, 2021; Yeh et al., 2021; Yeh, 2009; Yeh, 2017; Yeh et al., 2011; Yeh, 2014), quantum computing (Su et al., 2022), neural network hyperparameter optimization (Yeh, Lin, et al., 2023), Vehicle Routing Problem (Yeh & Tan, 2021), multi-level programming (Yeh et al., 2022) and so on.

In simplified swarm optimization algorithm, we set three parameters, C_g, C_p and C_w. where $C_g > C_p > C_w$. The update mechanism of SSO is defined by Eq. (1):

$$X_{ij}^{t} = \begin{cases} x_{ij}^{t-1} & \text{if} \quad \rho \in [0, C_w) \\ p_{ij}^{t-1} & \text{if} \quad \rho \in [C_w, C_p) \\ g_j & \text{if} \quad \rho \in [C_p, C_g) \\ x & \text{if} \quad \rho \in [C_g, 1] \end{cases} \tag{1}$$

Note that x_{ij}^{t} is the j^{th} variable of i^{th} solution at iteration t, ρ is a uniform random number within [0, 1], p_{ij}^{t-1} is j^{th} the variable of *pbest* (best i^{th} solution among t-1 iterations), g_j is j^{th} the variable of *gbest* (i.e. best solution among t-1 iterations) and x is a random variable between the lower bound and the upper bound of the feasible solution space.

For each update process, $\rho$is generated first. If ρ is located in [0, C_w), the value of variable will maintain the same as last generation. If ρ is located in [C_w, C_p), the value of the variable will be generated from *pbest*. If ρ is located in [C_p, C_g), the value of the variable will be generated from *gbest.* Otherwise, a random value, x, will be generated and replace the current variable.

In this study, we employ the Bi-objective Simplified Swarm Optimization (BSSO) as our primary approach for solution updates. BSSO, a recent innovation by Yeh (Yeh, Zhu, et al., 2023), extends the principles of the original Simplified Swarm Optimization (SSO) method (Yeh, 2009). The main distinction between Multi-objective Simplified Swarm Optimization and BSSO lies in the *gBest* updating mechanism. In BSSO, the *pBest* is removed and each non-dominated solution in the external archive is treated as a *gBest*. This approach enhances both the speed of convergence and solution diversity and has demonstrated superior performance over other notable algorithms such as MOPSO, MOSSO, and NSGA-II.

SSO is characterized by a straightforward update mechanism, the stepwise function, which can be adapted into different forms according to specific applications (Huang & Yeh, 2019; Yeh, 2009; Yeh, 2012, 2017; Yeh et al., 2011; Yeh, 2014; Yin, 2018). This flexibility extends to multi-objective problems as well. Unlike single-objective problems, multi-objective optimization problems do not have a singular *gBest* solution. Instead, each solution in the non-dominated solution archive is treated as equivalent to a *gBest* solution. Therefore, in each generation, BSSO randomly selects a non-dominated solution from the archive to serve as the *gBest* solution. The pseudocode for this procedure is presented on the next page:

Table 1
BSSO pseudo code

Proposed technique: Bi-Objective Simplified Swarm Optimization

Initialization:
population X= $\{X_1, X_2, \dots X_N\}$
non-dominated solution archive A= $\{A_1, A_2 \dots A_{N_F}\}$

```
1.  for gen= 0 to Ngen do
2.      Randomly pick a solution A_r to be gBest solution
3.      for sol=0 to Nsol do
4.          for var=0 to Nvar do
5.              r_1 =ρ ∈ [0,1]
6.              If (r_1<C_g) then
7.                  X_sol, var = A_r, var
8.              Else if (r_1 <C_w) then
9.                  continue
10.             Else
11.                 r_2=ρ ∈ [0, Nvm]
12.                 X_sol, var = r_2
13.
                end if
14.         end for
15.     end for
16.     Combine updated solutions X* with archive as X_Nsol +NF
        A* = Fast non-dominated sort (X_Nsol +NF)
17.     If (size of A* >predetermined archive size) then
18.
19.         A*= crowdingDistanceSelector(A*)
20.     end if
21. end for
```

Output: non-dominated solution archive A= $\{A_1, A_2 \dots A_{N_F}\}$

3. Problem Statement

3.1. System Model

A cloud system is composed of the cloud server and multiple fog devices. Each fog device is located at different areas. Each fog device receives requests from the users and upload the data information to cloud computing infrastructure. After receiving the task scheduling request, the cloud server runs the optimization procedure to determine assignments. When the algorithm is done, tasks are assigned to different processors, either fog devices or the cloud.

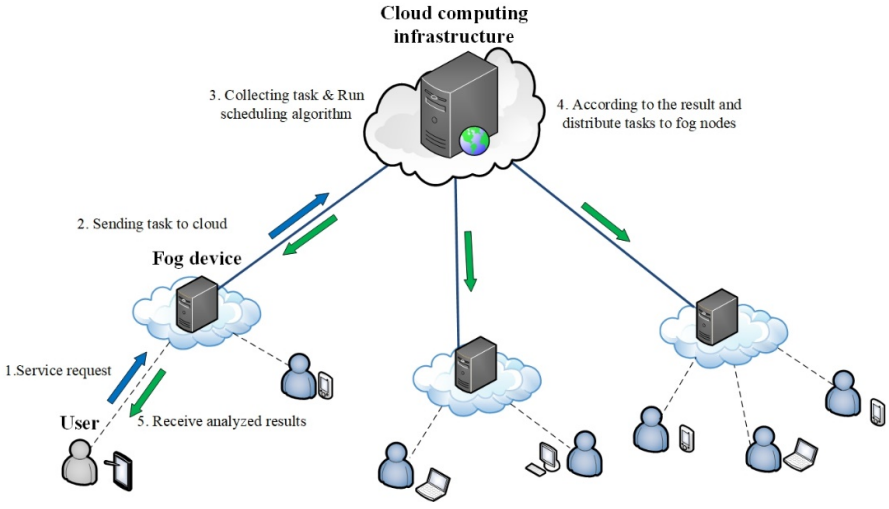


Fig. 2. System model

In this paper, virtual machine migration is not considered. Each task can be processed only on one processor, the fog devices are not allowed to halt and transfer tasks to other fog devices. Furthermore, one virtual machine can only process one task at once. The encoding of the solution is based on the assignment of each task. Each dimension represents the destination of a task. For example, assume the total number of tasks is 6, and there is a solution (3, 4, 1, 5, 2, 3). It means 1st task is assigned to processor 3 and 2nd task is assigned to processor 4, etc.

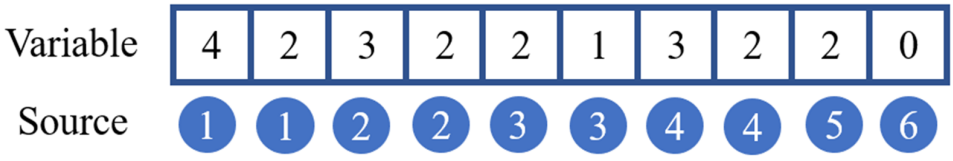


Fig. 3. Source locating

If the task is assigned to other processors away from its origin, a transmission cost is considered.

3.2. Notations

In BSSO, the integer variable represents the number of components in the node, and it is necessary to use multi-state BAT to find out all the state vectors $X = (x_1, x_2, \ldots, x_m)$, each state vector represent an integer combination. The traditional multi-state BAT algorithm is proposed by Yeh (Yeh, 2021). The pseudocode for traditional multi-state BAT is shown below:

In this subsection, we list notations in the following. 3.2.1 shows the indexes and coefficients; 3.2.2 shows the functions which are used for the calculation of objective functions.

Notations for mathematical model are listed and introduced as follows:

T_i: i^{th} task of set T.
P_j: j^{th} processor of set P.
TCU: transmission cost per unit distance.
P_o : fixed violation cost when the makespan of an assignment violated the SLA regulation.
P_t : variable violation cost

$n(P_j)$: number of tasks allocated on j^{th} processor.
$ET(T_i, P_j)$: the execution time of i^{th} task processed on j^{th} processor.
$ECU(P_j)$: the execution cost of j^{th} processor per unit time.

Notations for Card-Sorting BSSO are listed and introduced as follows:
$Ngen$: generation number.
$Nsol$: particle number of each generation.
$Nvar$: number of tasks of current service request.
Nvm: number of processors of the fog computing system.
gen: current BSSO generation
sol: current BSSO solution
var: current BSSO variable
C_g: a positive parameter which determines the probability of updating variable from non-dominated solutions.
C_w: a positive parameter which determines the probability of remaining original variable of the solution.
C_s: a positive parameter which determines the probability of card sorting current solution.

3.3. Mathematical Model

We formulate our problem in a mathematical model. An objective function is defined to evaluate the quality of a solution and our goal is to minimize two performance evaluation metrics, makespan and cost.

3.3.1 Model

Objective function:

$$Minmize \;\; Makespan = Execution\ time \tag{2}$$

$$Minimize\ Cost = Execution\ Cost + Idle\ Time\ Cost + Transmission\ Cost\ (+ Penalty) \tag{3}$$

3.3.2 Makespan

Makespan is determined by the completion time of the last task. The formula is shown as formula 4:

$$Makespan = \max_{j=1}^{M} Time_{P_j} = \max_{j=1}^{M} \sum_{i=1}^{n(P_j)} ET(T_i, P_j) \tag{4}$$

We calculate the execution time of i^{th} task processed j^{th} processor, and sum up for the total execution time of execution time of j^{th} processor, $Time_{P_j}$.Then determine the longest execution time as $Makespan_i$.

3.3.3 Cost

In this research, cost is composed of four elements which are execution cost, idle cost, transmission cost and penalty respectively. Execution cost is determined by the unit processing cost of processors multiplying execution time on each processor. Execution cost is determined by the unit processing cost of processors multiplying execution time on each processor. The execution cost formula is as shown in formula 5.

$$Cost = \sum_{j=1}^{M} Cost_{P_j} = \sum_{j=1}^{M} ECU(P_j) \times Time_{P_j} \tag{5}$$

We calculate $Time_{P_j}$as mentioned in makespan and multiply the execution cost per unit time of j^{th} processor as $Cost_{P_j}$, then sum up $Cost_{P_j}$. Then $Cost_i$ is calculated. Even when a processor is not tackling tasks, standby power consumption should be considered. This metric is aim to reduce some the occurence of extremely unbalance solutions which would lead the utilization of fog computing system low. Hence, we can untilize this evaluator to leverage the system utilization (Tasiopoulos et al., 2019)**.** The Idle Time Cost is shown as the formula 6.

$$\sum_{j=1}^{M} ECU(P_j) \times (Makespan - Time_{(P_j)}) \tag{6}$$

The processor idle cost is calculated by multiplying the cost of each processor by its idle time.

When a task is sent away from the local area, transmission cost should be considered. Specifically, the cost linearly rose with the increase of distance between two computational devices. We calculate the transmission cost by measuring the distance from the receiving fog node to processing fog node. In practice, we multiply unit transmission cost by a predetermined fog node distance matrix as shown in formula 7.

$$\sum_{j=1}^{M} TCU \times Distance_j(F_{Start}, F_{End}) \tag{7}$$

If the makespan of an assignment exceeds the deadline, penalty cost is generated. The penalty function is shown below as formula 8:

$$P_o + P_t \times (Makespan - Deadline) \tag{8}$$

P_o is the fixed penalty expense which is aroused when the makespan exceeds the deadline which was predetermined by contract.P_t is the variable cost of exceeding time.

4. Methodology

In this chapter, we present the methodologies used in this study. Firstly, we introduce a fast non-dominated sorting technique in 3.1. Secondly, we present the elite Multi-Objective Simplified Swarm Optimization in 3.2. In 3.3, we proposed a new local search method, Card Sorting Mechanism. Lastly, two performance metrics which measure the obtained Pareto front are presented.

4.1. Non-dominated Sorting

Multi-objective optimization usually refers to problems with two important metrics but conflict with each other. For example, in this study, an assignment with comparatively lower makespan usually costs more than other assignments with higher makespan. Hence, the ultimate goal of optimization in such case is to seek for a non-dominated solution set which combines a variety of assignment combination and close enough to the Pareto Front $P*$ which is the best non-dominated solution set in reality (Li, 2003). Non-dominated sorting is to distinguish the dominance relationship from each solution in set $\{X_1, X_2, \dots X_N\}$ and store as a descending order set $\{F_1, F_2, \dots F_K\}$. A proper non-dominated solution set satisfies the following conditions:

1. All the solutions in a certain front is non-dominated with each other.
$$\forall X_i, X_j \in F_k : X_i \not\prec X_j \text{ and } X_j \not\prec X_i, k = 1,2,\dots,k$$
2. Any front with higher index will dominate those with lower index.
$$\forall X \in F_k : \exists X' \in F_{k-1} : X' < X, k = 2,3,\dots,k$$

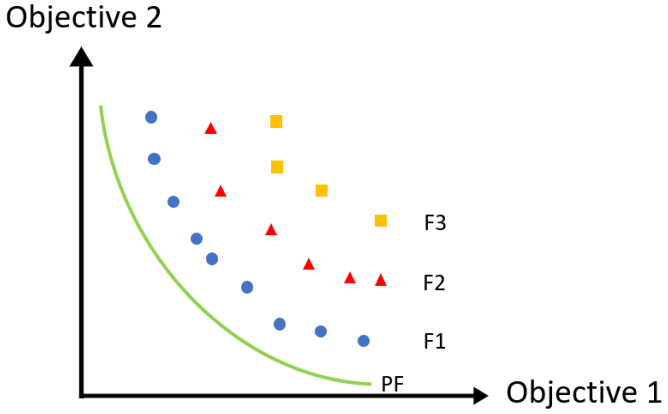


Fig. 4. Visualization of sorting result

Fig. 4 shows the dominance relationship of solution set *X*. In this case, we minimize both objectives 1 and 2. Hence, the solutions being closer to the lower-left corner is better. Those circle-marked solutions belong to front 1 which dominates the rest fronts in current solution set. Triangle-marked and rectangle-marked solutions belong to front 2 and 3 respectively. A popular non-dominated sorting algorithm, Fast non-dominated sorting was first proposed in 2002 (Deb et al., 2002). Each solution$p$ is compared with each other and store the comparison result by updating S_p or n_p. S_p is a set that stores solutions dominated by p. n_p is a counter that count the number of solutions dominating p. If solution p that dominate the compared solution, S_p will be increased by one. Otherwise, if solution p is dominated, its n_p will be increased by 1. For each front creation, it stores the solutions with n_p equals to 0. After all the iterations, fast non-dominated sorting is done. Fast non-

dominated sorting has a time complexity of $O(MN^2)$ is dominated, its n_p for it must conduct $MN(N-1)$ fitness comparisons (N is the population size). Moreover, it requires a space complexity of $O(N^2)$ to record two assistant indexes.

4.2. Fast Elite Selecting

To reduce computation burden and shorten the simulation time for task scheduling. We proposed Fast Elite Sorting technique (FES) for the multi-objective algorithm that requires only one front in each generation. The simple procedure is presented in Table 2.

Table 2
Fast Elite Selecting procedure

Proposed technique: Fast Elite Selecting

Input: Solution set X= $\{X_1, X_2, \dots X_N\}$
Initialization: DefaultFront1 = X //Assume all solutions are in front 1.

```
1. for each X_i ∈DefaultFront1 do
2.     for each X_j ∈DefaultFront1 do
3.         if (X_i dominate X_j) then
4.             remove X_j from DefaultFront1
5.         else if (X_j dominate X_i) then
6.             remove X_i from DefaultFront1 and break
7.         end if
8.     end
9. end
```

Output: Front1 solution set F= $\{X_1, X_2, \dots X_{N_{F1}}\}$

In this initialization of FES, we assume all the solutions are in front 1 and let **DefaultFront1** include all the solutions. Then, for each solution$X_i$in **DefaultFront1,** we compare it with other solution X_j and see if X_j is dominated. If X_j is dominated, it will be immediately removed from **DefaultFront1** because it is impossible to be in front 1. With the same idea, X_i is dominated by X_j instead, the iteration will be broken and go to the next **DefaultFront1 element** X_{i+1}. This step will continue until each element in **DefaultFront1** is iterated. After that, the survival of solutions in **DefaultFront1** is the winner being dominated by nobody. The advantage of FES is its time complexity. The complexity can be expressed as $O(MN_F{}^2)$. (M is the number of objective functions and N_F is the number of first front non-dominated solutions which is always less or equal to than N) The best case of time complexity of FES is $O(MN)$ where the first solution dominates all the others and end at first solution. On the other hand, the worst case lies in all the solutions are non-dominated with each other, then the time complexity will be $O(MN^2)$ which is the same time complexity of fast non-dominated sort.

It is because BSSO requires only one front for updating the solutions in each generation, so applying fast non-dominated sorting technique in BSSO becomes redundant for pairwise comparison. With the benefit of FES, computation burden can be significantly reduced especially when the size of solution set is large. The reason should be contributed to the remove right after comparison idea. We remove solutions being dominated from **DefaultFront1** right after comparison. Hence, comparison times can be reduced. On the other hand, fast non-dominated sorting requires N^2 on any conditions.

4.3. Card Sorting Local Search

4.3.1 Idea of Card Sorting Local Search (CSLC)

Card Sorting Fable:

Card Sorting is a procedure when you are playing poker games like Big two. In the beginning of Big two, the dealer shuffles the cards and deals 13cards to 4 players. Before the players check the cards, everyone expects their luck to be fair (non-dominated concept). However, after sorting the cards, some will realize that their cards are not better, but some lucky man will get a full house, four of a kind bomb or even straight flush (dominating solutions).

Explanation:

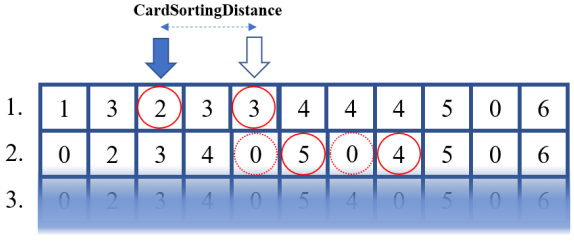
This story explicitly illustrated the idea of CSLS. Imagining that the cards each player gets is a non-dominated solution. Some cards will be found to be better after the card sorting procedure, but some do not. Like CSLS, it cannot assure any solution to be better after updating but it can make sure the solution will not be worse. It is worth mentioning that, CSLS doesn't require anything but the solution itself (Every player can only play his/her own cards). The detailed procedures are presented in Table 4.

4.3.2 The CSLS Operation Explanation

CSLS can start from the final output of any multi-objective algorithm, non-dominated solution set. In the initialization state, Lgen, the iteration time, is initialized based on the scale of the scenario. C_s, the card sorting percentage, is predetermined by ANOVA test. CardSortingDistance is also associated with the size of problems. About the above-mentioned parameters, we will discuss it in detail in the next section. In this section, we discuss the details of this approach and append the pseudo code in Table 3.

For each solution in non-dominated solution set archive *A,* we determine a random number r_1 with an interval [0, 1]. This random number is to determine the operation of this solution. If $r_1 < Cs$, then we do card sorting. If not, we force a random task of this solution to be processed on the cloud.

If $r_1 < Cs$, then we set *Sorting times* to be 0, and do card sorting. Note that for each solution, we need only one effective card sorting operation. This is the reason we used while here because this card sorting may not success at the first pick of the cards. For instance, if we get T_i and $T_{i+CardSortingDistance}$ and they are both originally assigned to P_j, then a card sorting here will fail due to the same processor. Therefore, we must check if the two cards (two tasks) share the same value (processors). If not, then card sort. If yes, then record it by counter and move the index, *Card1,* to the next and continue the while loop until this card sorting operation is successful.



Non-dominated Solutions

Fig. 5. An example of two trial card sorting in 2nd solution

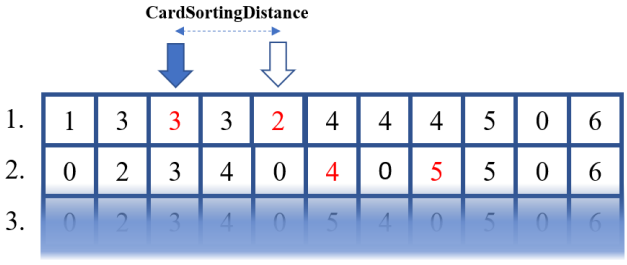


Fig. 6. Updated Solutions

A card sorting example is shown in Fig. 5, 1st solution has successfully done a card sorting at once, while 2nd solution failed in first trial, but moved to next variable and succeeded this time. The results are presented in Fig. 6.

If $r_1 > Cs$, CardForcing operation is conducted, card forcing means forcing a random card to be a certain card that the magician meant to. In fog computing task scheduling problem. The cloud processor is extraordinarily special for its astonishing high processing rate but significantly high operation cost. As a result, we make C_f number of tasks to be processed on the cloud and see if the solution could be better. The illustration of CardForcing technique is shown in Fig. 7.

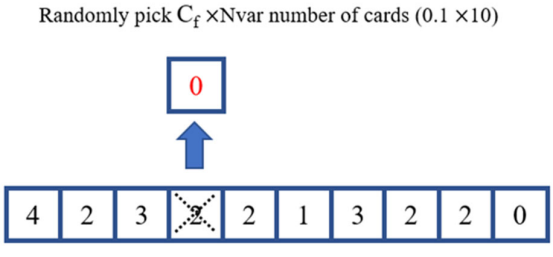


Fig. 7. CardForcing operation

Table 3
Pseudo code of CSLS

Proposed technique: Card Sorting Local Search

Input: Non-dominated Solution set archive $A= \{A_1, A_2, \dots A_N\}$

Parameter initialization: Lgen, Cs, CardSortDistance.

```
for A_i in A do
    counter = 0
    r_l = ρ    ∈ [0,1]
    Card1 = ρ    ∈ [0, "Nvar"]
    if (r_l < Cs) then
        Sorting times= 0
        While (Sorting times <1) do
            Card1 = Card1 %Nvar
            Card2 = (Card1 +CardSortingDistance) %Nvar
            if (Card1≠Card2) then
                Swap Card1 with Card2
            else
                counter+1
                if (counter>3)
                    break while
                else
                    Card1 +1
                end if
            end if
        end while
    else
        Pick Cf×Nvar random cards and force them to process on the cloud
    end if
end for
```

4.3.3 Parameters

Before introducing the approach, the parameters description is listed in Table 4.

Table 4
CSLS parameters description

Parameter	Description
Lgen	Local search times
C_s	Card sorting rate
C_f	Percentage of card forcing variables
CardSortingDistance	Swapping distance of one sorting operation

$$Lgen = \frac{\text{Nvar}}{2} \tag{9}$$

In card sorting operation, the probability of any variable to be chosen was$P(X_j) = \frac{1}{\text{Nvar}}$ (let X_j be the event thatj^{th} variable is chosen to be swapped). However, if a variable has been chosen to beX_j, the variable $X_{j+CardSortingDistance}$ can also be considered the one to be chosen, for swapping is a bilateral operation. Therefore, we set the iteration number, Lgen to be half of *Nvar.* Then we expect there are more than half of the variables to be swapped ($\text{E}(X_j) = P(X_j) \times LGen = \frac{1}{\text{Nvar}} \times \frac{\text{Nvar}}{2} = \frac{1}{2}$).

$$CardSortingDistance = \frac{\text{Nvar}}{Nvm - 1} \tag{10}$$

CardSortingDistance is determined by the average task number received by processors. The idea is to make sure the card sorting operation is meaningful and effective. In this study, every task has been marked by its source of area and the solution is composed in location order. If the CardSortingDistance is too short, the card sorting operation is meaningless. The difference between an inappropriate CardSortingDistance an appropriate one is shown in Fig. 8 and Fig. 9.

In Fig. 9, a meaningless card sorting is operated. Task 1 and 2 originally been processed on the same processor and sent from the same location but swapped again.

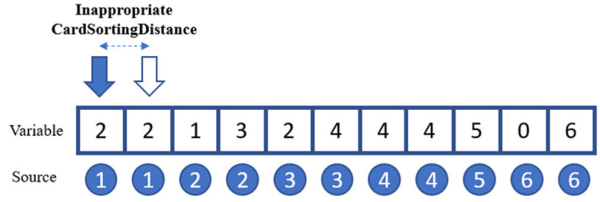


Fig. 8. Inappropriate CardSortingDistance

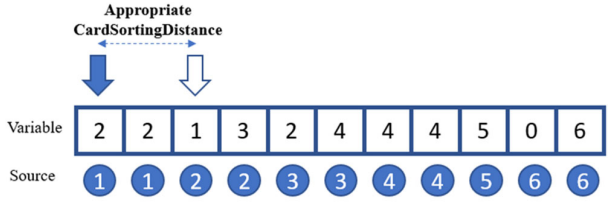


Fig. 9. Appropriate CardSortingDistance

Fig. 10 shows an available sorting operation; T_1 and T_3 were sent from different locations. After sorting, an additional transmission is saved. This example shows valid card sorting that brings the potential of searching for a better solution. As a result, an appropriate CardSortingDistance setting is necessary. We calculate the average number of tasks from each fog device of one scheduling request and divide it by the number of fog devices. The reason is to avoid swapping with the same local processor which makes it meaningless.

4.3.4 An Insight of Card Sorting Local search

In this section, we go deep into the insight of CSLS, and see what it did to a solution. The followings are the benefits gained from the novel mechanisms:

1. The card sorting is between the processors.
   Reason: We card sort until the chosen variables are unequal, then an effective operation is completed.
   Advantage: Reduce redundant and meaningless operations.
2. Card sorting swaps tasks which are sent near each other.
   Reason: It is due to the setting of CardSortingDistance.
   Advantage: It lets the task has a chance to be processed on the processor near the source. Hence, the solution has a chance to be better by striking a better load balance or reduction of transmission cost.

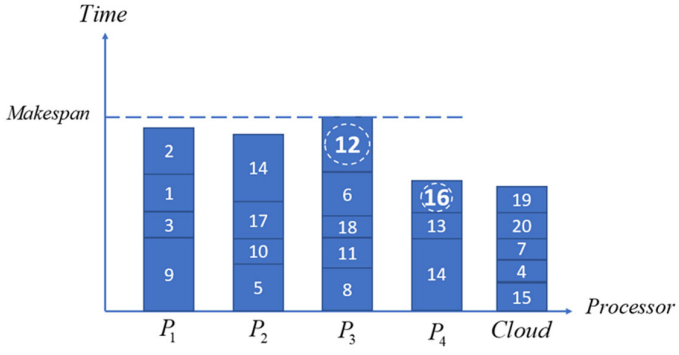


Fig. 10. Solution before card sorting

In Fig. 10, a solution $X = \{1,1,1,0,2,3,0,3,1,2,3,3,4,2,0,4,2,3,0,0\}$ is conducting local search (bold numbers are the chosen number to be swapped). A random ***Card1*** 12 is picked and ***Card2*** 16 which is CardSortingDistance 4 ($\frac{\text{Nvar}}{\text{Nvm}-1} = 4$) far from ***Card1*** is determined, too.

Before card sorting, the computation load was unbalanced and makespan was high. Besides, due to the unbalanced assignment, idle time cost was also high.

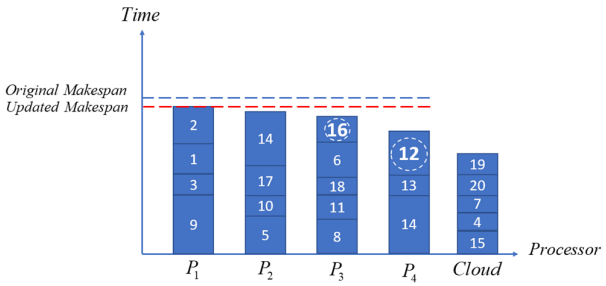


Fig. 11. Makespan after CardSorting

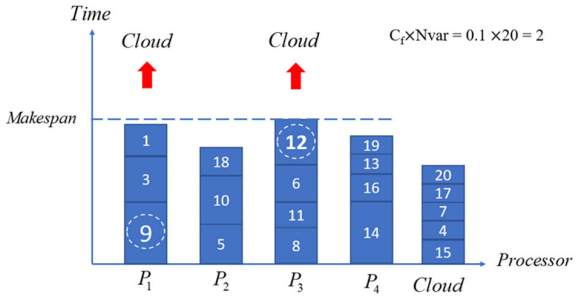


Fig. 12. Solution illustration before CardForcing

After card sorting, T_{12} and T_{16} are swapped. As you can see in Figure 11, Makespan is shortened and P_1 became a new bottleneck of this assignment. On the other hand, the benefit of CardForcing is illustrated in Fig.12. In Fig. 12, 9th task and 12th task was forced to process on the cloud. The updated solution after CardForcing is in Fig. 13.

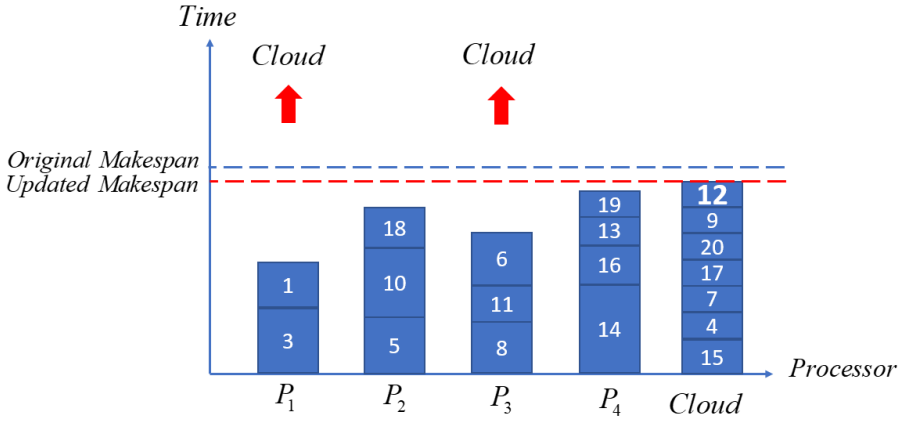


Fig. 13. Solution illustration after CardForcing

As shown in Fig. 13, the makespan was shortened because the computation burden in bottle neck, P_3, was released to the cloud. As a result, a new solution is generated and the makespan has also been shorten again.

4.4. Proposed Strategy

4.4.1 The Overview of Task Scheduling in Fog Computing

In order to present every method that was previously introduced clearly, we visualized the main algorithms combining with the tasks scheduling procedures in Fig. 14.

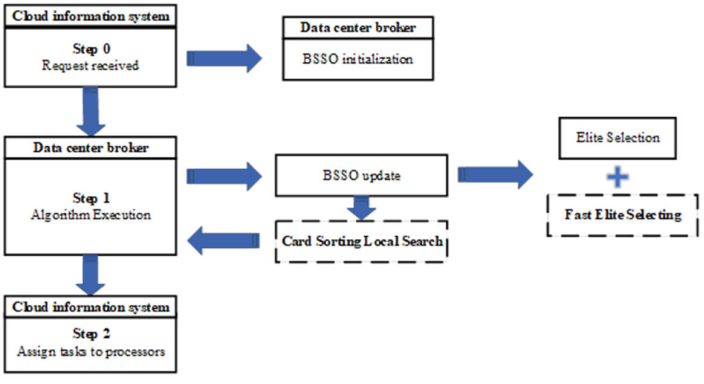


Fig. 14. Task scheduling procedures in this study

Those blocks with dotted line mean the new proposed methods in this study. The description of each step is shown in the following:

Step 0: cloud center sends a signal to the data center broker to schedule an assignment to different processors. The broker received the signal and started BSSO initialization.
Step 1: The broker starts the BSSO update and execute Card Sorting Local Search after the completion of BSSO.
Step 2: According to the results of algorithm, assign tasks to determined processors.

4.4.2 EliteSSO Strategy

In this subsection, we present the whole procedures in pseudo code as follows:

Table 5
EliteSSO strategy

Proposed strategy: EliteSSO strategy
1. BSSO Initialization:
1. X = InitializePopulation() //Randomly generation initial solutions
2. A = FastEliteSelecting(X) //Select solutions of X in front1.
Output: population X, non-dominated solution set archive A

2. BSSO Update
Input: population X, non-dominated solution set archive A
1. **for** *gen* =0 to Ngen **do**
2. *gBest* = selectGbest(A) //Randomly select one *gBest* from A
3. **for** *sol* =0 to Nsol **do**
4.
5.
6. StepwiseUpdate(X_{sol}, *gBest*) //**Elite selection SSO**
7. **end for**
TempX = merge (*X**, *A*) //merge two sets
*A** = FastEliteSelecting (*TempX*)

8. **end for**
Output: non-dominated solution archive A

3.Card Sorting Local Search
Input: non-dominated solution set archive A
1. **while** LGen is not reached **do**
2. *X* = SetArchiveAsSolutions(*A*) //update archive as new X
3. *X** = CardSorting(*X*)
4. *TempX* = merge (*X**, *A*)
5. *A** = FastEliteSelecting (*TempX*)
6. **end while**
Output: non-dominated solution archive A

5. Experiments

In this chapter, we listed the experimental data and scenarios in section 5.1. In section 5.3, we introduce two performance metrics, IGD and spread. An ANOVA test for Card Sorting Local Search parameter, Cs, is conducted in 5.3. Most importantly, the experiment results are presented in 5.4.

5.1. Datasets

We set three different scales of problems, small, medium and large, to evaluate all the algorithms. In addition, for a general and fair comparison, each of them has three different datasets. The contents of each dataset are presented in Table 6 to Table 8. Tasks length and tasks source are attached in **Appendix A**, processing rates and execution cost are attached in **Appendix B**.

Table 6
Dataset 1

Scenarios	Small	Medium	Large
Cloud #	1	1	1
Fog device #	4	7	9
Tasks #	30	50	100
Deadline	80	100	140
P_o	50	100	200
P_t	3	5	10
TCU	1	1	1
Idle cost	5% execution cost	5% execution cost	5% execution cost

- Fixed Violation Cost = instant penalty when the makespan exceed the deadline
- TCU = transmission cost/unit distance
- P_o: fixed violation cost when the makespan of an assignment violate the SLA regulation.
- P_t: variable violation cost

Table 7
Dataset 2

Scenarios	Small	Medium	Large
Cloud #	1	1	1
Fog device #	4	7	9
Tasks #	30	50	100
Deadline	70	90	130
P_o	80	130	230
P_t	4	6	15
TCU	2	2	2
Idle cost	3% execution cost	3% execution cost	3% execution cost

Table 8
Dataset 3

Scenarios	Small	Medium	Large
Cloud #	1	1	1
Fog device #	4	7	9
Tasks #	30	50	100
Deadline	75	85	130
P_o	20	120	225
P_t	2	6	13
TCU	0.5	0.5	0.5
Idle cost	2% execution cost	2% execution cost	2% execution cost

5.2. Performance Metrics

The result of the 50 independent experiments for each of the 8 combinations is shown in Table 11-14. The values in the table are the fitness values obtained by the algorithm In this section, we introduce three performance metrics used in this study. They are Inverted Generation Distance (*IGD)*, Spacing (*Spc)* and Error Rate (ER).

Inverted Generation Distance

IGD is a widely used performance metric for measuring the proximity of convergence and diversity of the discovered Pareto front (Czyżżak & Jaszkiewicz, 1998). The IGD formula is derived as follows formula 11:

$$IGD = \frac{\sqrt{\sum_{v \in P*} d(v,P)^2}}{|P*|} \tag{11}$$

For each solution v in pareto front, we find a solution $P$with minimum Euclidean distance$d(v,P)$ in the non-dominated solution set found by the algorithm and sum them up then divide it by the size of simulated Pareto front P*.

Spacing

Spacing(*Spc*) is used to measure the extent of the non-dominated solutions are distributed along the discovered (Schott, 1995) as shown in formula 12.

$$Spc = \sqrt{\frac{1}{n-1}\sum_{i-1}^{n}\left(d_i - \bar{d}\right)^2} \tag{12}$$

where $d_i = \min_j\left(\left|f_1^i - f_1^j\right| + \left|f_2^i - f_2^j\right|\right), i,j = 1,2,\ldots n$. Where n is the number of discovered non-dominated solutions. If the value of this metric is zero, it indicated that all members of the discovered Pareto front are equidistantly spaced.

Error Rate

Error Rate (*ER*) is to calculate the percentage of true Pareto solutions among discovered temporary non-dominated solutions*P* (Van Veldhuizen, 1999) is shown as formula 13 below:

$$ER = \frac{\sum_{i=1}^{n} e_i}{n} \tag{13}$$

n is the number of solutions in P.e_i is a binary variable. If the solution found in P is the same solution in P*, then the e_i will be 0. Otherwise, if the solution is not the solution in P*, e_i will be 1. Hence, this metric is the lower the better.

5.3. CSLS Parameter Design

There are two customizable parameters in CSLS, C_s and C_f respectively. C_s represents the card sorting rate and C_f indicates the percentage of card forcing variables. However, C_f is strongly interact with C_s. It is tough to determine an appropriate combination for them. Hence, for a simple and convincing parameter design. We conduct a one-way ANOVA test to determine C_s only and fix C_f=0.1 for our experiments.

For each size of problem, we set 3 different levels: Cs =0.75, Cs =0.85 and Cs =0.95. For each level, we execute 40 runs We set three different level of the proposed algorithm with C_s=0.75, C_s=0.85 and C_s=0.95, and compare it with other multi-objective optimization algorithms, i.e., MOPSO and NSGA-II. As we mentioned above, the collected data is better to meet the normality test. We present the normality test in 5.3.1 and provide the ANOVA table in 5.3.2. Interval plots are presented in 5.3.3 to distinguish the difference of each level. Finally, the discussion and conclusion are offered in 5.3.4

5.3.1. Normality Test

In this subsection, normality tests are conducted on each metric. The figures are shown below:

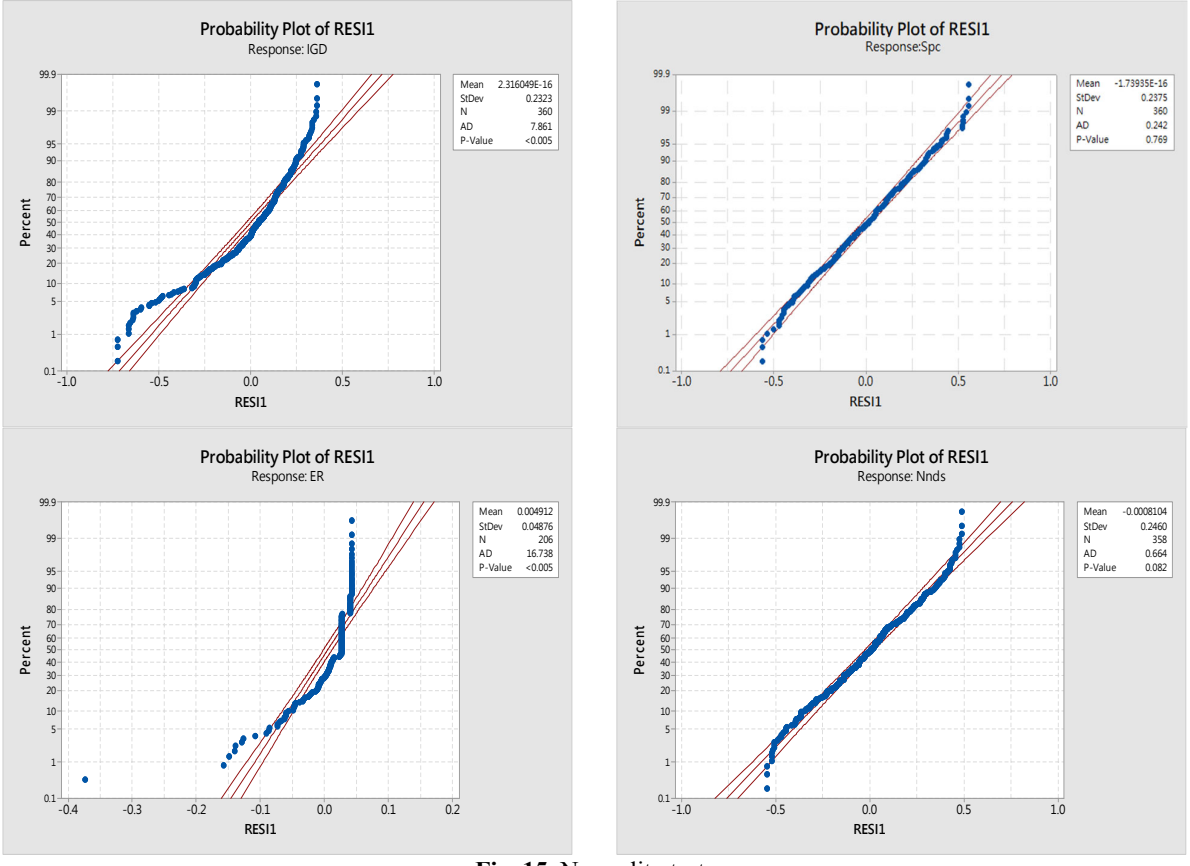


Fig. 15. Normality tests

To summarize, *Spc* and Nnds meets the normality assumption, but IGD and ER do not. That is, we mainly rely on the results of *Spc* and Nnds to design the C_s but also refer to the results of IGD and ER.

5.3.2. ANOVA

For a common parameter for every size of problem, we execute 40 runs for each scale and combine for analysis. To combine the results of a distinct scale of problems, all the metrics are normalized to [0,1] interval by min-max normalization. The ANOVA model information and results are shown below:

Table 9
ANOVA table

Metric	Source	DF	SS	MS	*F*-value	*P*-value
IGD	Factor	2	0.4521	0.22607	4.16	**0.016**
	Error	357	19.3811	0.005429		
	Total	359	19.8333			
	S=0.0263		R-sq=2.68%	R-sq(adj)=1.73%		
Spc	Factor	2	0.9357	0.46786	8.25	**0.000**
	Error	357	20.26510	0.5673		
	Total	359	21.1867			
	S=0.238171		R-sq=4.42%	R-sq(adj)=3.88%		
ER	IGD	2	0.01826	0.009132	2.95	0.054
	Error	357	1.10520	0.003096		
	Total	359				
	S=0.0556400		R-sq=1.63%	R-sq(adj)=1.07%		
Nnds	IGD	2	0.0737	0.03684	0.61	0.546
	Error	357	21.6918			
	Total	359				
	S=0.24698		R-sq=0.34%	R-sq(adj)=0%		

We can infer from Table 9 that different levels on IGD and *Spc* are significantly unequal. However, for ER and Nnds, we have no strong evidence to prove the difference between each level. In the following, we distinguish the difference between each level by interval plots.

From the ANOVA table, interval plots in Fig. 16, we can infer the following.
:

1. Statistically, the *Spc* data acquired by C_s=0.85 is significantly better than the other levels.
2. IGD and ER data do not pass the assumption of normality. However, they all point to the same level, C_s=0.85.
3. Nnds passes the normality assumption, but there exists no significant difference between each level. Nevertheless, it still shows that the level, C_s=0.85, is a little bit better than the other.

In summary, all the results of metrics point to the level, C_s=0.85. That means, we should leave 15% probability for the Card forcing operation which can help the CSLS search non-dominated solutions more effectively.

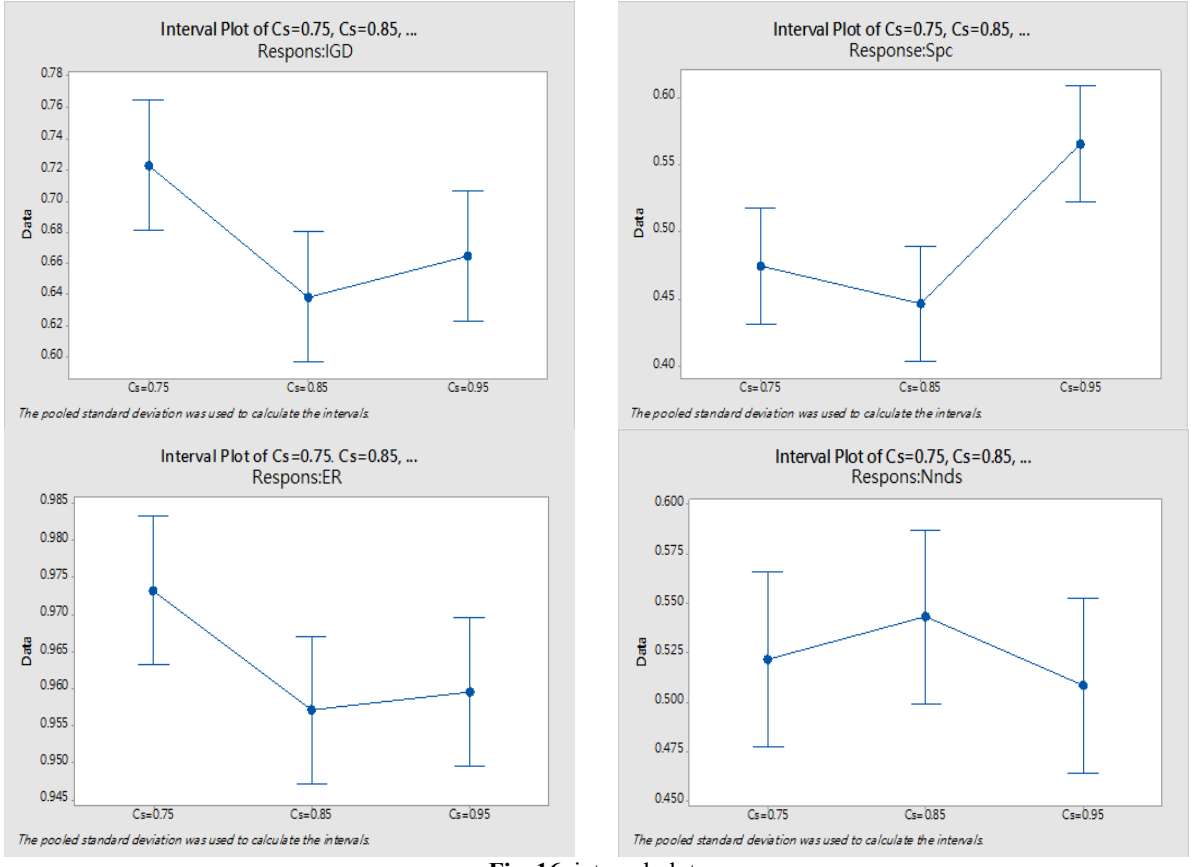


Fig. 16. interval plots

5.4 Experimental Results

In this section, we present the experiment results. We generated 3 datasets for each size of problem. Hence, 9 different experiments are conducted. For each experiment, we execute 40 runs for each algorithm. The parameter of each algorithm and CSLS is shown in Table 10

Table 10
Algorithm parameters

BSSO	
C_g	0.7
C_w	0.9
EliteSSO	
C_g	0.7
C_w	0.9
MOPSO	
w	0.871111
C_1	1.496180
C_2	1.496180
NSGA-II	
Crossover percentage	0.7
Mutation percentage	0.3
Mutation rate	0.05
CSLS	
C_s	0.85
C_f	0.1

For each size of problems, we set different size of particle number and generation numbers. The related information is organized in Table 11.

Table 11
Particle, generation number and archive size

Small	BSSO	MOPSO	NSGA-II	EliteSSO
Nsol	50	50	50	50
Ngen	1000	1000	1000	1000
Archive	50	50	50	50
Medium	BSSO	MOPSO	NSGA-II	EliteSSO
Nsol	100	100	100	100
Ngen	1000	1000	1000	1000
Archive	100	100	100	100
Large	BSSO	MOPSO	NSGA-II	EliteSSO
Nsol	150	150	150	150
Ngen	1000	1000	1000	1000
Archive	150	150	150	150

The results of small, medium, and large are presented in Table 12, Table 13 and Table 14. The final non-dominated solutions of each result are shown after each table. Also, a brief discussion of the results is attached to the end of the figures. All algorithms are coded in Eclipse Java on a 64-bit Windows 10 PC, implemented on an Intel Core i7-7500U CPU @ 2.70 GHz notebook with 12 GB of memory. In addition, we conducted 10 runs on four different threads for every algorithm (i.e., each algorithm was conducted 40 runs.).

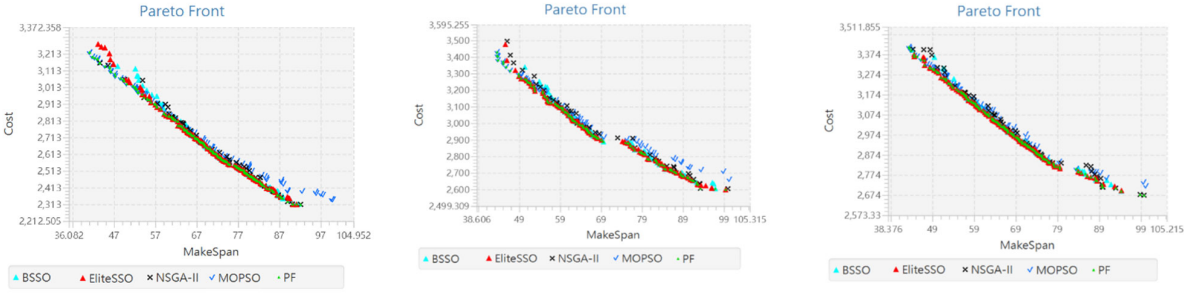


Fig. 17. Small size dataset pareto front

Table 12
Experimental results of small size problem

S1

Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T		
BSSO	0.6249	0.2693	17.0903	**6.9479**	0.8805	0.2644	8.9500	6.2247	7.8046	0.9779		
MOPSO	**0.3710**	0.0238	18.5076	9.3089	0.9869	0.0193	10.0250	3.5531	**5.8975**	**0.3651**		
NSGA-II	0.5892	**0.1797**	23.3736	10.3677	0.9893	0.0494	2.3250	**1.8893**	10.5644	0.3488		
EliteSSO	0.5124	0.2445	**15.9746**	9.5279	**0.8720**	**0.1840**	**20.5000**	9.9800	6.4351	0.7545		

S2

	Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T	
BSSO	0.6899	0.2373	17.7102	**5.9968**	0.9371	0.1655	11.9750	8.2749	7.6923	0.9497		
MOPSO	**0.3703**	**0.0327**	20.4190	11.459	0.9861	**0.0259**	10.7250	3.2786	**5.2376**	0.3226		
NSGA-II	0.5152	0.1511	17.8956	8.2646	0.9942	0.0360	2.9000	**1.9723**	9.6751	**0.2996**		
EliteSSO	0.5909	0.2534	**16.9341**	9.3648	**0.8928**	0.1647	**21.6500**	10.0288	5.7790	0.4812		

S3

Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T		
BSSO	0.6610	0.2529	16.5421	4.7461	0.9191	0.2163	11.4000	6.7483	8.3106	0.8701		
MOPSO	**0.2988**	**0.0236**	19.1278	10.4615	0.9743	0.0313	18.2500	4.6301	7.2008	0.7665		
NSGA-II	0.4077	0.0834	25.7452	9.1027	0.9945	**0.0200**	2.6750	**1.5064**	11.0483	0.5097		
EliteSSO	0.4830	0.2400	**13.6351**	**3.4612**	**0.8674**	0.1316	**24.9000**	11.2978	**6.2987**	**0.4981**		

In small scale problems, MOPSO has a strong competitiveness in *IGD* with EliteSSO. However, EliteSSO has a better ability in searching for the final non-dominated solutions. In addition, EliteSSO has a better diversity for its *Spc* dominated all the other algorithms. Note the interesting part in the third front in Fig. 17, the gap was caused by the deadline. The deadline set in dataset 2 is 70, that is, the assignments on the right-hand side are still worthy even they exceeded the deadline and get penalized. It is because the assignments assign many tasks to fog devices to save the cost instead of executing on the cloud.

Table 13
Experimental results of medium size problem

M1

Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T		
BSSO	0.2951	**0.0270**	14.5437	**4.2349**	0.9983	0.0058	11.150	5.6460	21.898	2.3124		
MOPSO	0.6627	0.0761	26.4642	7.64556	1.0000	**0.0000**	0.1750	0.3800	18.840	**1.0688**		
NSGA-II	0.5848	0.0542	36.132	10.7435	1.0000	**0.0000**	0.1250	**0.3307**	50.649	1.4013		
EliteSSO	**0.2714**	0.0340	**10.2454**	4.1356	**0.9241**	0.0601	**50.800**	5.9841	**18.546**	1.2998		

M2

	Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T	
BSSO	0.3841	0.0591	16.2154	**4.4420**	0.9982	0.0065	9.1250	4.5232	21.318	1.3276		
MOPSO	0.6656	0.0810	27.5213	6.5413	1.0000	**0.0000**	0.1000	0.3000	18.900	1.5734		
NSGA-II	0.6127	0.0834	34.5256	11.2414	1.0000	**0.0000**	0.0750	**0.2634**	49.079	1.6549		
EliteSSO	**0.3347**	**0.0374**	**14.1525**	4.5796	**0.9407**	0.0721	**49.900**	6.6250	**16.470**	**0.7728**		

M3

	Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T	
BSSO	0.291	0.037	15.6693	**4.0185**	0.9944	0.0118	11.000	4.1292	23.578	1.7260		
MOPSO	0.542	0.050	28.0490	8.7335	1.0000	**0.0000**	0.2500	0.6982	19.888	1.2219		
NSGA-II	0.522	0.058	31.9854	12.0512	1.0000	**0.0000**	0.0750	**0.2634**	51.842	3.0614		
EliteSSO	**0.260**	**0.027**	**13.6744**	5.9447	**0.9277**	0.0409	**58.550**	9.5130	**18.357**	**0.8437**		

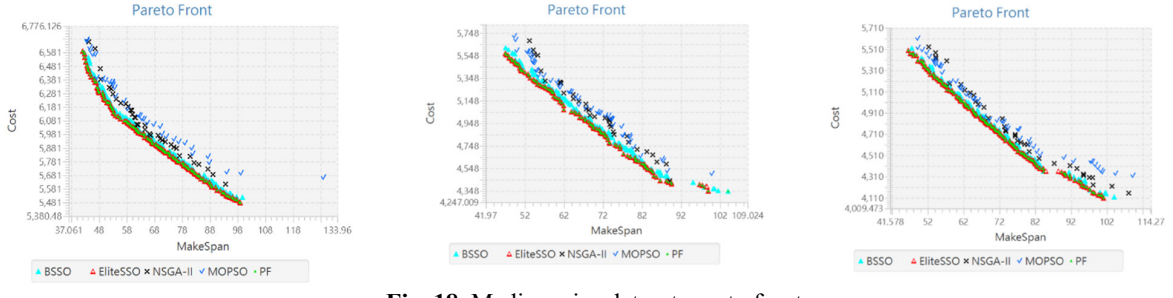


Fig. 18. Medium size dataset pareto front

From the above figures, we can see that nearly all the final non-dominated solutions are found by EliteSSO. Also, EliteSSO has dominated all the other algorithms in *IGD*, *ER*, *Spc*, N_{nds} and *Time*. BSSO has also demonstrated its power in medium-

sized problems. However, due to the Fast Elite selecting technique, EliteSSO is faster than BSSO, even it has applied local search technique, CSLS.

Table 14
Experimental results of large size problem

L1

Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T
BSSO	0.6229	0.0759	15.4398	**4.5254**	1.0000	**0.0000**	2.8000	2.5120	64.1200	3.5634
MOPSO	1.0922	0.1135	76.1219	13.1457	1.0000	**0.0000**	0.1000	0.3000	71.4048	3.4427
NSGA-II	0.9409	0.0798	28.8946	11.8243	1.0000	**0.0000**	0.0250	**0.1561**	174.0804	10.2135
EliteSSO	**0.4300**	**0.0410**	**11.8291**	4.8247	**0.9305**	0.0579	**64.4000**	8.4876	**51.6086**	**1.8958**

L2

Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T
BSSO	0.4238	0.0424	12.64613	5.3751	1.0000	**0.0000**	3.7500	4.1638	60.0497	2.4371
MOPSO	1.0104	0.0713	86.1468	15.1584	1.0000	**0.0000**	0.0250	**0.1561**	81.3329	4.3876
NSGA-II	0.7518	0.0850	28.5113	11.7432	1.0000	**0.0000**	0.1000	0.3742	178.4684	8.0741
EliteSSO	**0.3251**	**0.0314**	**9.4253**	**5.2142**	**0.9439**	0.0654	**77.4750**	9.7082	**51.6956**	**2.3856**

L3

Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\overline{T}$	σ_T
BSSO	0.3852	**0.0307**	14.88989	5.2499	1.0000	**0.0000**	1.7500	2.0218	82.3662	**4.4507**
MOPSO	0.6533	0.0624	40.69087	12.9534	1.0000	**0.0000**	0.1750	0.4409	95.3730	6.8829
NSGA-II	0.6561	0.0706	31.90342	12.9282	1.0000	**0.0000**	0.1500	0.4770	194.3408	8.6384
EliteSSO	**0.3107**	0.0347	**10.8291**	**5.1606**	**0.9536**	0.0604	**82.6250**	**11.2109**	**71.2579**	5.0152

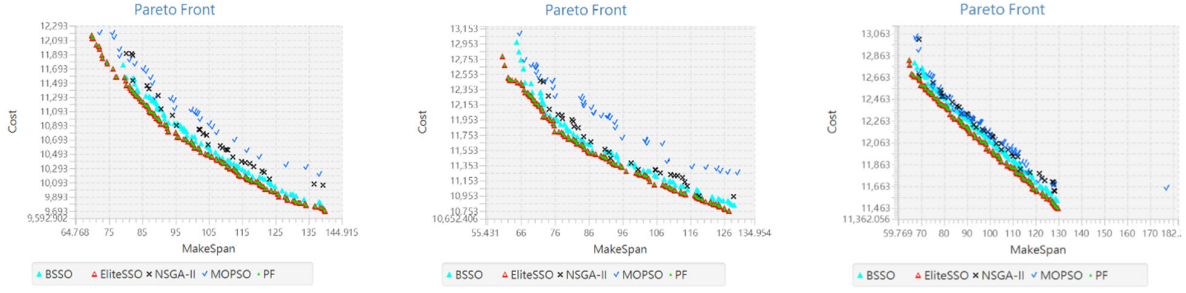


Fig. 19. Large size dataset pareto front

In large scale problems, the difference of each algorithm has been enlarged. EliteSSO acquired more high-quality solutions in shorter simulation time. In summary, the power of EliteSSO is strengthened along with the increase of problem size. In time efficiency, solution quality and solutions diversity, EliteSSO overwhelmed other algorithms in medium and large problems.

5.4. Statistical Verification

To verify the significant difference over the whole multiple comparison, we conducted Friedman's test on the compared algorithms. We set the significance level $\alpha = 0.05$ as a threshold to determine whether to reject hypothesis. *IGD, Spc* and *ER.* The statistical results based on three performance metrics, *IGD*, *Spc* and *ER*, are presented in Table 15 to Table 17:

Table 15
The results of Friedman's test on *IGD*

Friedman's test			
Algo.	$\overline{\text{Rank}}$	Statistic	*p*-value
BSSO	2.6667	6.6	0.086
MOPSO	2.8889		
NSGA-II	2.8889		
EliteSSO	1.5556		

Table 16
The results of Friedman's test on *Spc*

Friedman's test			
Algo.	$\overline{\text{Rank}}$	Statistic	*p*-value
BSSO	2	24.33	**0.000**
MOPSO	1		
NSGA-II	3.4444		
EliteSSO	3.5556		

Table 17
The results of Friedman's test on *ER*

Friedman's test			
Algo.	$\overline{\text{Rank}}$	Statistic	*p*-value
BSSO	2.3333	24.12	**0.000**
MOPSO	3.1667		
NSGA-II	3.5		
EliteSSO	1		

According to the above results of Friedman's tests, there exist significance difference among all algorithms on *Spc* and *ER*. Since the significant difference, we applied Holm's method to further distinguish the differences pairwise between the proposed EliteSSO and other algorithms on *Spc* and *ER*. The results are shown in Table 18 and Table 19.

Table 18
Holm's test on *Spc*

	Holm's test	
Algo.	Statistic	*p*-value
BSSO	1.732	0.083
MOPSO	4.330	**0.000**
NSGA-II	4.330	**0.000**
EliteSSO		

Table 19
Holm's test on *ER*

	Holm's test	
Algo.	Statistic	*p*-value
BSSO	2.191	**0.028**
MOPSO	3.560	**0.000**
NSGA-II	4.108	**0.000**
EliteSSO		

According to the Holm's test on *Spc*, EliteSSO has no significant difference with BSSO. Since EliteSSO searched for the potential non-nominated solutions around the final non-dominated solutions of BSSO. Nevertheless, EliteSSO has significant differences with MOPSO and NSGA-II. For another metric, *ER*, EliteSSO is superior to all the other algorithms.

6. Conclusion

In conclusion, this study has made significant strides towards improving and streamlining BSSO through the introduction of the Fast Elite Selecting (FES) one-front non-dominated sorting technique. This novel method successfully reduced the time complexity of sorting techniques from $O(MN^2)$ to $O(MNnds^2)$, marking a significant advance in computational efficiency. Furthermore, our research has expanded the potential of non-dominated solution exploration by proposing a local search approach named "Card Sorting". This method, specifically designed for task scheduling, has shown promising results in enhancing the exploration ability of non-dominated solutions. Addressing the critical issue of simulation delay in fog computing task scheduling, we have approached it from three distinct aspects: algorithm time complexity, solutions exploration, and parallel computing. In doing so, we have provided a new perspective on this problem and established a pathway for future research in addressing simulation delay issues. Lastly, we have also paved the way for further studies in utilizing multi-objective optimization algorithms for fog computing task scheduling problems. This signifies a new direction for this field, highlighting the potential of multi-objective optimization algorithms in solving such complex problems. Our study, therefore, serves not just as a demonstration of these novel approaches but also as a stepping stone for future research in these fields. We believe that the methods and perspectives introduced in this paper will significantly contribute to the ongoing efforts to optimize fog computing task scheduling and similar complex multi-objective optimization problems.

Acknowledgment

The authors wish to thank the anonymous editor and the referees for their constructive comments and recommendations, which significantly improved this article. This research was supported in part by the Ministry of Science and Technology, R.O.C. under grant MOST 110-2221-E-007-107-MY3.

References

Binh, H. T. T., Anh, T. T., Son, D. B., Duc, P. A., & Nguyen, B. M. (2018, December). An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment. In *Proceedings of the 9th International Symposium on Information and Communication Technology* (pp. 397-404).

Bitam, S., Zeadally, S., & Mellouk, A. (2018). Fog computing job scheduling optimization based on bees swarm. *Enterprise Information Systems*, *12*(4), 373-397.

Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012, August). Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (pp. 13-16).

Chiang, M., & Zhang, T. (2016). Fog and IoT: An overview of research opportunities. *IEEE Internet of things journal*, *3*(6), 854-864.

Coello, C. A. C., Pulido, G. T., & Lechuga, M. S. (2004). Handling multiple objectives with particle swarm optimization. *IEEE Transactions on evolutionary computation*, *8*(3), 256-279.

Czyżżak, P., & Jaszkiewicz, A. (1998). Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of multi-criteria decision analysis*, *7*(1), 34-47.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, *6*(2), 182-197.

Deng, R., Lu, R., Lai, C., Luan, T. H., & Liang, H. (2016). Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE internet of things journal*, *3*(6), 1171-1181.

Doğan, A., & Özgüner, F. (2005). Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems. *The Computer Journal*, *48*(3), 300-314.

Fard, H. M., Prodan, R., Barrionuevo, J. J. D., & Fahringer, T. (2012). A multi-objective approach for workflow scheduling in heterogeneous environments. Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on,

Fieldsend, J. E., & Singh, S. (2002). A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence.

Han, K. K., Xie, Z. P., & Lv, X. (2018). Fog computing task scheduling strategy based on improved genetic algorithm. *Computer Science*, *4*, 22.

He, J., Cheng, P., Shi, L., Chen, J., & Sun, Y. (2013). Time synchronization in WSNs: A maximum-value-based consensus approach. *IEEE Transactions on Automatic Control*, *59*(3), 660-675.

Huang, C. L., & Yeh, W. C. (2019). A new SSO-based algorithm for the bi-objective time-constrained task scheduling problem in cloud computing services. *arXiv preprint arXiv:1905.04855*.

Jena, R. K. (2015). Multi objective task scheduling in cloud environment using nested PSO framework. *Procedia Computer Science*, *57*, 1219-1227.

Jiang, Y., Liu, Z., Chen, J.-H., Yeh, W.-C., & Huang, C.-L. (2023). A novel binary-addition simplified swarm optimization for generalized reliability redundancy allocation problem. *Journal of Computational Design and Engineering*, *10*(2), 758-772.

Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval research logistics quarterly*, *2*(1-2), 83-97.

Li, D., & Sun, X. (2006). *Nonlinear integer programming* (Vol. 84). Springer Science & Business Media.

Li, X. (2003). A non-dominated sorting particle swarm optimizer for multiobjective optimization. *Genetic and Evolutionary Computation Conference.*

Liu, J., Luo, X. G., Zhang, X. M., Zhang, F., & Li, B. N. (2013). Job scheduling model for cloud computing based on multi-objective genetic algorithm. *International Journal of Computer Science Issues (IJCSI)*, *10*(1), 134.

Matt, C. J. B., & Engineering, I. S. (2018). *Fog Computing*. 1-5.

Perera, C., Qin, Y., Estrella, J. C., Reiff-Marganiec, S., & Vasilakos, A. V. (2017). Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys (CSUR)*, *50*(3), 1-43.

Sarkar, S., & Misra, S. (2016). Theoretical modelling of fog computing: a green computing paradigm to support IoT applications. *Iet Networks*, *5*(2), 23-29.

Schott, J. R. (1995). *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization*.

Su, P.-C., Tan, S.-Y., Liu, Z., & Yeh, W.-C. (2022). A Mixed-Heuristic Quantum-Inspired Simplified Swarm Optimization Algorithm for scheduling of real-time tasks in the multiprocessor system. *Applied Soft Computing*, *131*, 109807.

Tasiopoulos, A., Ascigil, O., Psaras, I., Toumpis, S., & Pavlou, G. J. I. T. o. S. C. (2019). FogSpot: Spot Pricing for Application Provisioning in Edge/Fog Computing.

Van Veldhuizen, D. A. (1999). *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*.

Vaquero, L. M., & Rodero-Merino, L. (2014). Finding your way in the fog: Towards a comprehensive definition of fog computing. *ACM SIGCOMM computer communication Review*, *44*(5), 27-32.

Yannuzzi, M., Milito, R., Serral-Gracià, R., Montero, D., & Nemirovsky, M. (2014). Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing. *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD).*

Yeh, W.-C. (2019). A novel boundary swarm optimization method for reliability redundancy allocation problems. *Reliability Engineering & System Safety*, *192*, 106060.

Yeh, W.-C. (2021). One-batch Preempt Deterioration-effect Multi-state Multi-rework Network Reliability Problem and Algorithms. *arXiv preprint arXiv:2103.04325*.

Yeh, W.-C., Lin, Y.-P., Liang, Y.-C., Lai, C.-M., & Huang, C.-L. (2023). Simplified swarm optimization for hyperparameters of convolutional neural networks. *Computers & Industrial Engineering*, *177*, 109076.

Yeh, W.-C., Liu, Z., Yang, Y.-C., & Tan, S.-Y. (2022). Solving dual-channel supply chain pricing strategy problem with multi-level programming based on improved simplified swarm optimization. *Technologies*, *10*(3), 73.

Yeh, W.-C., Su, Y.-Z., Gao, X.-Z., Hu, C.-F., Wang, J., & Huang, C.-L. (2021). Simplified swarm optimization for bi-objection active reliability redundancy allocation problems. *Applied Soft Computing*, *106*, 107321.

Yeh, W.-C., & Tan, S.-Y. (2021). Simplified swarm optimization for the heterogeneous fleet vehicle routing problem with time-varying continuous speed function. *Electronics*, *10*(15), 1775.

Yeh, W.-C., Zhu, W., Yin, Y., & Huang, C.-L. (2023). Cloud Computing Considering Both Energy and Time Solved by Two-Objective Simplified Swarm Optimization. *Applied Sciences*, *13*(4), 2077.

Yeh, W.-C. J. E. S. w. A. (2009). A two-stage discrete particle swarm optimization for the problem of multiple multi-level redundancy allocation in series systems. *36*(5), 9192-9200.

Yeh, W. C. (2012). Novel swarm optimization for mining classification rules on thyroid gland data. *Information Sciences*, *197*, 65-76.

Yeh, W. C. (2017). A new exact solution algorithm for a novel generalized redundancy allocation problem. *Information Sciences*, *408*, 182-197.

Yeh, W. C. (2011). Optimization of the disassembly sequencing problem on the basis of self-adaptive simplified swarm optimization. *IEEE transactions on systems, man, and cybernetics-part A: systems and humans*, *42*(1), 250-261.

Yeh, W. C. (2014). Orthogonal simplified swarm optimization for the series–parallel redundancy allocation problem with a mix of components. *Knowledge-Based Systems*, *64*, 1-12.

Yin, Y. (2018). *Multi-objective Task Scheduling in Cloud Environment Using Multi-objective Simplified Swarm Optimization*. National Tsin Hua University. https://hdl.handle.net/11296/54qc4f

Zhou, A., Qu, B. Y., Li, H., Zhao, S. Z., Suganthan, P. N., & Zhang, Q. (2011). Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and evolutionary computation*, *1*(1), 32-49.

Appendix A

Tasks length and Task source

A. 1. Length and source of each task in S1.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	9719	*1*	**11**	4879	*2*	**21**	12960	*2*
2	10564	*1*	**12**	11602	*2*	**22**	13593	*3*
3	13090	*1*	**13**	9629	*2*	**23**	11834	*3*
4	16253	*1*	**14**	9278	*2*	**24**	7548	*3*
5	8640	*1*	**15**	9434	*2*	**25**	11856	*4*
6	12990	*1*	**16**	12224	*2*	**26**	12102	*4*
7	6723	*1*	**17**	9575	*2*	**27**	7594	*4*
8	7883	*1*	**18**	7207	*2*	**28**	6695	*4*
9	14306	*1*	**19**	10140	*2*	**29**	13738	*4*
10	8104	*2*	**20**	11194	*2*	**30**	10050	*4*

A. 2. Length and source of each task in S2.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	17744	*1*	**11**	9823	*2*	**21**	8687	*3*
2	10680	*1*	**12**	12383	*2*	**22**	13439	*3*
3	8018	*1*	**13**	10480	*2*	**23**	7539	*3*
4	13058	*1*	**14**	8921	*2*	**24**	11151	*3*
5	9873	*1*	**15**	9721	*2*	**25**	11292	*4*
6	12849	*1*	**16**	11835	*2*	**26**	6373	*4*
7	6251	*1*	**17**	11923	*2*	**27**	12387	*4*
8	9825	*1*	**18**	8374	*3*	**28**	7647	*4*
9	8461	*2*	**19**	4443	*3*	**29**	7128	*4*
10	7300	*2*	**20**	13042	*3*	**30**	7258	*4*

A. 3. Length and source of each task in S3.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	10402	*1*	**11**	9282	*3*	**21**	9356	*4*
2	13166	*1*	**12**	12975	*3*	**22**	11778	*4*
3	7334	*2*	**13**	7071	*3*	**23**	7364	*4*
4	13631	*2*	**14**	10078	*3*	**24**	9556	*4*
5	11098	*2*	**15**	9513	*3*	**25**	9168	*4*
6	8750	*2*	**16**	14252	*3*	**26**	9943	*4*
7	11961	*2*	**17**	10596	*3*	**27**	15449	*4*
8	11115	*2*	**18**	12046	*3*	**28**	8245	*4*
9	10054	*2*	**19**	14209	*3*	**29**	9969	*4*
10	10800	*2*	**20**	16246	*3*	**30**	9445	*4*

A. 4. Length and source of each task in M1.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	12306	*1*	**18**	13099	*2*	**35**	10967	*5*
2	10344	*1*	**19**	6014	*2*	**36**	5362	*5*
3	13154	*1*	**20**	6812	*3*	**37**	7695	*5*
4	11789	*1*	**21**	11654	*3*	**38**	13803	*5*
5	13537	*1*	**22**	6985	*3*	**39**	10868	*5*
6	8117	*1*	**23**	13273	*3*	**40**	9315	*6*
7	10504	*1*	**24**	5833	*3*	**41**	9178	*6*
8	15241	*1*	**25**	11476	*3*	**42**	9577	*6*
9	9430	*1*	**26**	8333	*3*	**43**	4187	*6*
10	12567	*1*	**27**	8863	*4*	**44**	9904	*6*
11	11410	*2*	**28**	10473	*4*	**45**	10506	*7*
12	9453	*2*	**29**	12786	*4*	**46**	9301	*7*
13	17191	*2*	**30**	9748	*4*	**47**	14309	*7*
14	16110	*2*	**31**	12044	*4*	**48**	12148	*7*
15	10733	*2*	**32**	6285	*4*	**49**	6821	*7*
16	10120	*2*	**33**	8407	*5*	**50**	11950	*7*
17	14119	*2*	**34**	15797	*5*			

A. 5. Length and source of each task in M2.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	7675	*1*	**18**	5324	*3*	**35**	10371	*6*
2	7953	*1*	**19**	11958	*3*	**36**	12640	*6*
3	16999	*1*	**20**	14635	*3*	**37**	5726	*6*
4	8678	*1*	**21**	11964	*4*	**38**	10074	*6*
5	9567	*1*	**22**	16906	*4*	**39**	10809	*6*
6	6386	*1*	**23**	5407	*4*	**40**	10848	*6*
7	13254	*1*	**24**	6950	*4*	**41**	6355	*6*
8	11634	*1*	**25**	10660	*4*	**42**	11337	*6*
9	7893	*1*	**26**	6396	*4*	**43**	9041	*6*
10	9425	*2*	**27**	9385	*4*	**44**	13308	*7*
11	13313	*2*	**28**	16265	*4*	**45**	10473	*7*
12	12032	*2*	**29**	9633	*4*	**46**	12786	*7*
13	6254	*2*	**30**	8987	*4*	**47**	9648	*7*
14	8307	*2*	**31**	14892	*5*	**48**	12043	*7*
15	16402	*3*	**32**	12926	*5*	**49**	6385	*7*
16	12152	*3*	**33**	10899	*5*	**50**	8413	*7*
17	6964	*3*	**34**	6034	*5*			

A. 6. Length and source of each task in M3.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	10282	*1*	**18**	7868	*3*	**35**	10046	*6*
2	10936	*1*	**19**	13993	*3*	**36**	11144	*6*
3	11959	*1*	**20**	13553	*3*	**37**	4128	*6*
4	11314	*1*	**21**	9610	*3*	**38**	15240	*6*
5	8511	*1*	**22**	13610	*4*	**39**	5661	*6*
6	11369	*1*	**23**	8992	*4*	**40**	3140	*6*
7	14368	*1*	**24**	9928	*4*	**41**	12874	*6*
8	8512	*2*	**25**	9362	*4*	**42**	7984	*7*
9	8741	*2*	**26**	9811	*4*	**43**	11216	*7*
10	15925	*2*	**27**	11686	*4*	**44**	6674	*7*
11	8413	*2*	**28**	4694	*4*	**45**	5360	*7*
12	7283	*2*	**29**	11383	*5*	**46**	8652	*7*
13	7068	*2*	**30**	7417	*5*	**47**	11450	*7*
14	13185	*2*	**31**	11179	*5*	**48**	6103	*7*
15	10188	*3*	**32**	12307	*5*	**49**	14942	*7*
16	6766	*2*	**33**	11824	*5*	**50**	8758	*6*
17	9646	*3*	**34**	4959	*5*			

A. 7. Length and source of each task in L1.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	5745	*1*	**35**	7649	*4*	**69**	9389	*7*
2	11441	*1*	**36**	9609	*4*	**70**	11283	*7*
3	6921	*1*	**37**	4709	*4*	**71**	4649	*7*
4	5759	*1*	**38**	7669	*4*	**72**	8974	*7*
5	11204	*1*	**39**	10040	*4*	**73**	9310	*7*
6	7005	*1*	**40**	12120	*4*	**74**	13240	*7*
7	7470	*2*	**41**	9587	*4*	**75**	11625	*7*
8	10155	*2*	**42**	4908	*4*	**76**	9567	*7*
9	11157	*2*	**43**	11258	*4*	**77**	9832	*7*
10	12669	*2*	**44**	8920	*4*	**78**	11054	*7*
11	10961	*2*	**45**	8670	*4*	**79**	13825	*8*
12	11123	*2*	**46**	1960	*5*	**80**	10040	*8*
13	10792	*2*	**47**	13552	*5*	**81**	8597	*8*
14	14891	*2*	**48**	8095	*5*	**82**	9239	*8*
15	8519	*2*	**49**	10969	*5*	**83**	11748	*8*
16	7995	*2*	**50**	14628	*5*	**84**	8207	*8*
17	6485	*2*	**51**	12043	*5*	**85**	10906	*8*
18	10578	*2*	**52**	13533	*5*	**86**	9497	*8*
19	8450	*2*	**53**	11221	*5*	**87**	7264	*8*
20	10150	*2*	**54**	13525	*5*	**88**	12620	*9*
21	7136	*3*	**55**	9689	*5*	**89**	10739	*9*
22	8902	*3*	**56**	8739	*5*	**90**	7874	*9*
23	11303	*3*	**57**	11041	*6*	**91**	7013	*9*
24	10551	*3*	**58**	7294	*6*	**92**	9437	*9*
25	10744	*3*	**59**	11642	*6*	**93**	11609	*9*
26	10803	*3*	**60**	14157	*6*	**94**	9322	*9*
27	6413	*3*	**61**	7541	*6*	**95**	8618	*9*
28	10042	*3*	**62**	12381	*6*	**96**	13325	*9*
29	11425	*3*	**63**	12124	*6*	**97**	7102	*9*
30	10306	*3*	**64**	13927	*6*	**98**	15348	*9*
31	9437	*3*	**65**	11463	*6*	**99**	11774	*9*
32	13268	*4*	**66**	10753	*6*	**100**	9038	*9*
33	12902	*4*	**67**	16585	*6*			
34	7123	*4*	**68**	11550	*6*			

A. 8. Length and source of each task in L2.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	7654	*1*	**35**	6132	*3*	**69**	9689	*7*
2	10694	*1*	**36**	9652	*3*	**70**	12826	*7*
3	13849	*1*	**37**	12838	*3*	**71**	11020	*7*
4	9013	*1*	**38**	10567	*4*	**72**	6148	*7*
5	8908	*1*	**39**	12317	*4*	**73**	12784	*7*
6	10148	*1*	**40**	7460	*4*	**74**	14702	*7*
7	11450	*1*	**41**	13996	*4*	**75**	12805	*8*
8	9974	*1*	**42**	16511	*4*	**76**	4320	*8*
9	11330	*1*	**43**	7891	*4*	**77**	10910	*8*
10	14690	*1*	**44**	10741	*4*	**78**	11280	*8*
11	10852	*1*	**45**	9768	*4*	**79**	9136	*8*
12	6544	*1*	**46**	12900	*5*	**80**	7615	*8*
13	12485	*1*	**47**	13888	*5*	**81**	14738	*8*
14	7568	*1*	**48**	14978	*5*	**82**	11173	*8*
15	4784	*1*	**49**	10313	*5*	**83**	5098	*8*
16	15674	*1*	**50**	5634	*5*	**84**	10492	*8*
17	6957	*1*	**51**	12188	*5*	**85**	13538	*8*
18	10079	*2*	**52**	9781	*5*	**86**	3990	*8*
19	12646	*2*	**53**	9445	*5*	**87**	11833	*8*
20	13094	*2*	**54**	14499	*5*	**88**	11235	*8*
21	11002	*2*	**55**	9847	*5*	**89**	18888	*9*
22	9013	*2*	**56**	10130	*5*	**90**	9826	*9*
23	11342	*2*	**57**	8242	*5*	**91**	7776	*9*
24	11497	*2*	**58**	10816	*6*	**92**	7760	*9*
25	14888	*2*	**59**	11621	*6*	**93**	11814	*9*
26	11057	*3*	**60**	11907	*6*	**94**	10125	*9*
27	9967	*3*	**61**	11853	*6*	**95**	9138	*9*
28	9411	*3*	**62**	10661	*6*	**96**	8109	*9*
29	**10451**	**3**	**63**	**14438**	*6*	**97**	8495	*9*
30	8244	*3*	**64**	7254	*6*	**98**	11319	*9*
31	8608	*3*	**65**	9408	*6*	**99**	9045	*9*
32	8985	*3*	**66**	15378	*6*	**100**	13113	*9*
33	9484	*3*	**67**	5012	*7*			
34	7036	*3*	**68**	7280	*7*			

A. 8. Length and source of each task in L3.

No.	Length	Source	No.	Length	Source	No.	Length	Source
1	14789	*1*	**35**	16189	*4*	**69**	10558	*7*
2	3773	*1*	**36**	12561	*4*	**70**	8220	*7*
3	10890	*1*	**37**	11396	*4*	**71**	10206	*7*
4	15952	*1*	**38**	10480	*4*	**72**	9157	*7*
5	10010	*1*	**39**	10583	*4*	**73**	7251	*7*
6	13250	*1*	**40**	15582	*4*	**74**	10828	*7*
7	4914	*1*	**41**	5526	*4*	**75**	8535	*7*
8	9148	*1*	**42**	11805	*4*	**76**	5459	*7*
9	9776	*2*	**43**	6109	*4*	**77**	11681	*7*
10	9839	*2*	**44**	9204	*5*	**78**	5854	*7*
11	7585	*2*	**45**	8136	*5*	**79**	6553	*8*
12	8432	*2*	**46**	2970	*5*	**80**	4791	*8*
13	12658	*2*	**47**	11552	*5*	**81**	9526	*8*
14	18774	*2*	**48**	11100	*5*	**82**	14406	*8*
15	13680	*2*	**49**	12190	*5*	**83**	8653	*8*
16	17091	*2*	**50**	12032	*5*	**84**	9064	*8*
17	6650	*2*	**51**	6617	*5*	**85**	7461	*8*
18	10036	*2*	**52**	10895	*5*	**86**	11424	*8*
19	11882	*3*	**53**	13733	*5*	**87**	6924	*8*
20	11069	*3*	**54**	4726	*5*	**88**	8420	*9*
21	8625	*3*	**55**	4892	*6*	**89**	13682	*9*
22	6348	*3*	**56**	7116	*6*	**90**	8031	*9*
23	8201	*3*	**57**	13532	*6*	**91**	8529	*9*
24	14034	*3*	**58**	9137	*6*	**92**	12104	*9*
25	9647	*3*	**59**	8903	*6*	**93**	6729	*9*
26	12868	*3*	**60**	11369	*6*	**94**	14374	*9*
27	12218	*3*	**61**	13020	*6*	**95**	6898	*9*
28	12515	*3*	**62**	12655	*6*	**96**	11411	*9*
29	10179	*3*	**63**	14565	*6*	**97**	9678	*9*
30	7529	*4*	**64**	5871	*6*	**98**	14625	*9*
31	11270	*4*	**65**	13150	*6*	**99**	10929	*9*
32	10802	*4*	**66**	8269	*6*	**100**	19363	*9*
33	6106	*4*	**67**	10179	*6*			
34	12858	*4*	**68**	9412	*7*			

Appendix B

Processing Rates and Cost

Table B-1
Processing Rates and Cost of SD1

	Problem Scale: Small Data set: 1				
	Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
Processing Rate	4000	2000	1500	1000	500
Cost	*55*	*15*	*12*	*8*	*3*

Table B-2
Processing Rates and Cost of SD2

	Problem Scale: Small Data set: 2				
	Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
Processing Rate	4000	1500	1000	800	500
Cost	*55*	*12*	*8*	*5*	*3*

Table B-3
Processing Rates and Cost of SD3

	Problem Scale: Small Data set: 3				
	Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
Processing Rate	4000	2000	1500	1000	500
Cost	*55*	*15*	*12*	*8*	*3*

Table B-4
Processing Rates and Cost of MD1

Problem Scale: Medium Data set: 1				
Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
4000	2500	2000	1500	1000
55	*32*	*27*	*15*	*8*
Fog5	**Fog6**	**Fog7**		
1000	*500*	*500*		
8	*3*	*3*		

Table B-5
Processing Rates and Cost of MD2

Problem Scale: Medium Data set: 2				
Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
4000	2000	1000	1000	1000
55	*27*	*8*	*8*	*8*
Fog5	**Fog6**	**Fog7**		
800	*800*	*500*		
5	*5*	*3*		

Table B-6
Processing Rates and Cost of MD3

Problem Scale: Medium Data set: 3				
Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
4000	2000	1000	1000	1000
55	*27*	*8*	*8*	*8*
Fog5	**Fog6**	**Fog7**		
800	*800*	*500*		
5	*5*	*3*		

Table B-7
Processing Rates and Cost of LD1

Problem Scale: Large Data set: 1				
Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
5000	2000	2000	1500	1500
65	*27*	*27*	*15*	*15*
Fog5	**Fog6**	**Fog7**	**Fog8**	**Fog9**
1000	*1000*	*800*	*800*	*500*
8	*8*	*5*	*5*	*3*

Table B-8
Processing Rates and Cost of LD2

Problem Scale: Large Data set: 2				
Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
5000	2500	2500	2000	2000
65	*32*	*32*	*27*	*27*
Fog5	**Fog6**	**Fog7**	**Fog8**	**Fog9**
1000	*1000*	*800*	*800*	*500*
8	*8*	*5*	*5*	*3*

Table B-9
Processing Rates and Cost of LD3

Problem Scale: Large Data set: 3				
Cloud	**Fog1**	**Fog2**	**Fog3**	**Fog4**
5000	2500	2000	2000	2000
65	*32*	*27*	*27*	*27*
Fog5	**Fog6**	**Fog7**	**Fog8**	**Fog9**
1000	*1000*	*800*	*500*	*500*
8	*8*	*5*	*3*	*3*