國 立 清 華 大 學

工業工程與工程管理學系

碩 士 論 文

Fog Computing Task Scheduling Optimization Based on Bi-Objective Simplified Swarm Optimization With Card Sorting Local Search and Fast Elite Selecting

應用雙目標簡化粒子群演算法結合理牌區域搜尋法及快速菁英挑選法求解多目標霧計算任務分配問題

系別 工業工程與工程管理學系 組別 工業工程組

學號姓名 106034553 曾冠程 Kuan-Cheng Tseng

指導教授 葉維彰 博士 Dr. Wei-Chang Yeh

中華民國 一百零八年 六 月

國　立　清　華　大　學

博碩士論文電子檔著作權授權書

（提供授權人裝訂於紙本論文書名頁之次頁用）

本授權書所授權之學位論文，為本人於國立清華大學工業工程與工程管理學系 ____________組， 107 學年度第____學期取得碩士學位之論文。

論文題目：應用雙目標簡化粒子群演算法 結合理牌區域搜尋法及快速菁英挑選法 求解多目標霧計算任務分配問題
指導教授：葉維彰

■ 同意

本人茲將本著作，以非專屬、無償授權國立清華大學與台灣聯合大學系統圖書館：基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立清華大學及台灣聯合大學系統圖書館得不限地域、時間與次數，以光碟或數位化等各種方法收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行線上檢索、閱覽、下載或列印。

論文電子全文上載網路公開之範圍及時間：

本校及台灣聯合大學系統區域網路	■ 中華民國 109 年 7 月 19 日公開
校外網際網路	■ 中華民國 109 年 7 月 19 日公開

論文中英文摘要、目次、參考文獻上載網路公開之範圍及時間：

本校及台灣聯合大學系統區域網路	■ 中華民國 109 年 7 月 19 日公開
校外網際網路	■ 中華民國 109 年 7 月 19 日公開

授 權 人：曾冠程

親筆簽名：______________________

中華民國　　　　年　　　　月　　　　日

國 立 清 華 大 學

博碩士紙本論文著作權授權書

（提供授權人裝訂於全文電子檔授權書之次頁用）

本授權書所授權之學位論文，為本人於國立清華大學工業工程與工程管理學系 ______________組， 107 學年度第____學期取得碩士學位之論文。

論文題目：應用雙目標簡化粒子群演算法 結合理牌區域搜尋法及快速菁英挑選法 求解多目標霧計算任務分配問題
指導教授：葉維彰

■ 同意

本人茲將本著作，以非專屬、無償授權國立清華大學，基於推動讀者間「資源共享、互惠合作」之理念，與回饋社會與學術研究之目的，國立清華大學圖書館得以紙本收錄、重製與利用；於著作權法合理使用範圍內，讀者得進行閱覽或重製。

本論文因本人期刊發表，請將論文延至 2020 年 07 月 19 日再公開。

■ 本人確認繳交的紙本論文與系統審核通過的電子檔論文內容相符。

授 權 人：曾冠程

親筆簽名：__________________________

中華民國 年 月 日

國家圖書館
博碩士論文電子檔案上網授權書

（提供授權人裝訂於紙本論文本校授權書之後）

ID:G021060345530

本授權書所授權之論文為授權人在國立清華大學工業工程與工程管理學系 107 學年度第___學期取得碩士學位之論文。

論文題目：應用雙目標簡化粒子群演算法 結合理牌區域搜尋法及快速菁英挑選法 求解多目標霧計算任務分配問題
指導教授：葉維彰

茲同意將授權人擁有著作權之上列論文電子全文（含摘要），非專屬、無償授權國家圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

☐ 讀者基於非營利性質之線上檢索、閱覽、下載或列印上列論文，應依著作權法相關規定辦理。

☐ 論文電子全文上載網路公開之範圍及時間：中華民國 109 年 7 月 19 日公開。

授權人：曾冠程

親筆簽名：______________

民國　　年　　月　　日

中文摘要

隨著網路社會的蓬勃發展，資料的傳輸量日益漸增，網路綿密交織，各式各樣的物聯網裝置相互連結，點連著點形成線、線交線形成面、面與面形成網。如此龐大複雜的網路使的傳統中心化的雲計算架構無法負荷，資料傳遞延遲隨規模成長。因此越來越多的學者開始研究去中心化的網路架構概念——霧計算。

在霧計算的應用場景中，許多應用對於時間延遲的容忍度非常低。而演算法又會因問題規模而使得模擬時間增長，使得任務分配無法在可容忍時間內完成。然而過往的研究中，尚無學者對此提出相關解決方案。因此本研究針對演算法時間複雜度以及演算法求解效率兩方面著手，分別提出快速菁英挑選法(Fast Elite Selecting)以及理牌搜尋演算法 (Card Sorting Local Search)。 本研究提出快速菁英挑選法取代快速非支配排序(Fast Non-dominated Sorting)使得篩選非支配解之時間複雜度從$O(N^2)$降至$O\left({N_F}^2\right)$。另一新提出的區域搜尋法——理牌搜尋演算法根據霧計算任務分配問題特性，加強雙目標簡化群體演算法(Bi-Objective Simplified Swarm Optimization)之區域搜尋能力。結合雙目標簡化群體演算法、快速菁英挑選法、理牌搜尋演算法及 CPU 平行處理的技巧，形成本篇研究針對霧計算任務分配所提出的菁英簡化群體演算法(EliteSSO)策略。

本研究的實驗結果顯示，本研究所提的策略在時間上及求解品質上皆勝過其他演算法如：NSGA-II 以及 MOPSO。其優勢隨著問題的規模及複雜度的增加而逐漸擴大。

關鍵詞：霧計算、任務分配、區域搜尋、簡化粒子群演算法、多目標、非支配排序。

Abstract

With the dramatic growth of data volume, the cloud computing structure has faced a severe difficulty, the latency. In order to deal with this problem, researchers have proposed the fog computing structure, which can successfully release the computation loads from one data center of cloud to multiple local fog devices. Hence, the tasks will be processed at the local fog device and avoid transmitting to data center which is not cost-effective, and the results can be transmitted to the users immediately. In this way, fog computing not only shorten the latency but achieved cost reduction.

In fog computing paradigm, many applications are time sensitive. However, most of the research so far haven't presented a time-efficient and well-customized algorithm especially for fog computing task scheduling problem. Hence, we focused on time-efficiency issue and proposed a novel local search mechanism, Card Sorting Local Search (CSLS), which effectively improved the non-dominated solutions found by Bi-objective Simplified Swarm Optimization (BSSO). Moreover, we also proposed a one-front non-dominated sorting method, Fast Elite Selecting (FES), reduce the time complexity of non-dominated sorting technique. We Combine BSSO, CSLS and FES, and proposed a new algorithm for this study, Elite Swarm Simplified Optimization (EliteSSO), to overcome difficulties in time-efficiency and number of non-dominated solutions especially in large scale problems like fog computing task scheduling problem.

In the end of this work, computational results demonstrated that the proposed algorithm is time-efficiency and significantly outperformed other algorithms.

Key words: Fog computing, Task scheduling, Local search, SSO, Multi-objective, Non-dominated sorting

List of Contents

List of Tables

List of Figures

Chaper 1 Introduction

1.1 Background

1.1.1. The Rise of Fog

With the arrival of the era of the Internet of Things (IoT), a huge amount of data will be generated from smart devices, e.g. mobile phones, automobiles, wearable devices…etc. It is estimated by IDC (International Data Corporation) that there will be 80 billion connected devices in 2025, helping generate 180 trillion gigabytes of new data that year. Cloud computing services, without a doubt, will suffer from this huge volume of data. Such enormous number of data is forcing the responding latency of cloud to be prolonged. Besides, many IoT applications are demanding either real-time or low latency Yannuzzi, Milito [1]. Traditional cloud computing is no longer be able to afford such a huge amount of data from plenty of IoT devices, and respond within an acceptable latency [2]. Hence, fog computing, a new breed of computing structure, has born to decentralize the cloud computing structure and solve the latency problem.

1.1.2. Fog Computing Definition

Fog computing was first well defined in by Flavio Bonomi, Rodolfo Milito, Jiang Zhu and Sateesh Addepalli in 2012. They defined the characteristics of fog computing paradigm and outlined the vision [3] and it was defined as an intermediate computing power between the cloud and users [2], it helps the cloud to shoulder the burden of heavy computation loads from plenty of smart devices. In addition, fog computing is not replacing cloud computing but regarded as a complementary structure to cloud computing [4]. It extends the computational services offered by the cloud computing to the edge of the network [5]. Moreover, fog computing was defined as a scenario which usually consists of lots ubiquitous, heterogeneous, and decentralized devices communicate and potentially cooperate among them without the intervention of third parties [6]. Mung Chiang &Tao Zhang (2016) focused on the challenges and opportunities of fog, and they showed how fog computing solves the problems existing in IoT framework. Furthermore, they pointed out the End-to-End architectural tradeoffs problem which is a critical issue discussed in our study [7].

1.1.3. Architecture of Fog Computing

The architecture of fog computing could be diverse. Many types of it have been proposed in recent studies. In this section, a general architecture is presented in Figure 1-1. It's basically composed of three layers (e.g., cloud layer, fog layer and device layer).

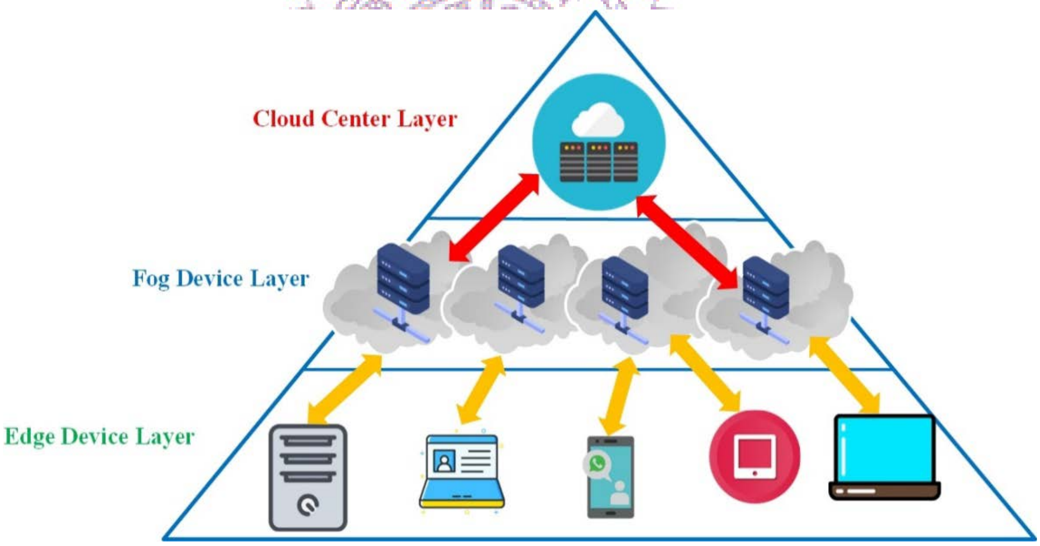

Figure 1-1 Architecture of fog computing

1. Cloud center layer

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 scheduling algorithm.

2. **Fog device layer**

Fog device layer consists of network devices such as router, access points, gateways and switches. They are distributed among edge devices and cloud center, and responsible for collecting raw data from edge devices or analyzed information from the cloud center. In addition, fog servers are able to store sensed data and process for real-time analyses. Moreover, fog servers can preprocess raw data before transmitting to cloud center.

3. **Edge device layer**

Edge devices contain a variety of IoT devices and devices with CPU processors, e.g., laptop, smart vehicles, smart phones. Edge devices are distributed geographically and usually not fixed position. 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.

1.1.4. Difference between Fog and Cloud

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 [3]. 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 number of terminal nodes [8], 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. Last but not least, heterogeneity made the conflict between makespan and cost. That is, a task is arrived, the scheduler will assign it to the cloud for shorter makespan or it will violate Service Level Agreement (SLA) and the service provider will be penalized. However, it will lead the cost to be high instead.

1.2 Motivation

Though fog computing is a simple concept that distributing the computation load to local area, much research hasn't been done yet. For instance, algorithm simulation time is too long for a large distribution system. Most of the research are utilizing linear optimization methods to schedule in 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, number of tasks in fog computing paradigm is usually tremendous, that is, 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 a little cloud and fog device in order 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 [5], GA [9], Evolutionary Algorithm(EA) [10], 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 problem will be worsened when the number of tasks and fog node increases. Though machine learning algorithms require relatively less time than linear optimization methods, the execution time are 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. 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 [11, 20] but also runs fast for it requires only one front to update, such as BSSO [28]. Furthermore, the structure of SSO is flexible, it could be varied to adapt to any kinds of problems. In addition, SSO has been demonstrated its powerful performance in cloud computing task scheduling problem [28]. 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 combine two evaluators as one fitness value. One research set a predetermined balance coefficient α between makespan and cost then optimize the fitness value as close to 1 as possible[10]. Another research set weights for memory and CPU execution time then combine them as a fitness evaluator[5]. Although these strategies have the advantage in shorter simulation time (because of the lower time complexity), the weights are not easy to be determined at first. Moreover, one-fitness strategies produce one solution a time. On the other hand, multi-objective optimization strategies produce a group of solutions that provide the decision maker a variety of choices. However, it requires more time for its higher time complexity. Both two ways of strategy has its benefit and flaw. In this paper, we devote to multi-objective optimization strategy in order to explore the future road of this strategy in this problem.

Last but not least, none of any efficient and effective local search approach has been proposed to search for the potential solutions in fog computing task scheduling problem according to the property of it. In large scale problems, the solution dimension could be complexity which makes the algorithm burdensome to converge. Hence, we are motivated to develop a specialized but without generality local search method.

1.3 Purpose

This research aims at accelerating the simulation time by minimizing time complexity and utilizing parallel computing. On top of that, we devote to developing a local search method to assist the algorithm to converge faster in this problem.

➢ Reducing simulation delay with high-speed strategy

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 $O(MN^2)$ to $O(N_F{}^2)$ (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, effectiveness, we proposed a novel local search method, Card Sorting Local Search, that helps the algorithm search highly potential area 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.

1.4 Structure

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

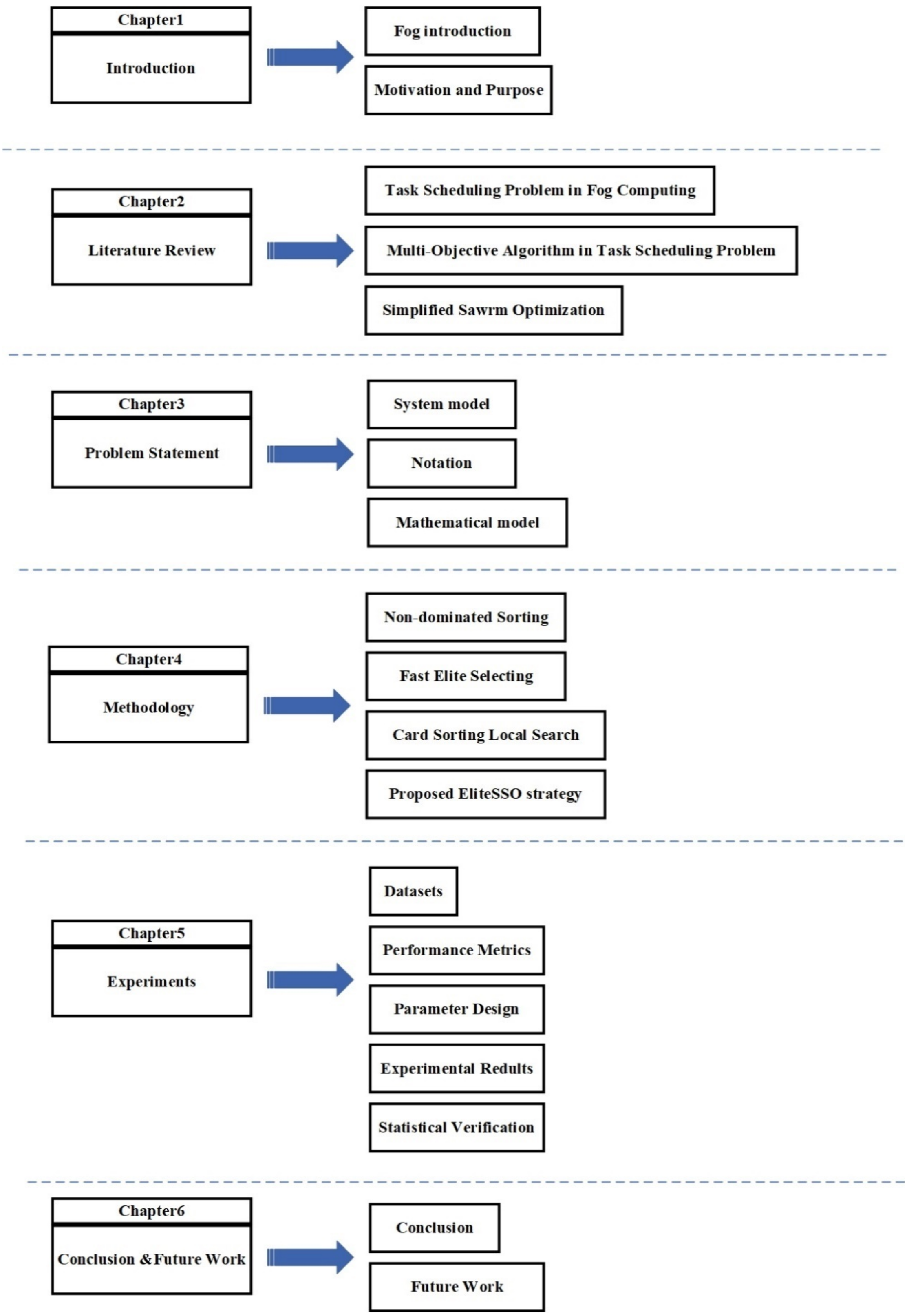

Figure 1-2 Structure

Chaper 2 Literature Review

In this chapter, we review the studies related to task scheduling problem in fog computing in section 2.1. We review the studies related to multi-objective optimization in section 2.2. Then we review the studies of SSO and introduce the update mechanism in section 2.3.

2.1 Task Scheduling Problem in Fog Computing

Deng, Lu, Lai & Luan (2016) considered a tradeoff problem between the power consumption and computation latency, and the author decomposed the problem into three sub-problems [11], the first sub-problem is to find an optimal compromise between the computation latency and power consumption. The author utilized a convex optimization [12]. The second sub-problem is to find an optimal compromise between the power consumption and the computation delay in the cloud computing. A nonlinear integer programming is applied for this sub-problem [13]. The third sub-problem is to minimize communication delay in WAN subsystem, which is considered as an assignment problem. Hence, a Hungarian method is applied [14]. However, this study utilized a centralized approach to performs the optimization process to reduce computation delay and power consumption, which is less appropriate with a fog computing infrastructure, due the fact that central node will easily face a performance bottleneck when allocating workloads, and it will degrade the system performance.

Due to the performance bottleneck of centralized optimization approach, meta heuristic approach is considered. Bitam, Zeadally & Mellouk (2018) proposed a new bio-inspired optimization approach called Bees Life Algorithm (BLA) to address the job scheduling problem in fog computing environment. They decompose jobs into tasks and allocate tasks to different fog devices considering the CPU execution time and

Allocated memory size. During the foraging step, they adopted greed local search process in order to reach the optimal solution among different solutions. Though proving a great performance in solving large scale problem, it did not consider some special characteristics of fog computing paradigm, e.g., the tradeoff problem whether send the tasks to the cloud or not, the communication cost arises from the distance between two distinct fog devices, penalty result from violation of SLA.

HAN Kui-kui, XIE Zai-peng & LV Xin (2018) proposed an improved genetic algorithm for a mixed cloud and fog computing infrastructure. In this study, they considered the cost as a performance evaluation metric which consist of operation cost of virtual machines and penalty result from violating SLA. In this way, the impact of makespan is simultaneously considered when considering the penalty. However, this study didn't consider the tradeoff problem of whether sending the tasks to the cloud or not. That is, this conflict between the makespan and cost is not considered.

2.2 Multi-Objective Algorithm in Task Scheduling Problem

As far as we know, studies of task scheduling in fog computing rarely applied multi-objective algorithm. However, it is of great importance to find an optimal compromise between makespan and cost in a fog computing paradigm. Fieldsend & Singh (2002) proposed Multi-Objective Algorithm (MOA) to solve two conflict metrics problem, and a non-dominated tree is applied to determine the global best for each particle [15]. Colleo, Pulido, Lechuga (2004) proposed multi-objective particle swarm optimization. Unlike other proposals extended PSO to solve multi-objective optimization problems, their algorithm uses an external repository of particles that is later used by other particles to guide their own flight [16]. Zhou, Qu, Zhao, Suganthan and Zhang proposed a Multi-Objective Evolutionary Algorithm (MOEA) to solve the task scheduling problem in grid computing [17]. Liu, Luo, Zhang, Zhang and Li proposed multi-objective genetic algorithm to solve the task scheduling problem in cloud computing [18]. Jena proposed Task Scheduling multi-objective nested Particle Swarm Optimization (TSPSO) for task scheduling with two performance evaluation metrics, power consumption and cost [19]. Fard, Prodan, Barrionuevo and Fahringer proposed Multi-Objective List Scheduling (MOLS) for workflow applications scheduling in heterogeneous systems such as Grids and Clouds [20]. Based on MOLS, Dogan and Ozguner proposed Bi-objective Dynamic Level Scheduling algorithm (BDLS) to maximize the reliability and minimize execution time [21]. Yin (2018) proposed Multi-Objective Simplified Swarm Optimization to solve the conflict cloud computing, with the consideration of makespan and power consumption [22]. Huang proposed Bi-Objective Simplified Swam Optimization (BSSO) in 2019, which removed *gBest* updating mechanism to foster the converge in multi-objective cloud computing task scheduling problem. In our study, we also adopt BOSSO to deal with the task scheduling problem in fog computing.

2.3 Simplified Swarm Optimization

Yeh (2009) proposed Simplified Swarm Optimization, which is a novel population-based stochastic optimization method. It is one of the swarm optimization approach, and its simplicity and efficiency have captured many scholars' attention. It has proved to solve discrete problems efficiently in many studies [22-28].

In simplified swarm optimization algorithm, we set three parameters, C_g, C_p and C_w. where $\mathrm{C}_g > \mathrm{C}_p > C_w$. The update mechanism of SSO is defined by Eq. (2-1):

$$X_{ij}^{t}=\begin{cases} x_{ij}^{t-1} & if \quad \rho\in[0,C_w) \\ p_{ij}^{t-1} & if \quad \rho\in[C_w,C_p) \\ g_j & if \quad \rho\in[C_p,C_g) \\ x & if \quad \rho\in[C_g,1] \end{cases} \tag{2-1}$$

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, ρ 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 paper, we adopted Bi-objective Simplified Swarm Optimization (BSSO) to be the main solution update approach. Bi-objective Simplified Swarm Optimization (BSSO)was recently proposed by Huang [28]. The spirit is from the original Simplified Swarm Optimization (SSO) [27]. The difference between Multi-objective Simplified Swarm Optimization lies in the gBest updating mechanism. In BSSO, it removed pBest and regards each non-dominated solution in external archive as gBest. It strengthened the ability in both the speed of convergence and solution diversity and proved that it outperforms existing famous algorithms, e.g., MOPSO, MOSSO and NSGA-II.

SSO has a simple update mechanism, the stepwise function, and it can be customized to different forms according to different applications [22-28]. There is no exception in multi-objective problems. Being different from single objective problem, multi-objective optimization problems exist not only one gBest solution. Every solution in the non-dominated solution archive is equivalently equal to gBest solution. Hence, in each generation, BSSO randomly pick one non-dominated solution in archive to be the gBest solution. The pseudo code is presented as next page:

Table 2-1 BSSO pseudo code

Proposed technique: Bi-Objective Simplified Swarm Optimization

Initialization:

population X= { $X_1, X_2, \ldots X_N$ }

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

```
for gen= 0 to Ngen do
    Randomly pick a solution A_r to be gBest solution
    for sol=0 to Nsol do
        for var=0 to Nvar do
            r1 = ρ ∈ [0,1]
            If (r1<Cg) then
                X_sol, var = A_r, var
            Else if (r1 <Cw) then
                continue
            Else
                r2= ρ ∈ [0, Nvm ]
                X_sol, var = r2
            end if
        end for
    end for
    Combine updated solutions X* with archive as X_Nsol +NF
    A* = Fast non-dominated sort (X_Nsol +NF)
    If (size of A* >predetermined archive size) then
        A*= crowdingDistanceSelector(A*)
    end if
end for
```

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

Chaper 3 Problem Statement

In this section, we introduce the task scheduling problem in fog computing and the mathematical model in this study. In the following subsections, we introduce the system model in 3.1. In section 3.2, we specify the notations and variables. We introduce the mathematical model in 3.3.

3.1 System Model

A cloud system is composed of the cloud server and multiple fog devices. Each of fog devices is located at different areas. Each fog device receive request 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.

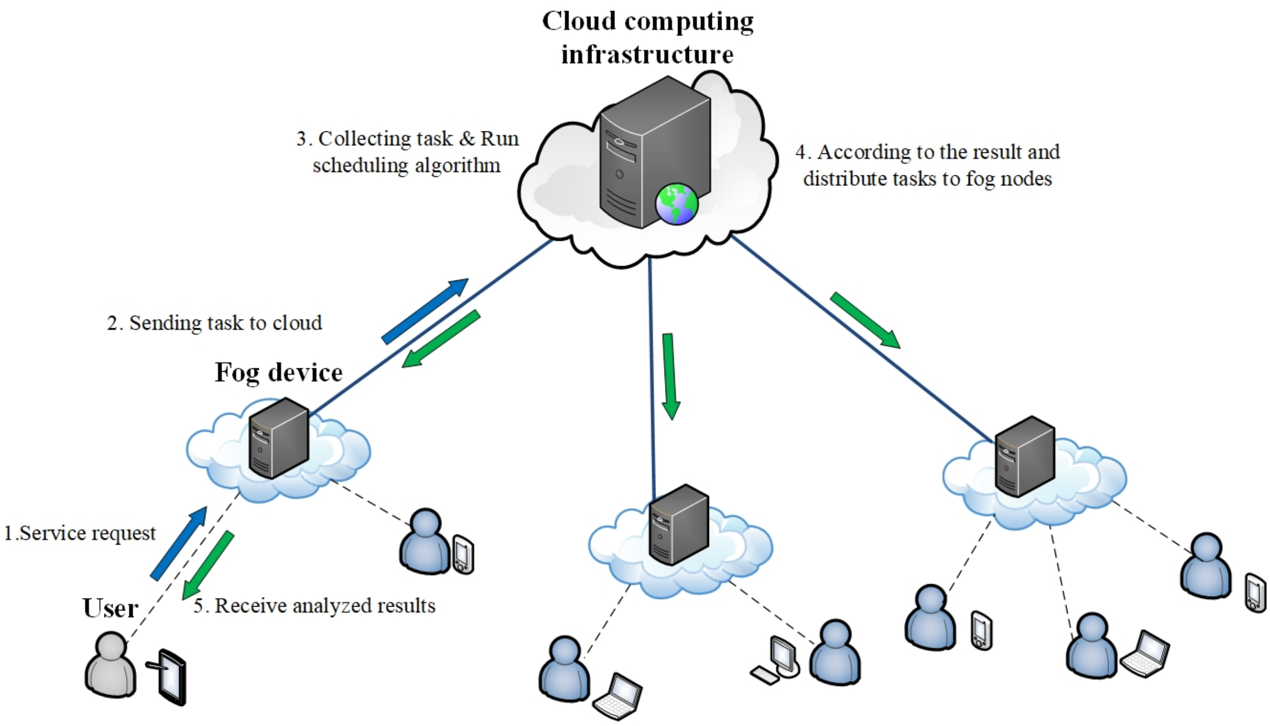

Figure 3-1 System model

In this paper, virtual machine migration is not considered. That is to say, each task can be processed only on one processor, the fog devices are not allowed to halt and transfer task 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 1^{st} task is assigned to processor 3 and 2^{nd} task is assigned to processor 4…etc.

In this study, transmission cost is considered, so we mark each task its source origin like Figure 3-2. Note that the solution in composed in location order.

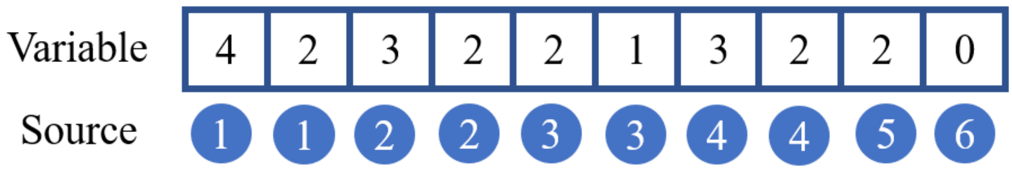

Figure 3-2 Source locating

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

3.2 Notations

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 function.

<u>Notations for mathematical model are listed and introduced as follows:</u>

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 violate 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.

<u>Notations for Card-Sorting BSSO are listed and introduced as follows:</u>

$Ngen$: 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 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:

$$\text{Minmize } Makespan = Execution\ time \tag{3-1}$$

$$\text{Minmize } Cost = Execution\ Cost + Idle\ Time\ Cost + Transmission\ Cost (+Penalty) \tag{3-2}$$

3.3.2. Makespan

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

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

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**

Execution cost is determined by the unit processing cost of processors multiply execution time on each processor.

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

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.

➢ Idle Time Cost

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[29].

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

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

➢ Transmission Cost

When a task is sent away from local area, transmission cost should be considered. Specifically, the cost linearly raised with the increase of distance between two computational devices.

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

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 nodes distance matrix.

➢ Penalty

If the makspan of an assignment exceeds the deadline, penalty cost is generated. The penalty function is shown below:

$$P_o + P_t \times (Makespan - Deadline)$$

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.

Chaper 4 Methodology

In this chapter, we present the methodologies used in this study. Firstly, we introduce 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 being 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 [30].

Non-dominated sorting is to distinguish the dominance relationship from each solution in set $\{X_1, X_2, ...X_N\}$ and store as a descending order set $\{F_1, F_2, ...F_K\}$. A proper non-dominated solution set satisfies the following conditions:

1. All the solutions in a certain front is non-dominated with each other.

$$\forall X_i, X_j \in F_k : X_i \nprec X_j \ and \ X_j \nprec X_i, \ k = 1, 2, ..., k$$

2. Any front with higher index will dominate those with lower index.

$$\forall X \in F_k : \exists X^{'} \in F_{k-1} : X^{'} < X, k = 2, 3, ..., k$$

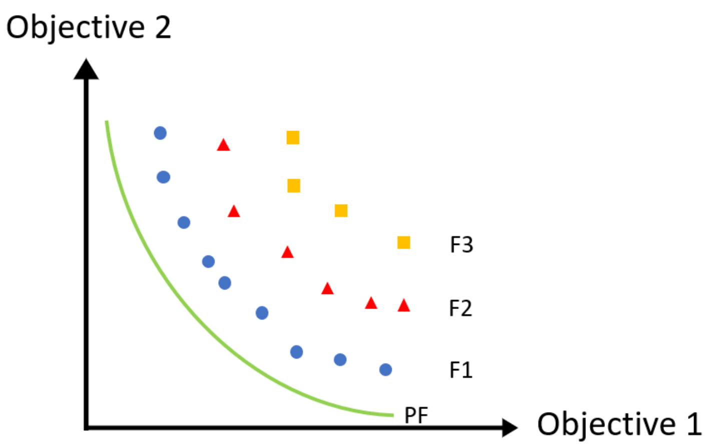

Figure 4-1 Visualization of sorting result

Figure 4-1 shows the dominance relationship of solution set *X*. In this case, we minimize both objective 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 [31]. 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) for it has to 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 4-1.

Table 4-1 Fast Elite Selecting procedure

Proposed technique: Fast Elite Selecting

Input: Solution set X= { $X_1, X_2, \ldots X_N$ }

Initialization: DefaultFront1 = X //Assume all solutions are in front 1.

for each $X_i \in$ DefaultFront1 **do**
    for each $X_j \in$ DefaultFront1 **do**
        if (X_i dominate X_j) **then**
            remove X_j from DefaultFront1
        else if (X_j dominate X_i) **then**
            remove X_i from DefaultFront1 and **break**
        end if
    end
end

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

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** for the reason that 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 that being dominated by nobody. The advantage of FES is its time complexity. Generally speaking, 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 exactly the same time complexity of fast non-dominated sort.

It is because that 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

In this paper, we proposed a novel local search approach, Card Sorting Local Search (CSLS), which is simple and powerful in task scheduling problem. In 4.4.1, we portray the idea of CSLS as a brief story to associate with concept of how CSLS finding better solutions. Then, we introduce the parameters in 4.4.2, and pseudo code in 4.4.3. Last, we presented an example in 4.4.4 to show the insight of CSLS.

4.3.1. The Idea of CSLS

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 checking 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 full house, four of a kind bomb or even straight flush (dominating solutions).

Explanation:

This story is explicitly illustrated the idea of CSLS. Imaging that the cards each player gets is a non-dominated solution. Some cards will find 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 4-3.

4.3.2. The CSLS Operation Explanation

CSLS can starts 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 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 discuss it in detail in next section. In this section, we discuss the detail of this approach and append the pseudo code in next page Table 4-2.

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 be assigned to P_j, then a card sorting here will fail due to the same processor. Therefore, we have to check if the two cards (two tasks) share the same value (processors). If no, 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 success.

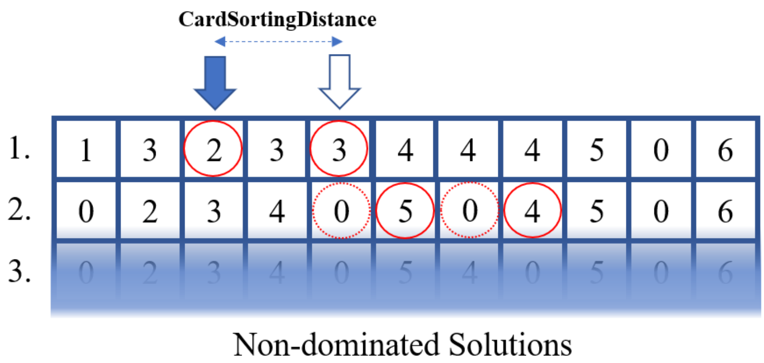

Figure 4-2 An example of two trial card sorting in 2nd solution

A card sorting example is shown in Figure 4-3, 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 Figure 4-3.

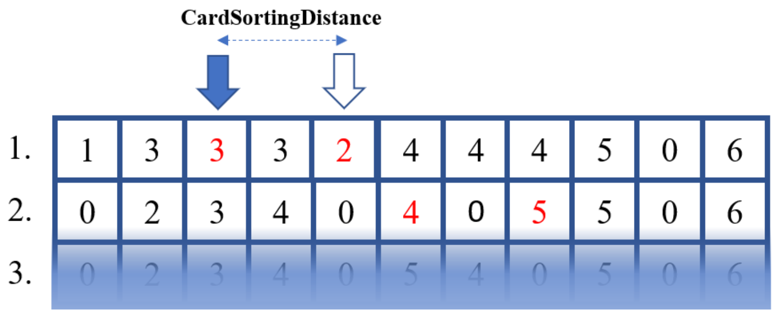

Figure 4-3 Updated Solutions

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 Figure 4-4.

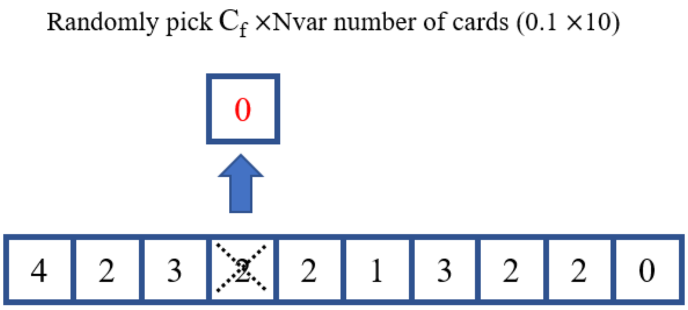

Figure 4-4 CardForcing operation

Table 4-2 Pseudo code of CSLS

Proposed technique: Card Sorting Local Search

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

Parameter initialization: Lgen, Cs, CardSortDistance.

```
for Ai in A do
    counter = 0
    r1 = ρ ∈ [0,1]
    Card1 = ρ ∈ [0, Nvar]
    if (r1 < 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
```

Output: New non-dominated solution set archive A^*

4.3.3. Parameter Introduction

Before introducing the approach, the parameters description is listed in Table 4-1.

Table 4-3 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

- $\text{Lgen} = \dfrac{\text{Nvar}}{2}$ (4-4-1)

In card sorting operation, the probability of any variable to be chosen was $P(X_j) = \dfrac{1}{\text{Nvar}}$ (let X_j be the event that j^{th} variable is chosen to be swapped). However, if a variable has been chosen to be X_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 ($E(X_j) = P(X_j) \times \text{LGen} = \dfrac{1}{\text{Nvar}} \times \dfrac{\text{Nvar}}{2} = \dfrac{1}{2}$).

- $\text{CardSortingDistance} = \dfrac{\text{Nvar}}{\text{Nvm-1}}$ (4-4-2)

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 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 Figure 4-5 and Figure 4-6.

In Figure 4-6, 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.

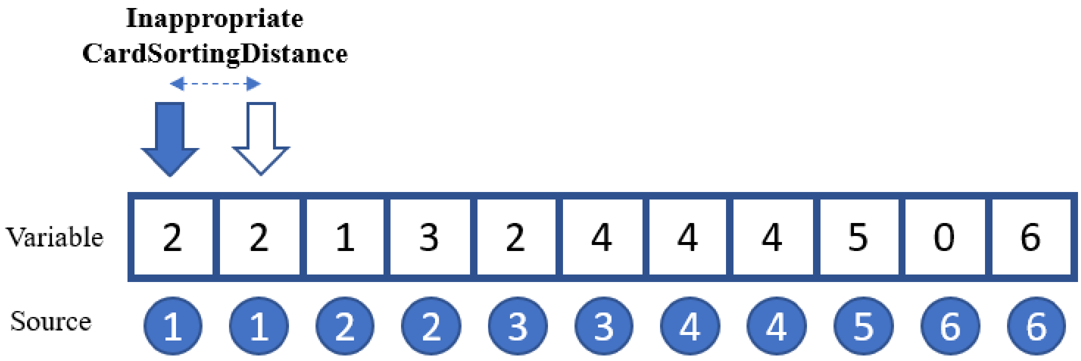

Figure 4-5 Inappropriate CardSortingDistance

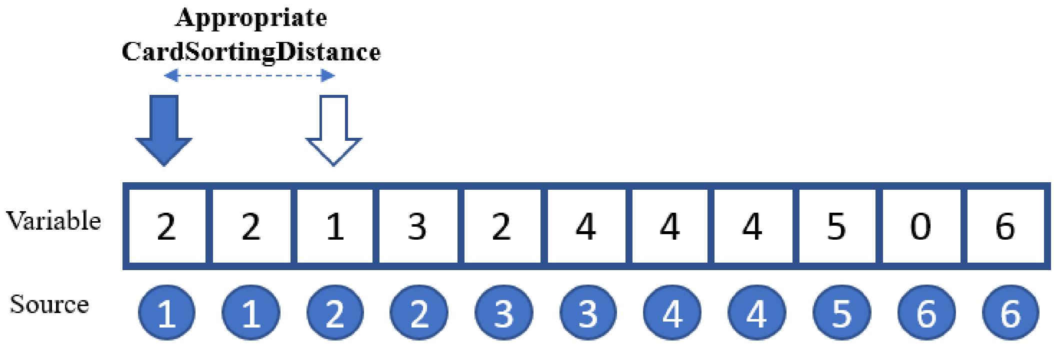

Figure 4-6 Appropriate CardSortingDistance

Figure 4-7 shows an available sorting operation, T_1 and T_3 was sent from different locations. After sorting, an additional transmission is saved. This example shows a valid card sorting that brings the potential of searching a better solution.

As a result, an appropriate CardSortingDistance setting is necessary. We calculate the average number of tasks from each fog devices 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 actually 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.

Benefit: 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.

Benefit: It let 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.

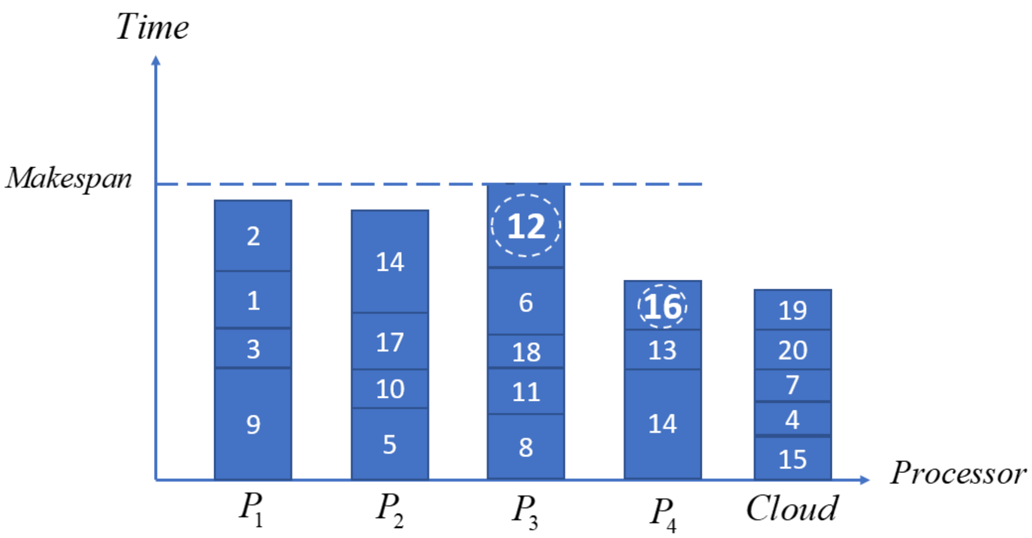

Figure 4-7 Solution before card sorting

In Figure 4-7, 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 is unbalanced and makespan was high. Besides, due to the unbalanced assignment, idle time cost was also high.

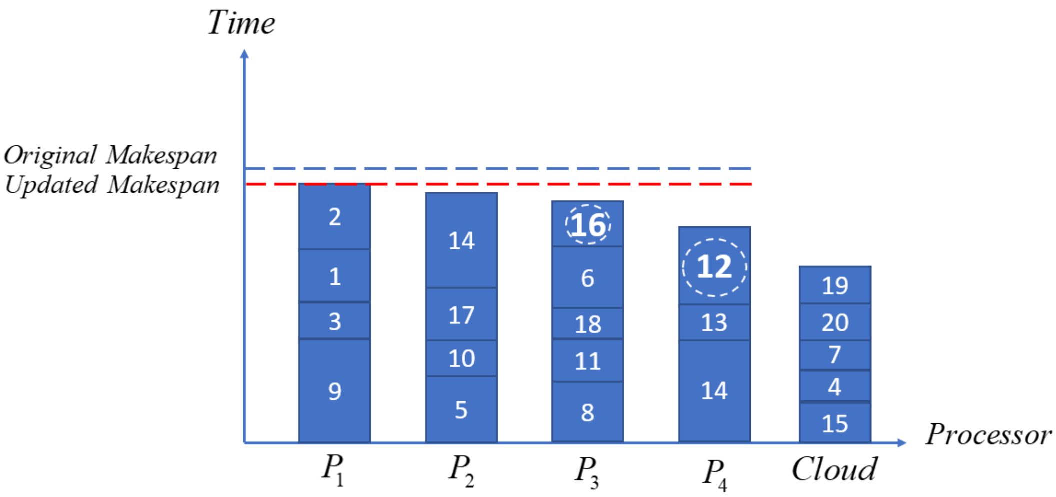

Figure 4-8 Makespan after CardSorting

After card sorting, T_{12} and T_{16} are swapped. As you can see in Figure 4-8, Makespan is shortened and P_1 became a new bottleneck of this assignment.

On the other hand, the benefit of CardForcing is illustrated in the following figures:

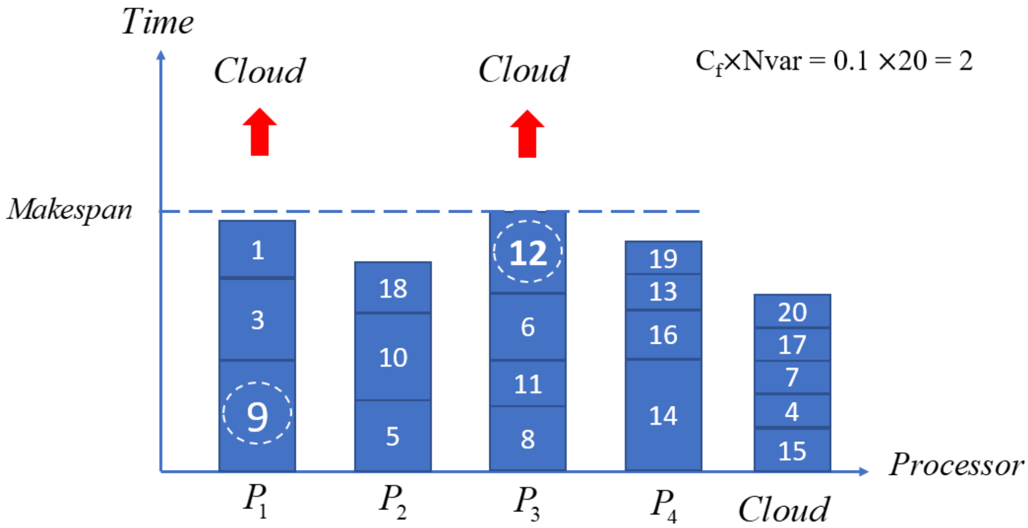

Figure 4-9 Solution illustration before CardForcing

In Figure 4-9, 9th task and 12th task was forced to process on the cloud. The updated solution after CardForcing is in Figure 4-10.

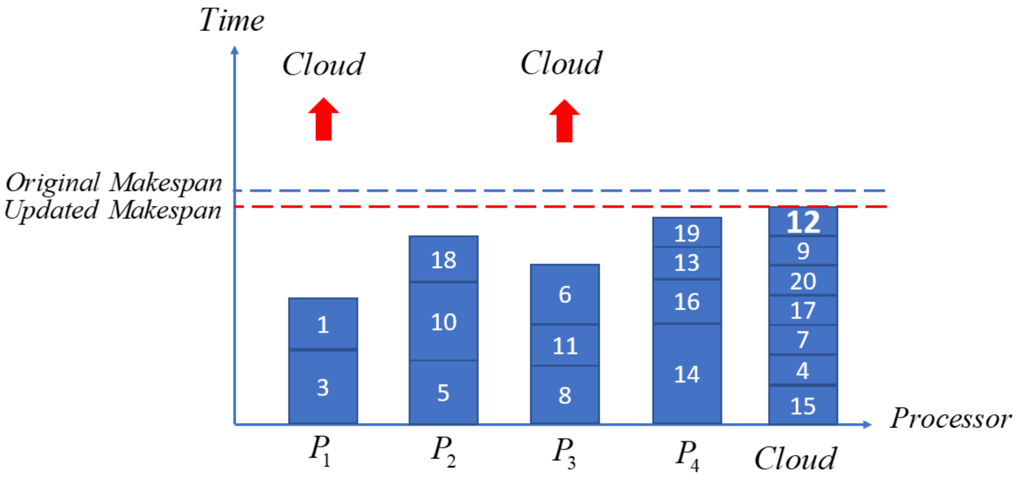

Figure 4-10 Solution illustration after CardForcing

As shown in Figure 4-10, 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 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 previously introduced clearly, we visualized the main algorithms combining with the tasks scheduling procedures in Figure 4-11.

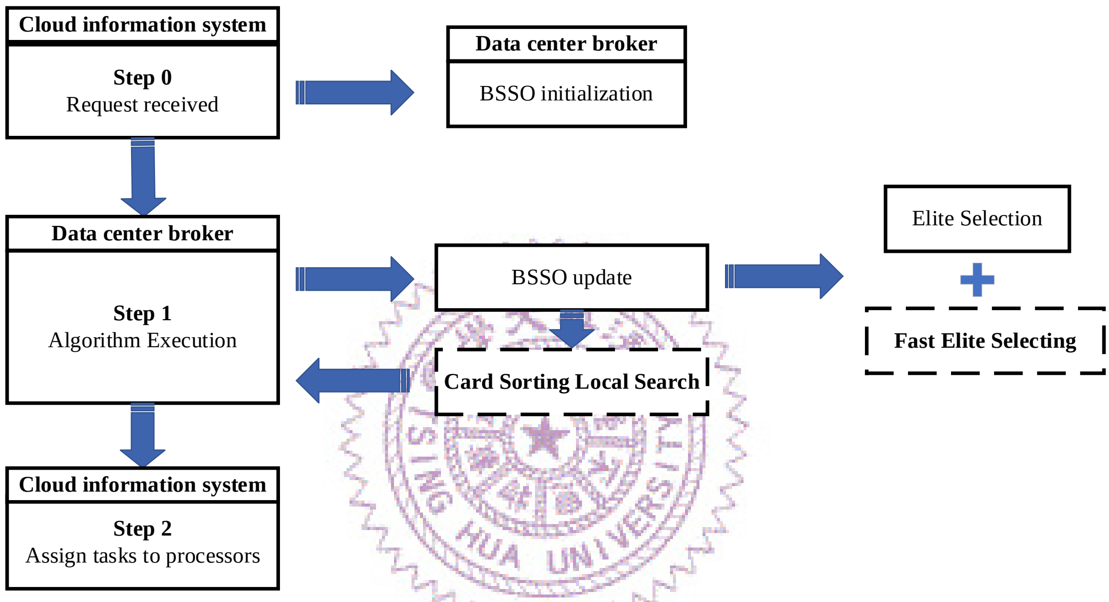

Figure 4-11 Task scheduling procedures in this study

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 starts 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 4-4 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
for *gen* =0 to Ngen **do**   *gBest* = selectGbest(A) //Randomly select one *gBest* from A   **for** *sol* =0 to Nsol **do**   StepwiseUpdate(X_{sol}, *gBest*) //**Elite selection SSO**   **end for**   *TempX* = merge (X^*, A) //merge two sets A^* = FastEliteSelecting (*TempX*) **end for**
Output: non-dominated solution archive A
3.Card Sorting Local Search
Input: non-dominated solution set archive A
while LGen is not reached **do** X = SetArchiveAsSolutions(A) //update archive as new X X^* = CardSorting(X)   *TempX* = merge (X^*, A) A^* = FastEliteSelecting (*TempX*) **end while**
Output: non-dominated solution archive A

Chaper 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 scale 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 5-1 to Table-5-3. Tasks length and tasks source are attached in **APPENDIX A**, processing rates and execution cost are attached in **APPENDIX B**.

Table 5-1 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 5-2 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 5-3 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

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 [32]. The IGD formula is derived as follows:

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

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[33].

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

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 [34]. The formula is shown below:

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

n is the number of solutions in P . e_i is a binary variable. If the solution found in P is exactly 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 one-way ANOVA test to determine C_s only and fix C_f =0.1 for our experiments.

For each size of problems, we set 3 different level 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:

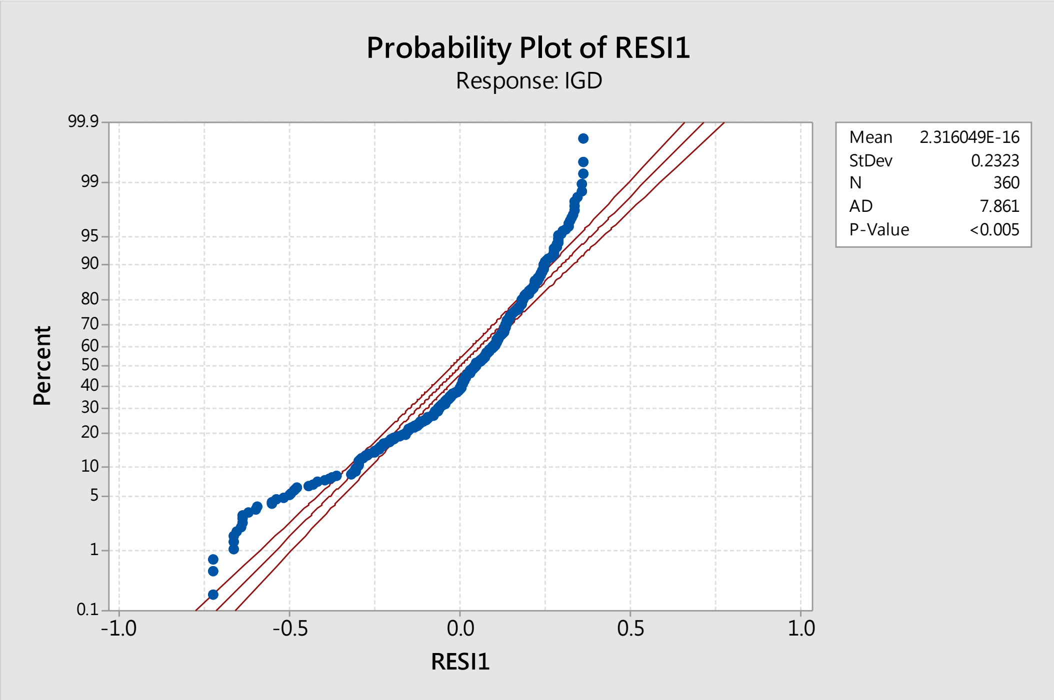

Figure 5-1 Normality test on IGD data

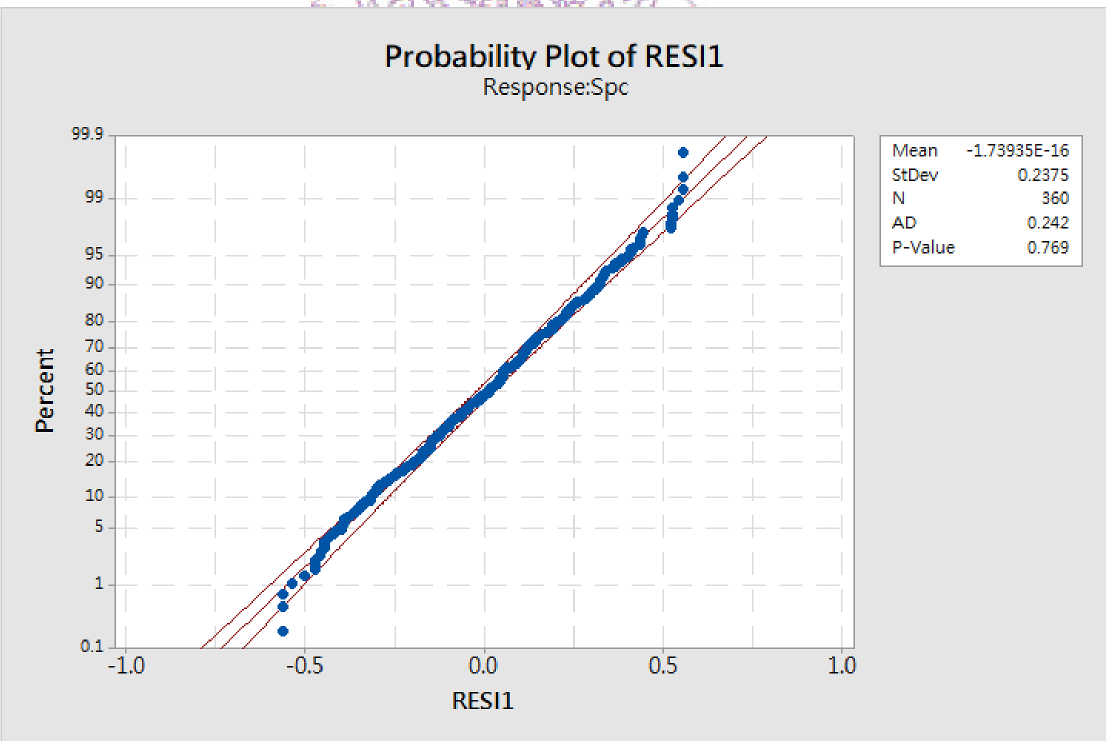

Figure 5-2 Normality test on Spc data

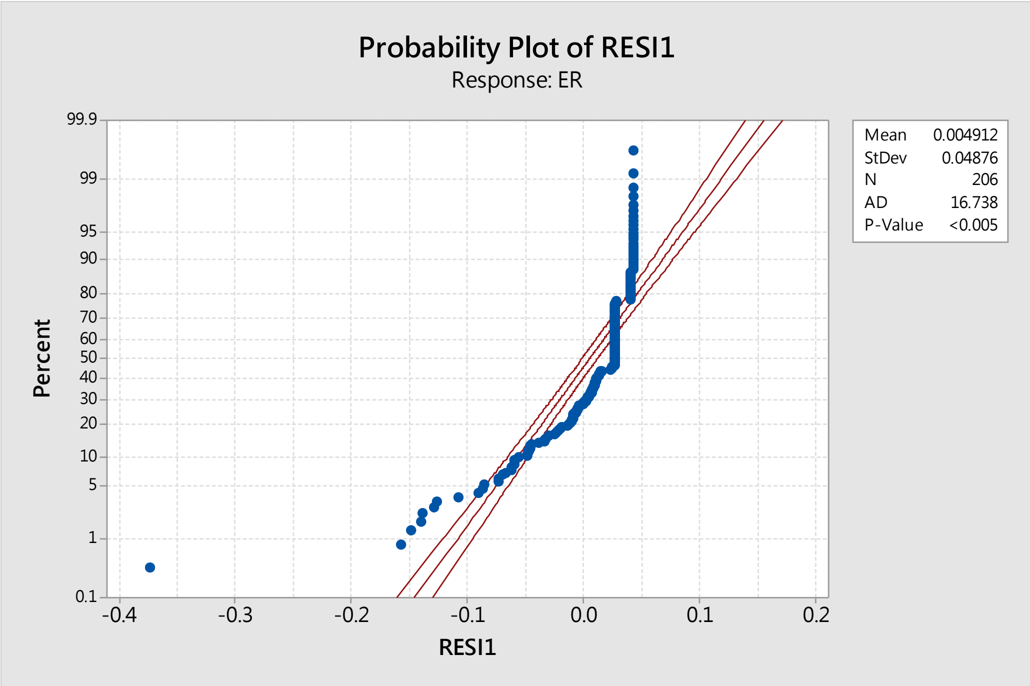

Figure 5-3 Normality test on ER data

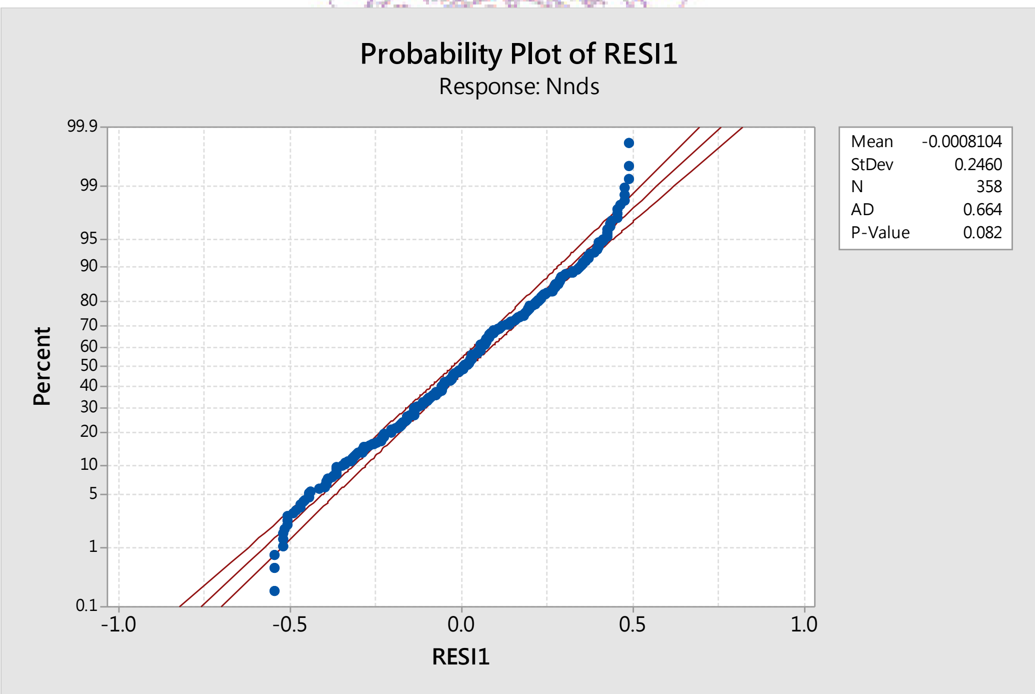

Figure 5-4 Normality test on Nnds data

To summarize, *Spc* and Nnds meets the normality assumption but IGD and ER don't. 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 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 5-4 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 the Table5-4 that different levels on IGD and *Spc* are significantly unequal. However, for ER and Nnds, we have no strong evidences to prove the difference between each level. In the following, we distinguish the difference between each level by interval plots.

1. IGD interval plot

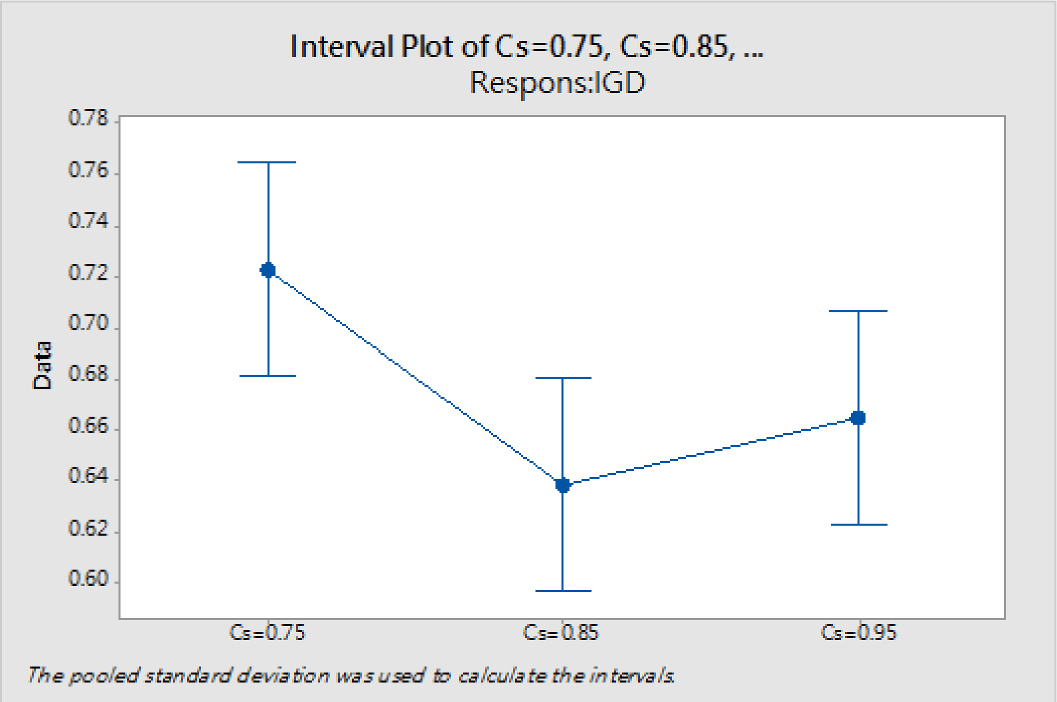

Figure 5-5 IGD interval plot under different levels

2. *Spc* interval plot

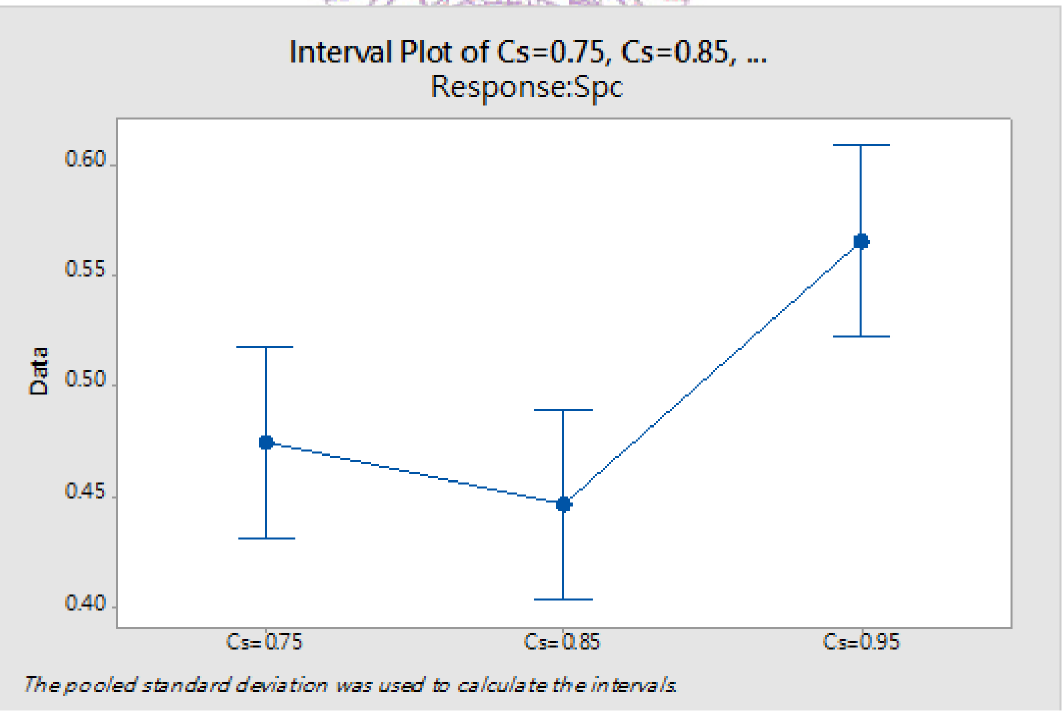

Figure 5-6 *Spc* interval plot under different levels

3. ER interval plot

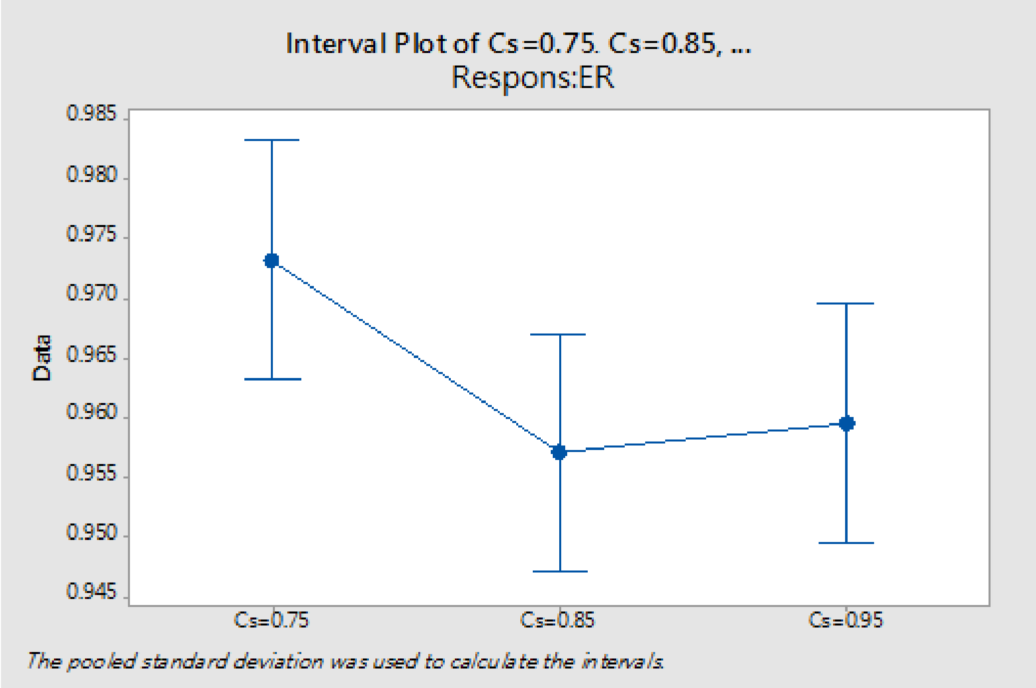

Figure 5-7 ER interval plot under different levels

4. Nnds interval plot

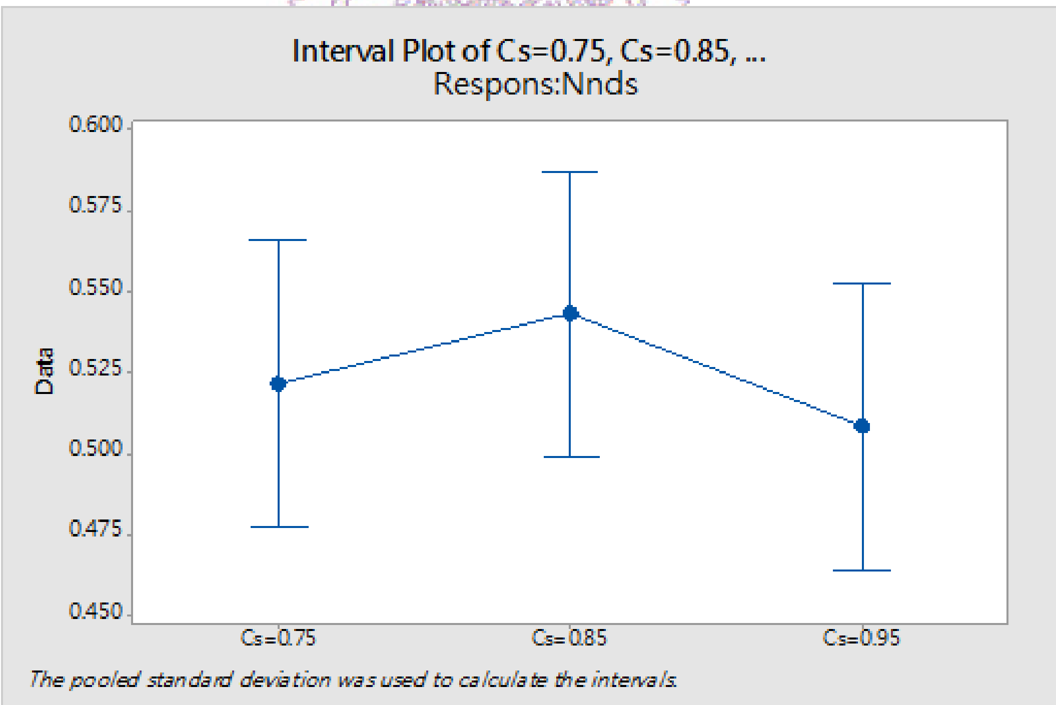

Figure 5-8 Nnds interval plot under different levels

5.3.3. Discussion and Conclusion of parameter

From the ANOVA table, interval plots and Figure5-5 to FigureF-8, we can infer the followings:

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 points 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.

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 5-5

Table 5-5 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 5-6.

Table 5-6 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 5-7, Table 5-8 and Table 5-9. The final non-dominated solutions of each results are shown after each table. Also, a brief discussion of the results is attached to the end of 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 thread for every algorithm (i.e., each algorithm was conducted 40 runs.).

Table 5-7 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}	$\bar{T}$	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}	$\bar{T}$	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}	$\bar{T}$	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**

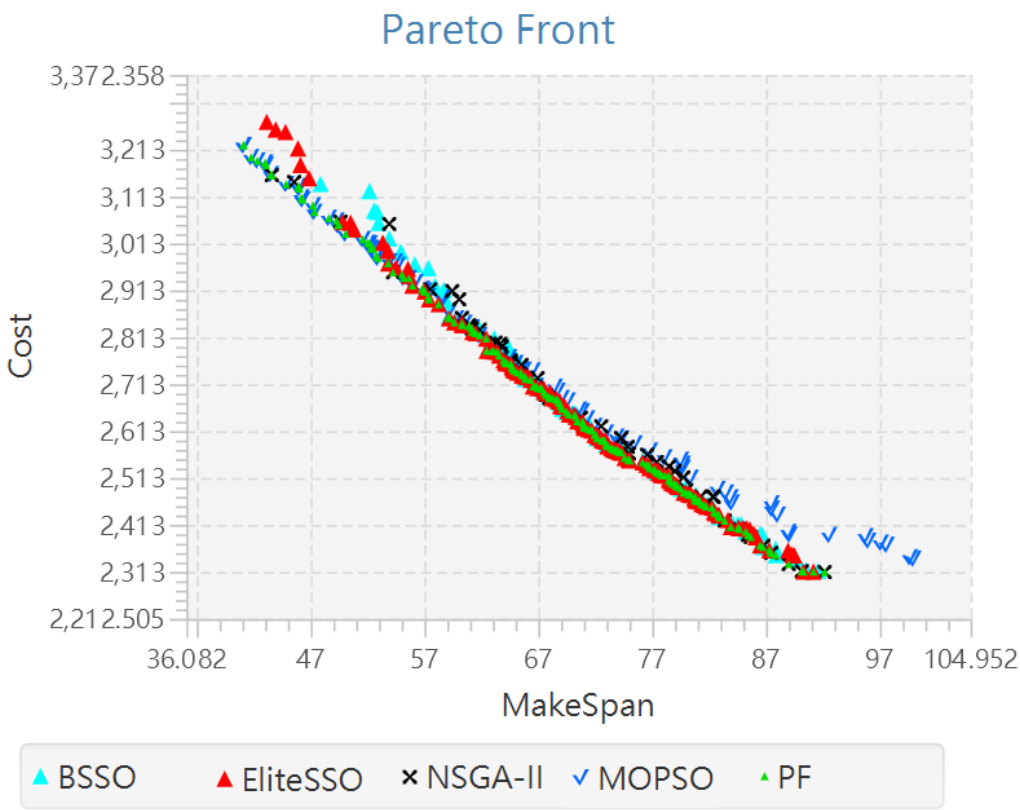

Figure 5-9 Small size, Dataset 1

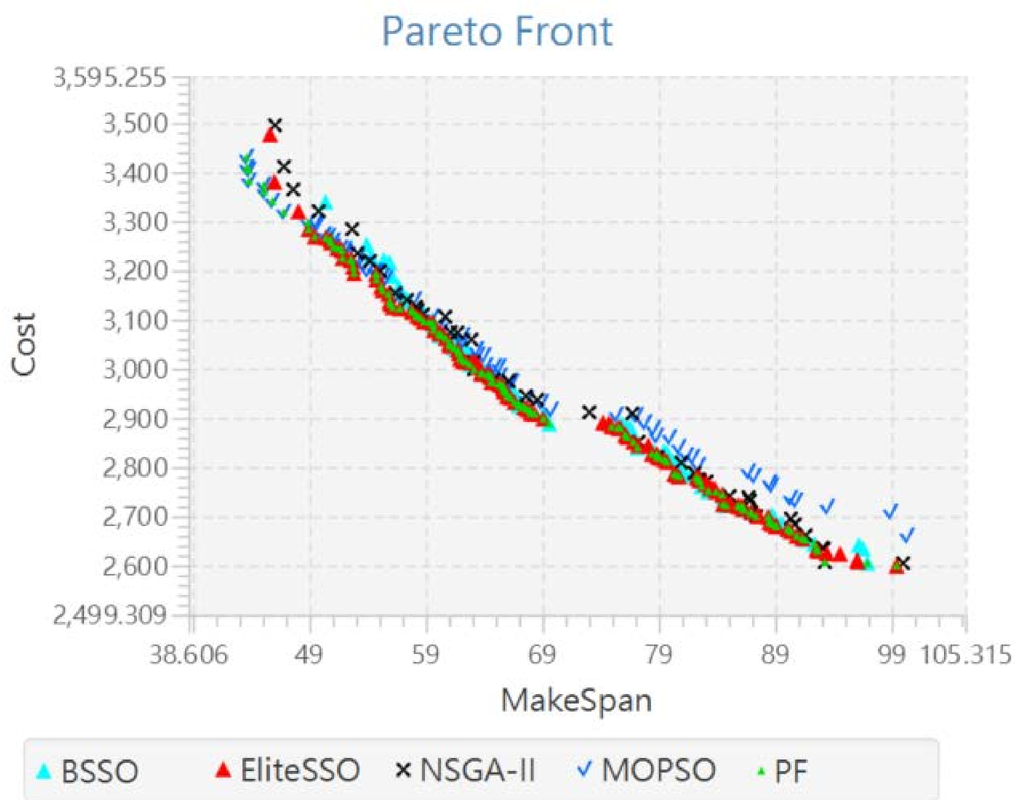

Figure 5-10 Small size, Dataset 2

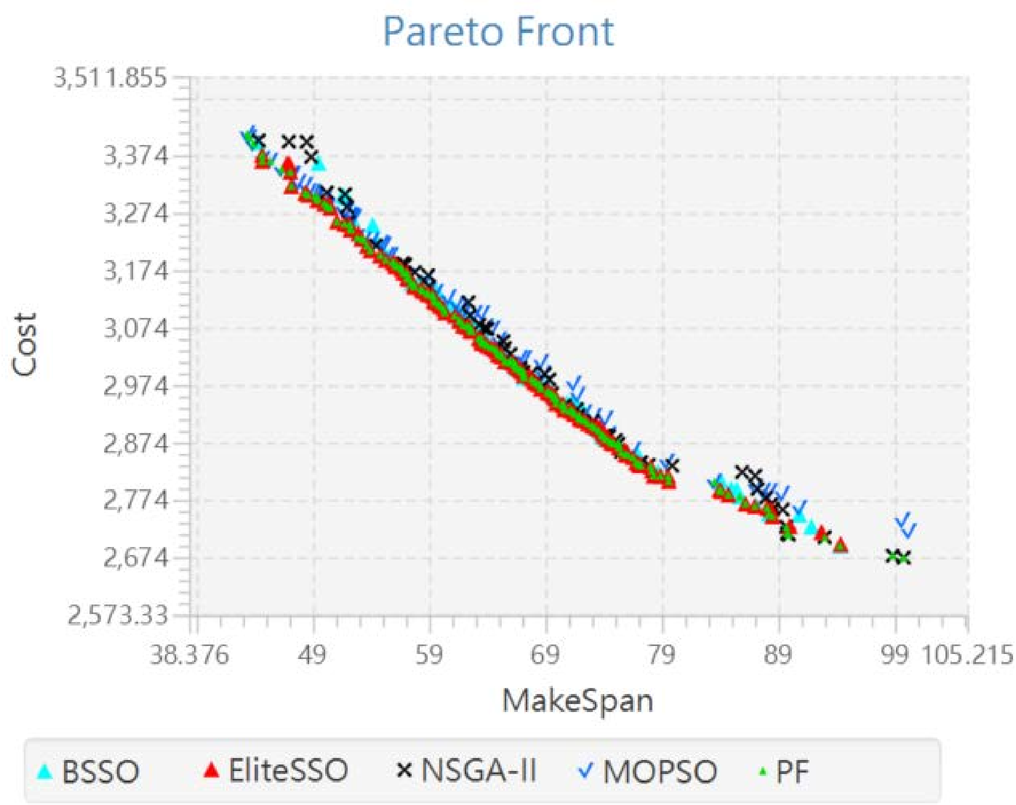

Figure 5-11 Small size, Dataset 3

In small scale problem, 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 Figure 5-11, 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 5-8 Experimental results of medium size problem

Algo.	$\overline{IGD}$	σ_{IGD}	$\overline{Spc}$	σ_{Spc}	$\overline{ER}$	σ_{ER}	$\overline{N_n}$	σ_{Nn}	$\bar{T}$	σ_T
M1										
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										
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										
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**

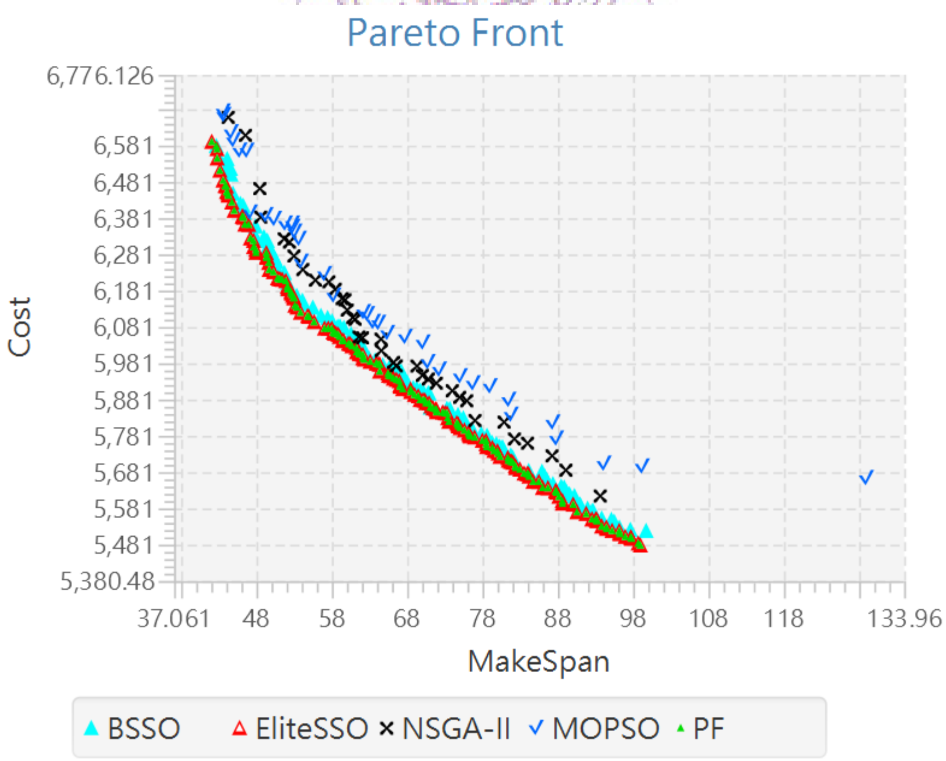

Figure 5-12 Medium size, Dataset 1

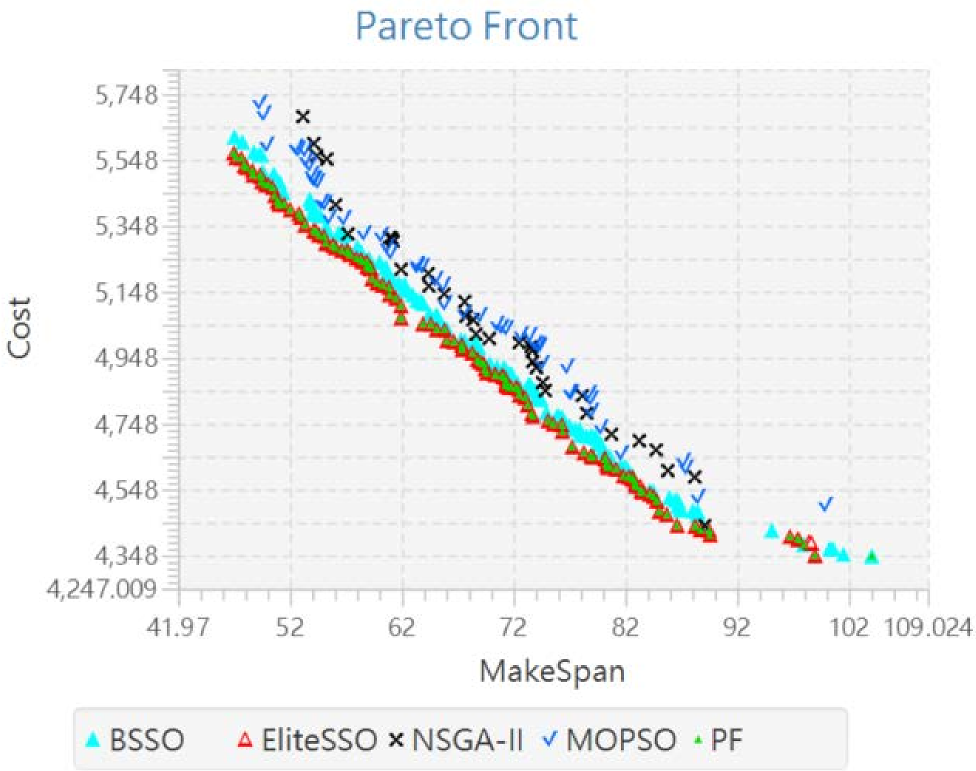

Figure 5-13 Medium size, Dataset 2

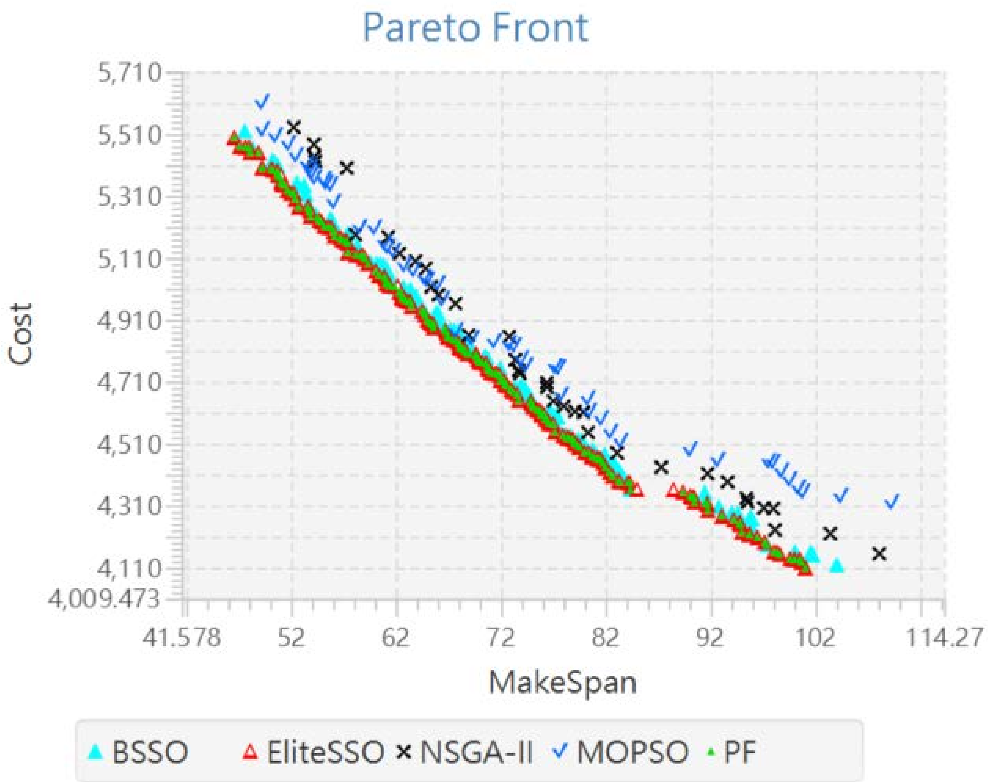

Figure 5-14 Medium size, Dataset 3

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

Table 5-9 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}	$\bar{T}$	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}	$\bar{T}$	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}	$\bar{T}$	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

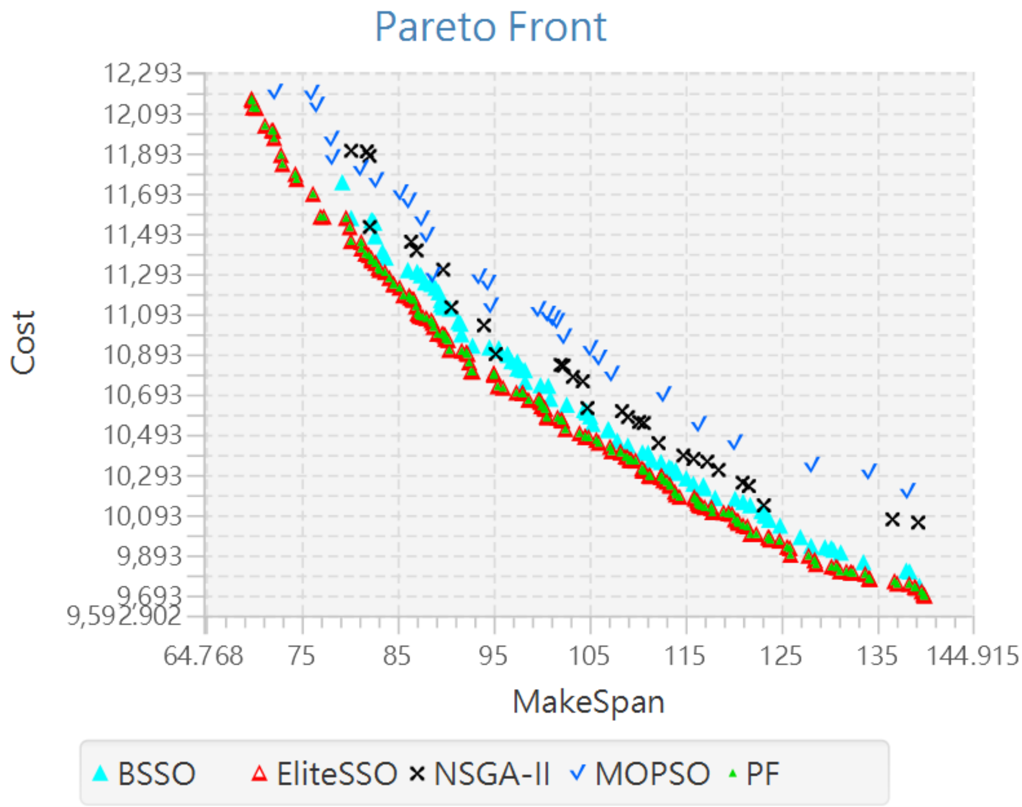

Figure 5-15 Large size, Dataset 1

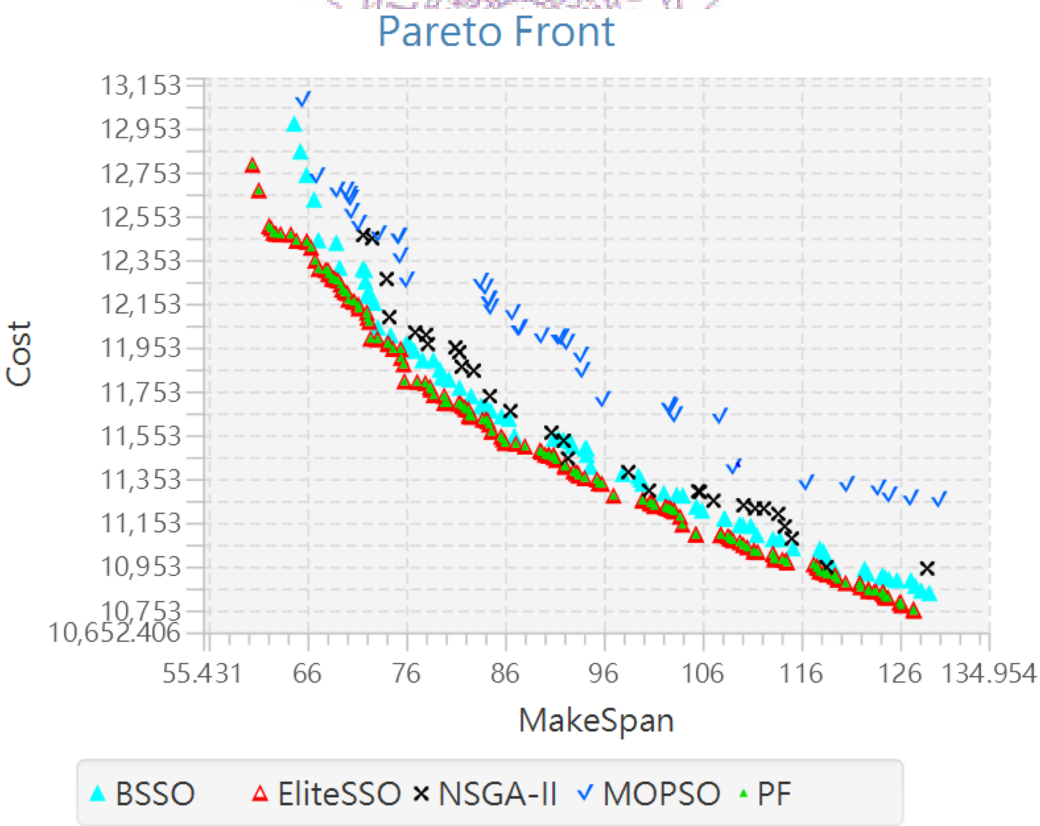

Figure 5-16 Large size, Dataset 2

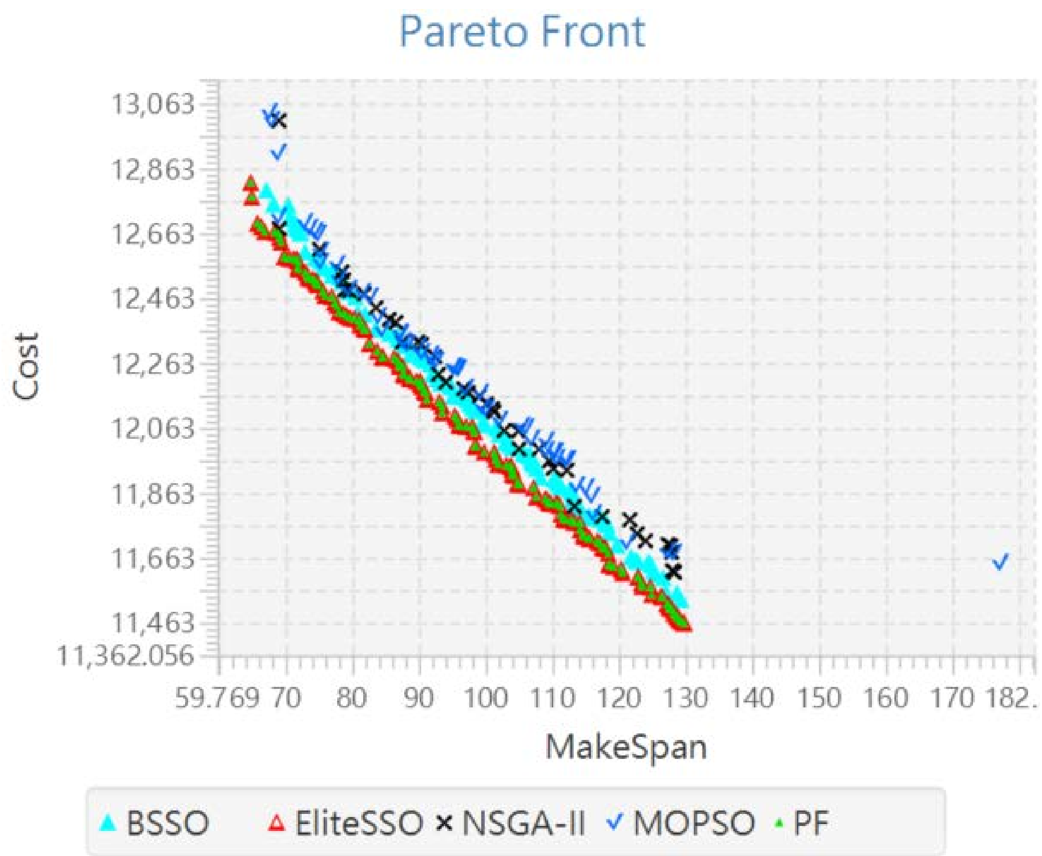

Figure 5-17 Large size, Dataset 3

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.

5.5 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 5-10 to Table5-12:

Table 5-10 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 5-11 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 5-12 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 Table5-13 and Table 5-14.

Table 5-13 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 5-14 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 search for the potential non-nominated solutions around the final non-dominated solutions of BSSO. Nevertheless, EliteSSO has significant difference with MOPSO and NSGA-II. For another metric, *ER*, EliteSSO is superior than all the other algorithms.

Chaper 6 Conclusion and Future works

6.1 Conclusion

In this paper, we devoted our effort in the following aspects:

1. We proposed a one-front non-dominated sorting skill, Fast Elite Selecting (FES) for BSSO.
   - **Contribution:** Successfully reduced the time complexity of sorting technique from from $O(MN^2)$ to $O(MN_{nds}^2)$
2. We also proposed a local search approach, Card Sorting, which focuses on the property of task scheduling.
   - **Contribution:** Enhanced the non-dominated solution exploration ability.
3. We dealt with the simulation delay problem in fog computing task scheduling from three aspects, i.e., algorithm time complexity, solutions exploration and parallel computing.
   - **Contribution:** Provided a future direction in simulation delay issue.
4. We provided a direction in utilizing multi-objective optimization algorithm in fog computing task scheduling problem.
   - **Contribution:** A new direction in solving this problem.

6.2 Future Work

We listed 3 directions of future work in the following:

1. GPU parallel computing:

In recent years, GPU is widely used in deep learning model which requires a tremendous computation burden to train a model. The saved computation time is significantly much. However, there is nearly no related works in swarm intelligence of evolutionary algorithm. We wish more and more scholars join and study this issue.

2. The Timing of Local Search

In this study, the local search is applied at the end of BSSO. It seems reasonable from the system utilization aspect. However, we cannot assure that some real pareto solutions are missed because of only one variable difference. Hence, we strongly suggest more research on this issue to develop a mechanism to determine the timing of local search during the update step.

3. Card Sorting or Card Forcing?

In this study, we design the CSLS with a fixed probability parameter C_s. If we can determine the operation based on the fitness of solutions, the effectiveness might be better.

According to our tests, we found that EliteSSO could beat MOPSO in Small scale problems by increasing C_f. MOPSO has a better ability in searching the solutions with more "0" which is the margin of variable domain. That is, more non-dominated solution which has shorter makespan were found by MOPSO. Hence, if we can increase C_f to strengthen the power of searching solutions with shorter makespan. EliteSSO might be able to dominate MOPSO.

Reference

1. Yannuzzi, M., et al. *Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing*. in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. 2014. IEEE.
2. Perera, C., et al., *Fog computing for sustainable smart cities: A survey*. 2017. **50**(3): p. 32.
3. Bonomi, F., et al. *Fog computing and its role in the internet of things*. in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012. ACM.
4. Matt, C.J.B. and I.S. Engineering, *Fog Computing*. 2018: p. 1-5.
5. Bitam, S., S. Zeadally, and A.J.E.I.S. Mellouk, *Fog computing job scheduling optimization based on bees swarm*. 2018. **12**(4): p. 373-397.
6. Vaquero, L.M. and L.J.A.S.C.C.R. Rodero-Merino, *Finding your way in the fog: Towards a comprehensive definition of fog computing*. 2014. **44**(5): p. 27-32.
7. Chiang, M. and T.J.I.I.o.T.J. Zhang, *Fog and IoT: An overview of research opportunities*. 2016. **3**(6): p. 854-864.
8. Sarkar, S. and S.J.I.N. Misra, *Theoretical modelling of fog computing: A green computing paradigm to support IoT applications*. 2016. **5**(2): p. 23-29.
9. Kui-kui, H., X. Zai-peng, and L.J.C.S. Xin, *Fog Computing Task Scheduling Strategy Based on Improved Genetic Algorithm*. 2018(4): p. 22.
10. Binh, H.T.T., et al. *An Evolutionary Algorithm for Solving Task Scheduling Problem in Cloud-Fog Computing Environment*. in *Proceedings of the Ninth International Symposium on Information and Communication Technology*. 2018. ACM.
11. Deng, R., et al., *Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption*. 2016. **3**(6): p. 1171-1181.
12. He, J., et al., *Time synchronization in WSNs: A maximum-value-based consensus approach*. 2014. **59**(3): p. 660-675.
13. Li, D. and X. Sun, *Nonlinear integer programming*. Vol. 84. 2006: Springer Science & Business Media.
14. Kuhn, H.W.J.N.r.l.q., *The Hungarian method for the assignment problem*. 1955. **2**(1-2): p. 83-97.
15. Fieldsend, J.E. and S. Singh, *A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence*. 2002.
16. Coello, C.A.C., G.T. Pulido, and M.S.J.I.T.o.e.c. Lechuga, *Handling multiple*

objectives with particle swarm optimization. 2004. **8**(3): p. 256-279.

17. Zhou, A., et al., *Multiobjective evolutionary algorithms: A survey of the state of the art.* 2011. **1**(1): p. 32-49.
18. Liu, J., et al., *Job scheduling model for cloud computing based on multi-objective genetic algorithm.* 2013. **10**(1): p. 134.
19. Jena, R.J.P.C.S., *Multi objective task scheduling in cloud environment using nested PSO framework.* 2015. **57**: p. 1219-1227.
20. Fard, H.M., et al. *A multi-objective approach for workflow scheduling in heterogeneous environments*. in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. 2012. IEEE.
21. Doğan, A. and F.J.T.C.J. Özgüner, *Biobjective scheduling algorithms for execution time–reliability trade-off in heterogeneous computing systems.* 2005. **48**(3): p. 300-314.
22. Yin, Y., *Multi-objective Task Scheduling in Cloud Environment Using Multi-objective Simplified Swarm Optimization*. 2018, National Tsin Hua University.
23. Yeh, W.-C.J.I.t.o.s., man,, c.-p.A. systems, and humans, *Optimization of the disassembly sequencing problem on the basis of self-adaptive simplified swarm optimization.* 2011. **42**(1): p. 250-261.
24. Yeh, W.-C.J.I.S., *A new exact solution algorithm for a novel generalized redundancy allocation problem.* 2017. **408**: p. 182-197.
25. Yeh, W.-C.J.K.-B.S., *Orthogonal simplified swarm optimization for the series–parallel redundancy allocation problem with a mix of components.* 2014. **64**: p. 1-12.
26. Yeh, W.-C.J.I.S., *Novel swarm optimization for mining classification rules on thyroid gland data.* 2012. **197**: p. 65-76.
27. Yeh, W.-C.J.E.S.w.A., *A two-stage discrete particle swarm optimization for the problem of multiple multi-level redundancy allocation in series systems.* 2009. **36**(5): p. 9192-9200.
28. Huang, C.-L. and W.-C.J.a.p.a. Yeh, *A new SSO-based Algorithm for the Bi-Objective Time-constrained task Scheduling Problem in Cloud Computing Services.* 2019.
29. Tasiopoulos, A., et al., *FogSpot: Spot Pricing for Application Provisioning in Edge/Fog Computing.* 2019.
30. Li, X. *A non-dominated sorting particle swarm optimizer for multiobjective optimization*. in *Genetic and Evolutionary Computation Conference*. 2003. Springer.
31. Deb, K., et al., *A fast and elitist multiobjective genetic algorithm: NSGA-II.* 2002. **6**(2): p. 182-197.

32. Czyzżak, P. and A.J.J.o.M.C.D.A. Jaszkiewicz, *Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization.* 1998. **7**(1): p. 34-47.
33. Schott, J.R., *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization*. 1995, AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH.
34. Van Veldhuizen, D.A., *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*. 1999, AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING.

APPENDICES

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*

B. 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*