VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Web Service Distribution Design

Boxiong Tan

Supervisors: Hui Ma, Yi Mei, Mengjie Zhang

Submitted in partial fulfilment of the requirements for
Master of Computer Science.

Abstract

Contents

Figures

Chapter 1

Introduction

1.1 Introduction

Modern enterprises need to respond effectively and quickly to opportunities in today's competitive and global markets. To accommodate business agility, companies are supposed to use existing applications instead of developing them from scratch. A contemporary approach for addressing these critical issues is embodied by (Web) services that can be easily assembled to form a collection of autonomous and loosely coupled business processes [32].

The arising of Web services developments and standards in support of automated business integration has driven major technological advances in the integration software space, most notably, the Service Oriented Architecture (SOA) [9]. In an SOA, software resources are packaged as Web services, which are well defined, self-contained modules that provide standard business functionality and are independent of the state or context of other services [34].

A Web service is a software system allowing to expose services via Internet. The SOA promotes the composition of coarse-grained web services to build more complex web applications using standards such as WS-BPEL [31]. Because of the convenience, low cost [1] and capacity to be composed into high-level business processes, Web service technology is becoming increasing popular.

With the ever increasing number of functional similar Web services being available on the Internet, the Web service providers (WSPs) are trying to improve the quality of service (QoS) to become competitive in the market. QoS, also known as non-functional requirements to Web service, is the degree to which a service meets specified requirements or user needs [46], such as response time, security and availability. Among numerous QoS measurements, service response time is a critical factor for many real-time services, e.g. traffic service or finance service.

Service response time has two components: transmission time (variable with message size) and network latency [21]. Study [20] shows that network latency is a significant component of service response delay. Ignoring network latency will underestimate response time by more than 80 percent [37], since network latency is related to network topology as well as physical distance [18]. To reduce the network latency, large Web service providers like Google, Facebook or Microsoft have their own high-bandwidth data centers. The majority of WSPs can not afford to build a data center, therefore they rent servers provide by Web service intermediaries (WSIs). WSPs need to allocate their services wisely so that the overall network latency is minimized. According to a popular Web traffic analyzer, Alexa, 96% of top 1 million web services were hosted in heterogeneous server clusters or co-location centers [15] that were widely distributed across different geographic regions.

Hence, it is not only necessary but urgent to provide an effective Web services allocation guide to WSPs so that they can be benefited.

The Web service location-allocation problem is essentially a multi-objective optimization problem [5], for which there are two conflict objectives, to maximize QoS and to minimize the deployment cost. This problem can be classified as a combinatorial optimization problem, that is, it is considered NP-hard due to the fact that the combinatorial explosion of the search space [39].

Evolutionary multi-objective optimization (EMO) methodologies are ideal for solving multi-objective optimization problems [11], since EMO works with a population of solutions. With an emphasis for moving toward the true Pareto-optimal region, an EMO can be used to find multiple Pareto-optimal solutions in one single simulation run [22]. The first major challenge is to model the Web service location-allocation into a suitable representation so that the EC algorithms and EMO techniques can be employed. The second challenge is to developed EC algorithms to solve this problem.

To model the service location-allocation problem, we consider the following assumptions. In order to apply optimization technique over the Web service location allocation problem, Some basic information must be provided by WSPs. A list of user centers, WSP must provides a list of user centers that each of them indicates a center city of a user concentrated area. This step can be achieved by analyzing Web requests data from existing services or conducting a market survey;

A list of candidate locations, a candidate location is the geometric location that is suitable to deploy services (e.g. existing WSIs). The selection of a candidate location is based on other criterion such as facilities or fixed fees of servers. User centers and candidate locations are not necessarily overlapped. Service invocation frequency denotes the number of invocation from a user center to a Web service within a period of time; Average network latency denotes the latency between user centers and candidate locations over a period of time. It is a critical QoS measurement criterion where lower latency means better QoS; Web service fixed cost denotes the rental of a server in each candidate location.

In summary, the list below shows some critical information that should be provided by the WSPs.

1. A list of user centers
2. A list of candidate locations
3. Service invocation frequencies from user centers to services
4. Average network latency from user centers to candidate locations
5. Web service fixed cost for each candidate location

Worth noting the data in the above table are changing over time, some of the changes are critical. For example, the invocation frequency from a user center U to a Web service W drops rapidly. It indicates the service W was popular in U not becomes unfrequented. At this point, existing Web services need to be re-allocate in order to adjust to this change. Other scenarios such as, the network latency vary in different time period within a day. Existing Web services do not need to be re-allocated because the average network latency remain stable.

1.2 Objectives

The overall objectives of this project are, firstly, developed a model of Web service location allocation problem that suitable for EC algorithms. Secondly, apply EC algorithms to solve this problem. The performance of Web service distribution plan will be evaluated on a number of datasets with different numbers of user centers, candidate locations and services. In particular, the main objectives can be divided into two categories:

- To develop a common model for solving Web service location-allocation problem.
- To develop EC algorithms to solve the problem.

1.3 Major Contributions

The project have major three contributions. First, we developed a general model for solving Web service location-allocation problem. The data are modeled as matrix-based representation. This model can be easily adopted by various EC algorithms.

Second, we addressed the Web service location allocation by developing a single objective binary PSO and three multi-objective EC algorithms. We addressed the disadvantages of using single-objective algorithm to solve multi-objective problems. Three multi-objective algorithms: NSGA-II, NSPSO and MOPSOCD are first used in solving Web service location allocation problem.

Third, we developed a improved multi-objective PSO approach with two major improvements. 1. A rounding function mechanism makes a continuous algorithm compatible with binary problems. 2.An adaptive threshold algorithm is developed to embody the idea of transfer learning.

1.4 Organization

The report is organized as follows. Chapter 2 provides background information. Chapter 3 presents a binary PSO based algorithm. Chapter 4 presents two multi-objective algorithms. Chapter 5 presents an improved multi-objective PSO with crowding distance. Chapter 6 provides a discussion of the conclusion and some remaining future work.

Chapter 2

Background

2.1 Web Service Location Allocation

The recent development of Semantic Web provides a common framework that allows data to be shared and reused across application, enterprises and community boundaries. That is, Web services are become a on-line resource. Similar with Traditional industry, Web service providers and customers can be benefited from an appropriate allocation of Web services. From WSPs' perspective, it minimizes the number of deployed servers. From customers' perspective, it improves the QoS. However, the trade-off between cost and QoS occurs when WSPs try to fulfill all stakeholders' objectives. Similar to Web Service location allocation, Cloud computing resource management has encounter the same problems. The fundamental objectives of these two problems include allocation price, Quality of service (response time, user satisfaction) [13].

2.2 Evolutionary Computation

2.2.1 Non-dominated Sorting Genetic Algorithm II (NSGA-II)

NSGA-II is a multi-objective algorithm based on genetic algorithm (GA). It was proposed by Klyanony and et.al [10] in 2002. NSGA-II performs well in convergence and permits a remarkable level of flexibility. It has four innovative properties, a fast non-dominated sorting procedure, an elitist strategy, a parameterless approach and an efficient constraint-handling method.

2.2.2 Particle Swarm Optimization (PSO)

PSO proposed by Kennedy and Eberhart in 1995 [23]. PSO is a population based metaheuristic algorithm inspired by the social behavior of birds and fishes. In PSO, each individual is called a particle which flying around the search space. The underlying phenomenon of PSO is optimized by social interaction where particles sharing information to direct their movement.

PSO is based on the principle that each solution can be represented as a particle. At initial state, each particle has a random initial position in the search space which is represented by a vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$, where D is the dimensionality of the search space. Each particle has a velocity, represented as $v_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$ which is limited by a predefined maximum velocity, v_{max} and $v_{id} \in [-v_{max}, v_{max}]$. During the search process, each particle maintains a record of previous best performance, called *pbest*. The best position of its neighbors is also recorded, which is *gbest*. The position and velocity of each particle are updated according to the following equations:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{2.1}$$

$$v_{id}^{t+1} = w * v_{id}^{t} + c_1 * r_{1i} * (p_{id} - x_{id}^{t}) + c_2 * r_{2i} * (p_{pg} - x_{id}^{i}) \tag{2.2}$$

In the two equations, t shows the t^{th} iteration. $d \in D$ shows the d^{th} dimension. w is the inertia weight used to balance the local search and global search abilities of PSO. c_1 and c_2 are acceleration constants. r_{1i} and r_{2i} are random constants uniformly distributed in $[0, 1]$. p_{id} and p_{gd} denote the values of *pbest* and *gbest* in d^{th} dimension.

2.2.3 Binary PSO

PSO was originally developed to address continuous optimization problems. Therefore, the representation for both position and velocity of a particle in PSO is a vector of real numbers. However, this representation is not suitable for many discrete optimization problems. To address the discrete problem, in 1997 Kennedy and Eberhart developed a binary particle swarm optimization (BPSO) [24]. In BPSO, the position of each particle is a vector of binary numbers, which are restricted to 1 or 0. The velocity in BPSO represents the probability of the corresponding position taking value of 1. A sigmoid function is used to transform a velocity between 0 and 1. The following equation is used to update the position of each particle:

$$x_{id} = \begin{cases} 1 & \text{if } rand() < s(v_{id}) \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$s(v_{id}) = \frac{1}{1 + e^{-v_{id}}} \tag{2.4}$$

The rand() is a random number selected from a uniform distribution in $(0, 1)$.

2.2.4 Multi-objective PSO

Several multi-objective optimization algorithms are based on PSO such as Multi-objective PSO (MOPSO) [6], Non Dominated Sorting PSO (NSPSO) [29]. The performance of different multi-objective algorithms was compared in [6] using five test functions. These algorithms are NSGA-II [10], PAES [26], Micro-GA [8] and MOPSO. The results show that MOPSO was able to generate the best set of non-dominated solutions close to the true Pareto front in all test functions except in one function where NSGA-II is superior. Another major advantage for PSO-based algorithms is low computational cost. It is much effective than other EC algorithms.

Raquel and et al. [35] proposed a MOPSOCD that extends the MOPSO. The mechanism of crowding distance is incorporated into the algorithm on global best selection of an external archive of non-dominated solutions. The diversity of non-dominated solutions in the external archive is improved by using the crowding distance together with a mutation operator. The performance shows that MOPSOCD is highly competitive in generating a well-distributed set of non-dominated solutions.

2.3 Related work on Service Location-allocation

2.3.1 Traditional Service Location-allocation and Resource Management in Cloud

Most of the researchers treat service location-allocation problem a single objective problem. [1] [37] try to solve the problem by using integer linear programming techniques. In particular, [37] solved this problem by employing greedy and linear relaxations of Integer transportation problem.

Research on network virtualization [12, 4] has employed greedy algorithm allocate virtual machines (VMs) in the data center so that requirements of network bandwidth are meet. [27] presents a multi-layer and integrated fashion through a convex integer programming formulation.

The major drawback of greedy algorithm is that it easy to stuck at local optima. Linear programming is well-known as not scaling very well. They perform poorly when the number of variables is large.

2.3.2 EC Approaches

Huang [16] proposed an enhanced genetic algorithm (GA)-based approach, which make use of the integer scalarization technique to solve this problem. This algorithm solves the problem with one objective and one constraint. However there are some deficiencies in the integer scalarization techniques [5]. Firstly, decision makers need to choose appropriate weights for the objectives to retrieve a satisfactorily solution. Secondly, non-convex parts of the Pareto set cannot be reached by minimizing convex combinations of the object functions.

Kessaci [25] propose MOGA-CB for minimizing cost of VMs instance and response time. [33] proposed a framework, Green Monster, to dynamically move Web services across Internet data centers for reducing their carbon footprint while maintaining their performance. Green monster applies a modified version of NSGA-II algorithm [10] with an additional local search process.

2.4 Summary

Previous researchers have studied Web services location-allocation problem with single-objective algorithms, linear programming technique and greedy algorithm. These approaches have many obvious disadvantages (Section 2.3.1). Web service location-allocation problem in nature is a multi-objective problem should be addressed by multi-objective algorithms. From previous study, we found evolutionary multi-objective optimization (EMO) algorithms are promising. Specifically, among many EMO algorithms, multi-objective particle swarm optimization with crowding distance (MOPSOCD) is a recent development. It outperforms other approaches including NSGA-II, PAES in various aspects.

Chapter 3

Web Service Location Allocation Model Formulation

3.1 Introduction

In the previous study, Huang [16] developed a model of Web service location allocation. There are mainly two shortcomings of the model. Firstly, this model is based on a web service composition assumption, where all atomic services are composed to provide functionalities. This assumption does not consider atomic services could provide service independently. In other words, this model does not consider standalone Web services. Secondly, a major input data *atomic service demand* which indicates the popularity of a service is calculated based on assuming 20% Web services account for 80% of overall service requests. There is no validation for this assumption. Therefore, two problems may occur when applying this model on a real world problem.

1. The standalone Web services can not be modeled.
2. Only 20% of Web services are taken into account.

It is obvious this model is not general enough to solve real world problems. Hence, this chapter addresses Objective 1, aiming at developing a general model for Web service location-allocation problem. The model is not only more generalized but also could be easy to applied in EC algorithms. The model is presented in Section 3.2. In order to validate the model and provide a baseline of the optimized solution, we developed a single objective BPSO to accomplish this task. In BPSO, we use a linear aggregation method to combine two objectives into a *Fitness*. We use the normalized objectives to present the results.

3.2 Model Formulation

To model service location-allocation problem, we need to make use of a set of matrices, to present input information and output solutions.

For service location-allocation problem, we need information of service invocation frequency, network latency, and service fixed cost to decide service location-allocation so that the overall network latency can be minimized with minimal fixed cost and within constraints. Assume a set of $S = \{s_1, s_2, \ldots, s_s, s_x\}$ services are requested from a set of locations $I = \{i_1, i_2, \ldots, i_i, i_y\}$. The service providers allocate services to a set of candidate locations $J = \{j_1, j_2, \ldots, j_j, j_z\}$.

In this paper, we will use the following matrices.

Matrices	
L	server network latency matrix $L = \{l_{ij}\}$
A	service location matrix $A = \{a_{sj}\}$
F	service invocation frequency matrix $F = \{f_{is}\}$
C	cost matrix $C = \{c_{sj}\}$
R	user response time matrix $R = \{r_{is}\}$

A *service invocation frequency matrix*, $F = [f_{is}]$, is used to record services invocation frequencies from user centers, where f_{is} is an integer that indicates the number of invocations in a period of time from a user center to a service. For example, f_{13} = 85 denotes service s_1 is called 85 times in a predefined period of time.

$$F = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \end{array} \begin{array}{c} \begin{array}{ccc} s_1 & s_2 & s_3 \end{array} \\ \begin{bmatrix} 120 & 35 & 56 \\ 14 & 67 & 24 \\ 85 & 25 & 74 \end{bmatrix} \end{array} \qquad L = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 0 & 5.776 & 6.984 \\ 5.776 & 0 & 2.035 \\ 0.984 & 1.135 & 2.3 \end{bmatrix} \end{array}$$

A *network latency matrix* $L = [l_{ij}]$, is used to record network latencies from user centers to candidate locations. For example, the network latency between user center i_2 with candidate location j_1 is 5.776s. These data could be collected by monitoring network latencies [44] [45].

The cost matrix, $C = [c_{sj}]$, is used to record the fixed fees of servers in candidate locations, where c_{sj} is an integer that indicates the fixed fee of a server to a candidate location. For example, $c_{12} = 80$ denotes the cost of deploying service s_1 to candidate location j_2 is 80 cost units.

$$C = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 130 & 80 & 60 \\ 130 & 80 & 60 \\ 130 & 80 & 60 \end{bmatrix} \end{array} \qquad A = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \end{array}$$

Using service location allocation matrix $A = [a_{sj}]$ and network latency matrix $L = [l_{ij}]$, we can compute user response time matrix $R = [r_{is}]$,

$$r_{is} = MIN\{l_{ij} \mid j \in \{1, 2, ..., z\} \text{ and } a_{sj} = 1\} \tag{3.1}$$

For example, we can use the two example matrices L and A presented above to construct the response time matrix R. For each service s, by checking matrix A, we can find out which location the service has been deployed. Then we check matrix L, to find out its corresponding latency to each user center i. If there is more than one location, then the smallest latency is selected. Therefore, we can construct the response time matrix R as:

$$R = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \end{array} \begin{array}{c} \begin{array}{ccc} s_1 & s_2 & s_3 \end{array} \\ \begin{bmatrix} 5.776 & 6.984 & 0 \\ 0 & 2.035 & 0 \\ 1.135 & 2.3 & 0.984 \end{bmatrix} \end{array}$$

In order to accomplish these two objectives, we design two objective functions to evaluate how good each particle meets the objectives. We use *Cost* to calculate the overall cost of deploying services under an allocation plan

$$Cost = \sum_{s \in S} \sum_{j \in J} c_{sj} a_{sj} \tag{3.2}$$

where c_{sj} is the cost of deploying service s at candidate location j, a_{sj} represents whether service s is allocate to candidate location j. The sum of the multiplication of c_{sj} and a_{sj} is the total deployment cost.

For example, we use the above mentioned matrices C and A.

$$\begin{aligned} Cost &= c_{11} * a_{11} + c_{12} * a_{12} + c_{13} * a_{13} + ... + c_{33} * a_{33} \\ &= 130 * 0 + 80 * 1 + 60 * 0 + ... + 54 * 0 \\ &= 228 \end{aligned}$$

We use *Latency* to calculate the overall network latency. Where r_{is} denotes the optima response time from a user center i to a service s and f_{is} is the invocation frequency of a user center i to a service s.

$$Latency = \sum_{i \in I} \sum_{s \in S} r_{is} f_{is} \tag{3.3}$$

For example, we use the above mentioned matrices F and R.

$$\begin{aligned} Latency &= f_{11} * r_{11} + f_{12} * r_{12} + f_{13} * r_{13} + ... + f_{33} * r_{33} \\ &= 120 * 5.776 + 6.984 * 35 + 0 * 56 + ... + 0.984 * 74 \\ &= 1300.696 \end{aligned}$$

3.3 Binary Particle Swarm Optimization

In order to provide a baseline of optimization solutions and validate the model. We adopt the original BPSO. The key step is updating of *pbest* and *gbest*. The problem is trying to minimize the fitness value. Therefore, The *pbest* of a particle is updated when its current fitness is smaller than previous *pbest*. The *gbest* maintain the best solution in a global range. Hence, it is updated when a current *pbest* is smaller than previous *gbest*.

Algorithm 1 BPSO for Web Service Location-Allocation

```
Inputs:
Cost Matrix $C$,
Server network latency matrix $L$,
Service invocation frequency matrix $F$

Outputs: Pareto Front
1: Initialize the position and velocity of particles in the swarm ($Swarm$).
2: repeat
3:     Evaluate $Swarm$ with fitness functions, each particle has two objectives $Cost$ and $Latency$;
4:     Use linear normalization to normalize each particle's objectives
5:     Calculate a combined fitness value $F = (1 - w)Cost' + w * Latency'$;
6:     Update $pbest$ and $gbest$;
7:     Update velocity and position;
8: until maximum iterations is researched
9: Return $Swarm$
```

3.3.1 Fitness Function

We employ a linear aggregation method that all normalized objectives are combined into one and a weight w is assigned to each objective to indicate its relative importance (Equation 3.6).

Since the maximum and minimum values for total cost and total latency are deterministic, we use exhaustive search to find out the $Latency_{max}$. $Latency_{min}$ is achieved by deploying all services to all candidate locations. $Cost_{min}$ is the cost of allocating each of services at a location that leads to the minimal cost and $Cost_{max}$ is the cost of allocating each service is allocated to all the locations.

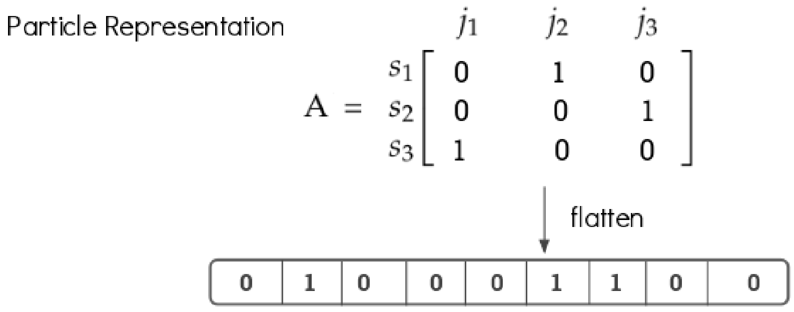
$$Cost' = \frac{Cost - Cost_{min}}{Cost_{max} - Cost_{min}} \tag{3.4}$$

$$Latency' = \frac{Latency - Latency_{min}}{Latency_{max} - Latency_{min}} \tag{3.5}$$

$$Fitness = w * Latency' + (1 - w) * Cost' \tag{3.6}$$

3.3.2 Particle Representation

In order to apply the BPSO on the Web service location-allocation model. A flatten matrix techniques need to be applied over the particle representation. Instead of directly using matrix-based representation, it uses a flattened matrix.



$$A = \begin{array}{c|ccc} & j_1 & j_2 & j_3 \\ \hline s_1 & 0 & 1 & 0 \\ s_2 & 0 & 0 & 1 \\ s_3 & 1 & 0 & 0 \end{array}$$

0	1	0	0	0	1	1	0	0

Figure 3.1: Matrix Flatten

3.3.3 Constraint Function and Constraint Handling

Five categories of constraint handling by Coello [7], 1. penalty functions, 2. Special representation and operators, 3.Repair algorithms, 4. Separation of objectives and constraints, 5. Hybrid Methods.

BPSO applies a "death penalty" strategy [7] in constraint handling where invalid solution is assigned the maximum fitness value.

$$Fitness(x_{id}) = \begin{cases} 1 & \text{if violate constraint} \\ \text{normalized fitness value} & \text{otherwise} \end{cases} \tag{3.7}$$

The invalid particles are remaining in the population, however, as a punishment, they temporarily lost the opportunity to become *gbest*. In the next generation, they could move out of invalid regions and become valid again. "Death penalty" method in PSO is not considered as harmful as in GA since all the invalid particles are still remained in the population.

We consider single basic constraint. The service number constraint, it requires that each service is deployed in at least one location.

$$\sum_{x \in S} a_{xj} \geq 1 \tag{3.8}$$

3.4 Experiment

3.4.1 Datasets

This project is based on both real world dataset and stimulate dataset. In this project, there are mainly 3 sub-datasets that needs to be provided, network latencies between candidate locations and user centers, server rental cost in candidate locations and Web service invocation frequencies information.

Table 3.1 shows the fourteen problem sets designed to study this problem, which were either generated from existing data or generated from distribution. In the problems we presented here, the problems will be in increasing size and difficulty. They are used as representative samples of the Web service location-allocation problems that the proposed algorithm can address.

Table 3.1: Problem set

Datasets	No. of Service	No. of Candidate Location	No. of user centers
Problem 1	20	5	10
Problem 2	20	10	10
Problem 3	50	15	20
Problem 4	50	15	40
Problem 5	50	25	20
Problem 6	50	25	40
Problem 7	100	15	20
Problem 8	100	15	40
Problem 9	100	25	20
Problem 10	100	25	40
Problem 11	200	25	40
Problem 12	200	25	80
Problem 13	200	40	40
Problem 14	200	40	80

Cost

A Web service provider rents servers from Web service intermediaries (WSIs). The rental fee are normally includes fixed fees and variable fees based on usage [36]. Fixed fees are refer to the monthly payment for a specific service plan from a WSI. Fixed fees is mainly depend on the quantity, namely the number of servers and quality features include computing power, reliability, network bandwidth. Variable fees includes extra charges from exceeded storage and other limits. This project would only consider fixed fees. The cost is under the following assumptions, the cost of services in the same candidate location are the same. This project uses stimulate cost by randomly generate from a normal distribution where mean equals 100, standard deviation equals 20. In real situation, the cost can be real rental cost.

Service Invocation Frequency

Frequency refers to the invocation frequency from a location to a Web service within a certain period of time [17]. This project uses simulated frequency data that randomly generate from a uniform distribution from 1 to 120. In real situation, the frequency data could be replaced by real data collected by servers.

Network Latency

Network latency denotes the latency between a user center and a candidate location. The network latency is not only depend on the geographic locations but also the topological location. The project sees the network topology as a black box and uses network latency as the network quality measurement. A network latency data set is provided by WS-DREAM [44] [45], it measures the network latency between 339 user centers to 5825 candidate locations. The provided data is represented as a matrix.

$$LatencyData = \begin{array}{c|cccc|} & j_1 & j_2 & \dots & j_{5825} \\ i_1 & 0.52 & 3.8 & \dots & 5.6 \\ i_2 & 1.4 & 6.7 & \dots & 2.4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ i_{339} & 8.5 & 2.5 & \dots & 7.4 \end{array}$$

The project uses subsets of the total matrix. It randomly selects a subset of the total matrix as the latency matrix L, where the number of user center and candidate location are defined by experimental design. The only selection constraint is the network latency value can not be *None* value since we assume the Web service provider provides the complete data.

For example, the experiment requires a $N * M$ latency matrix, that is, N user centers and M candidate locations. In the first step, it randomly selects a $N * M$ matrix from the data matrix. In the second step, it replaces the row which has *None* value with a random row. Repeat the second step until all matrix has None value.

Optimization is a process that highly depend on input data. Therefore, it is important to understand the distribution of input data. As figure 3.2 shows,

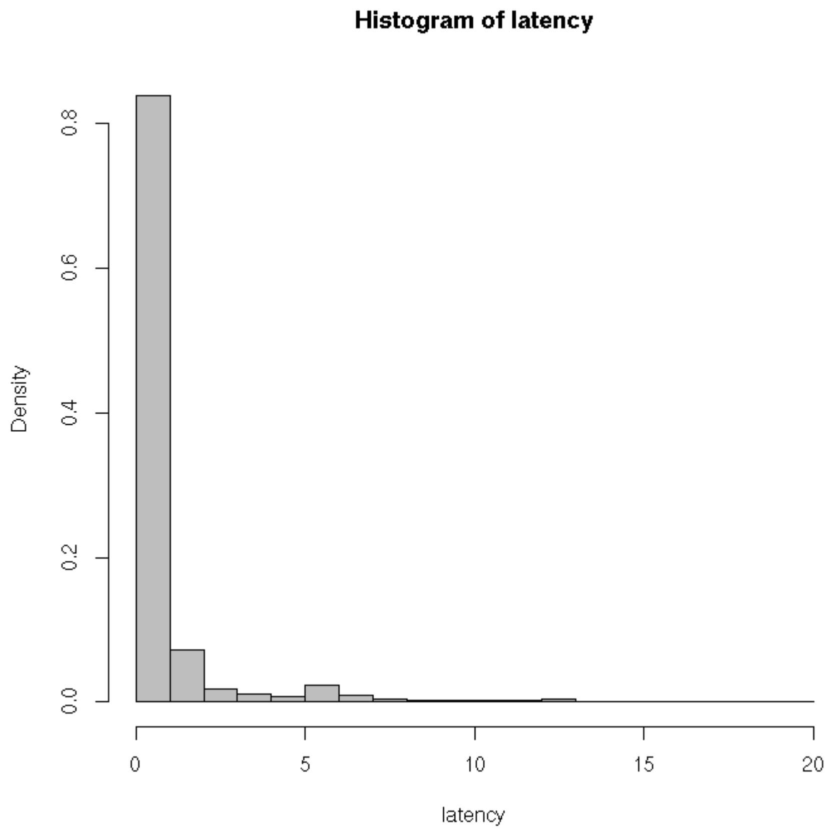


Figure 3.2: latency density

Table 3.2: Statistics of Latency data

Min	1st Quartile	Median	Mean	3rd Quartile	Max
0.001	0.156	0.32	0.9086	0.6720	20.0

Over 90% of the latency data concentrate on [0, 2] seconds. The range of network latency are from [0, 20]. As it clearly shows the frequency of the latency below zero account for the majority of the dataset. The extreme skew data has an effect on the experimental result.

3.4.2 Experiment Design

We conduct experiment on problem 1 $\sim$ 6. The parameters of the BPSO algorithms are set as follow: $c1$ = 2, $c2$ = 2, population size is 50 and the max number of iteration is 50. We assume both objectives are equally important, therefore, a weight w for linear aggregation is set to 0.5.

3.4.3 Experiment Results

The results are shown in terms of two separated objectives: cost and latency. As we expected, BPSO generates a set of solutions. In order to validate the model. We furthur tracked the optimization process of BPSO. We collected the solutions from 10th, 20th, 30th, 40th, and 50th generation and presents in Figure 3.4. In the beginning of an optimization process, BPSO randomly initialize the population. Then the population gradually moving towards the original point. We could observe the optimization process clearly from problem 1 and 2. As solutions moves from blue region to black region. It is not so obvious in problem 3 and 4. The reason is the BPSO could not find better solutions so the swarm schooling along the borderline which every solutions are equally good. This experiments show the solutions are optimized and provide a baseline of solutions.

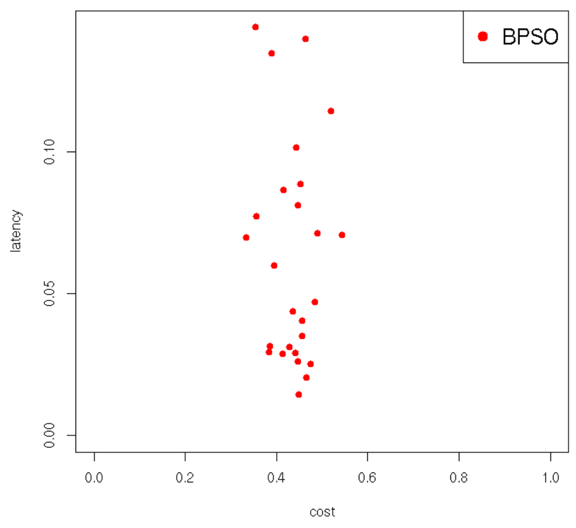
3.5 Discussion

In the Section 3.1, we discussed the new model overcome two problems faced by Huang's model.

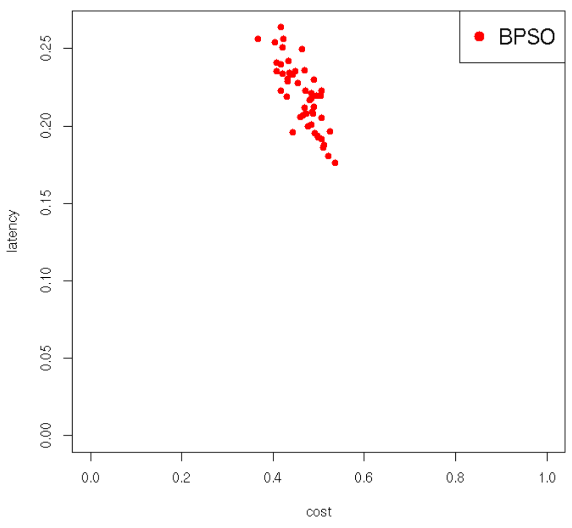
1. The standalone Web services can not be modelled.
2. Only 20% of Web services are taken into account.

We resolve these problems by introducing a new input data: invocation frequency. Inocation frequency is similar to web page hits and referrals. It simplified the problem by making no assumption of web services' consumer. All consumers (human or other services) could contribute to a Web service's invocation frequency. This model also make no assumption of the distribution of requests. The invocation frequency naturaly reflect the popularity of a Web service.

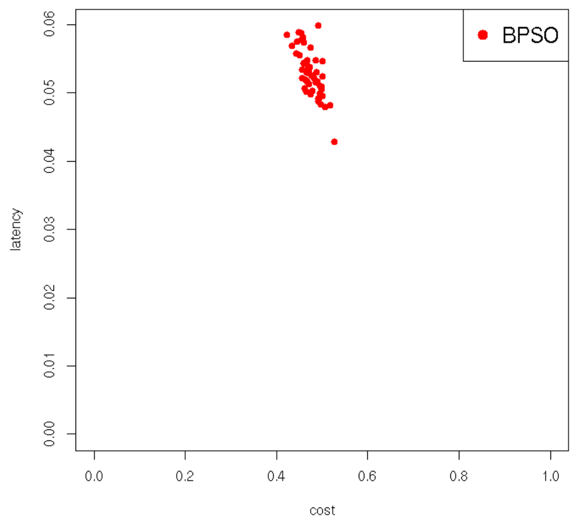
WSPs may have their specific requirements, for example, Fincial Service providers willing to make more invention in improving the QoS while Social network services provider perfer less expenditure even it means the decline of QoS. Then the BPSO might not meet their needs. As we observed in Figure 3.3, clearly, the solutions lack of diversity. This is the major drawback of using single objective algorithm to solve multi-objective problem. The linear aggregation method tries to optimize the overall fitness and ignore the objectives.



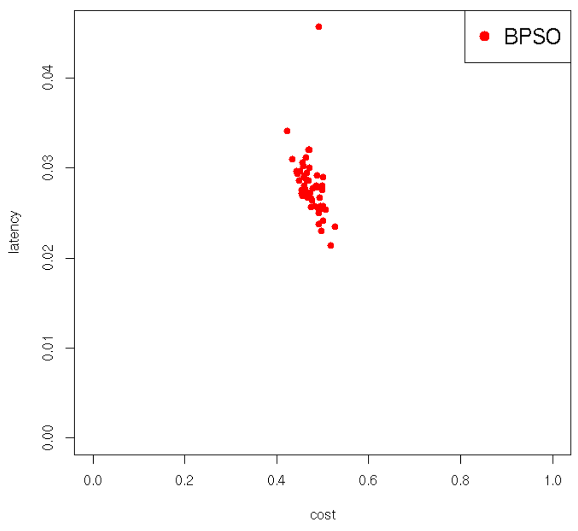
(a) Problem 1



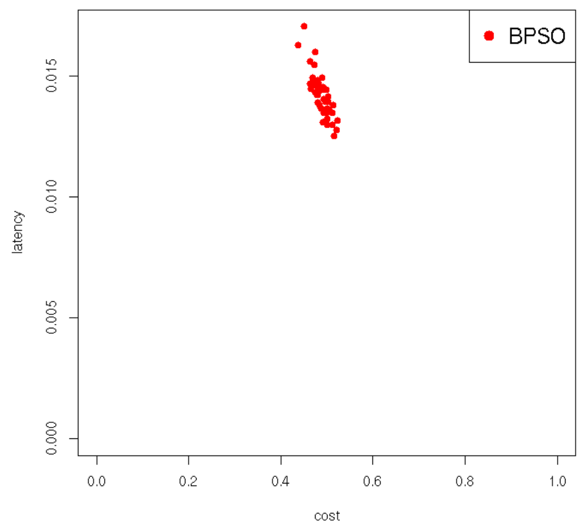
(b) Problem 2



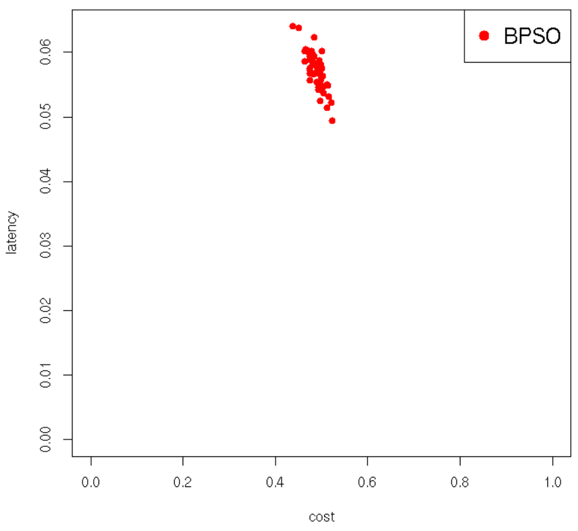
(c) Problem 3



(d) Problem 4

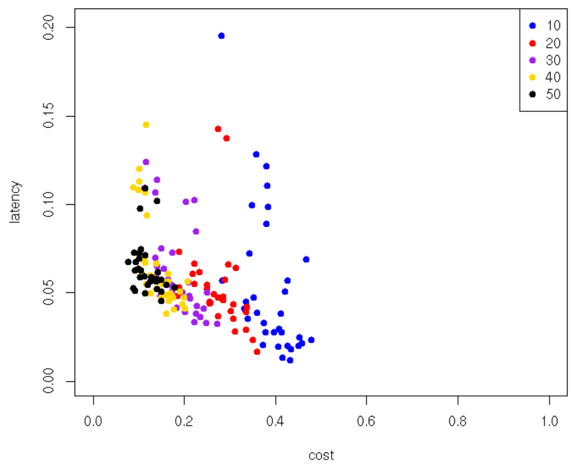


(e) Problem 5

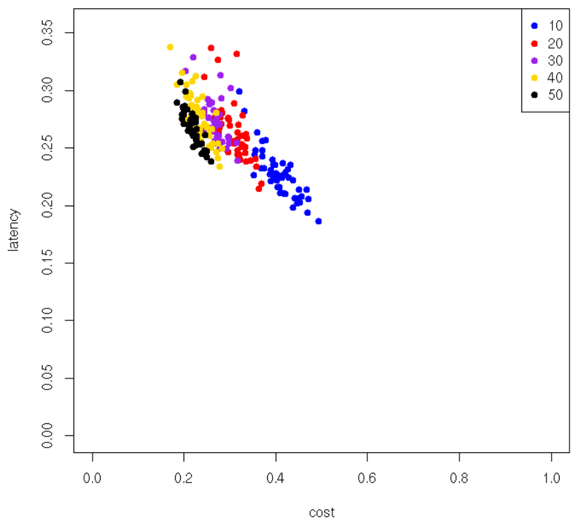


(f) Problem 6

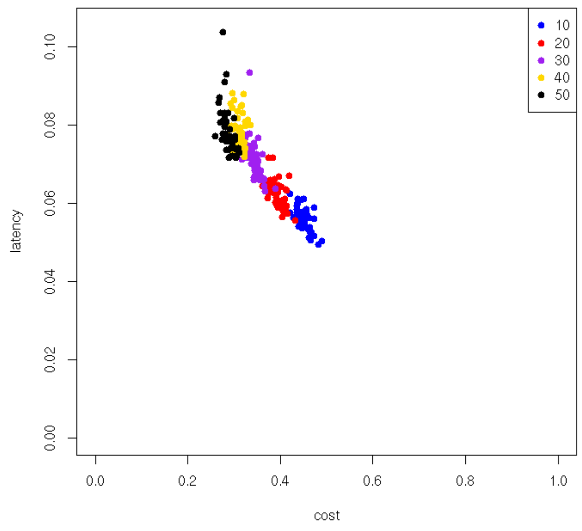
Figure 3.3: BPSO Experiments



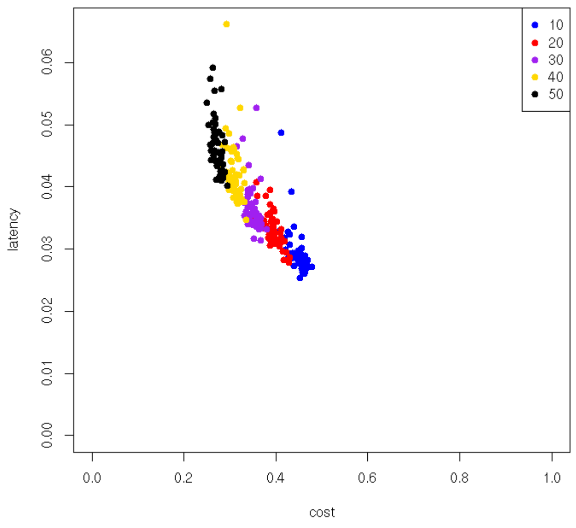
(a) Problem 1



(b) Problem 2



(c) Problem 3



(d) Problem 4

Figure 3.4: BPSO Optmization Process

3.6 Summary

This chapter first introduced a general model for Web service location-allocation problem that overcomes the drawbacks of previous research. In the meanwhile, it could be easily applied in EC algorithms. Then, it validates the model by developing a BPSO approach. The BPSO employed an linear aggregation method to combine two objectives. It uses a "death penalty" constraint handling approach. We conducted 6 experiments. The results shown BPSO adapted the model well and provided optimized solutions. However, the drawback of BPSO is that it couldn't provide a variety of choices. In the next step, we are going to multi-objective algorithms to improve the results.

Chapter 4

Multi-Objective Algorithms

4.1 Introduction

In the last chapter, we modeled the Web service location-allocation problem and developed a BPSO with linear aggregation approach. The results show BPSO provides optimized solutions with little diversity. In the chapter, we adopt two multi-objective binary EC algorithms to solve this problem.

4.2 NSGA-II for service location-allocation

NSGA-II is one of the most popular evolutionary multi-objective techniques. In this section, we investigate a NSGA-II based multi-objective approach for service location allocation. There are two version of NSGA-II, continuous and discrete. Because the Web service location allocation problem is a binary problem, we adopt the discrete version.

4.2.1 Chromosome Representation

The service location allocation matrix can be used as the representation of chromosome without making any changes.

$$A = \begin{array}{c|ccc} & j_1 & j_2 & j_3 \\ \hline s_1 & 0 & 1 & 0 \\ s_2 & 0 & 0 & 1 \\ s_3 & 1 & 1 & 0 \end{array}$$

4.2.2 Fitness Functions

NSGA-II has two fitness functions, we use the normalized objective function as the fitness functions (Equation 3.4 and 3.5).

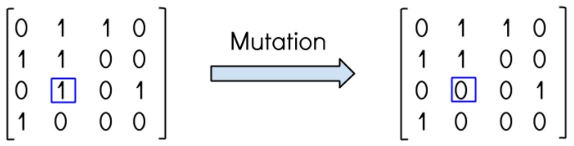
Algorithm 2 NSGA-II for Web Service Location-Allocation

```
Inputs:
Cost Matrix $C$,
Server network latency matrix $L$,
Service invocation frequency matrix $F$

Outputs: Pareto Front:a set of service allocation matrix $A$
 1: Initialize a population size of $M$;
 2: Evaluate population with fitness functions;
 3: Apply Fast Non-dominated sort and assign a ranking to each chromosome in population;
 4: Apply Tournament Selection childdren;
 5: Apply crossover and mutation over children;
 6: Apply Repair algorithm;
 7: Combine population and children to selected;
 8: while predefined generation do
 9:     Apply fast non-dominated sort on selected;
10:     Generate sets of non-dominated fronts $F = (F_1, F_2, \dots)$;
11:     Loop (inside) by adding solutions to next generation;
12:     starting from the $F_1$ until the $M$ individuals found;
13:     Evaluate the Crowding distance between points on each front;
14:     Select points (elitist) on the lower front (with lower rank) and are outside a crowding distance;
15:     Create next generation;
16:     Apply Tournament Selection;
17:     Crossover, mutation, repair algorithm and recombination;
18: end while
19: Return the chromosome in $F_1$ ;
```

4.2.3 Genetic Operators

Mutation The mutation operator works as follows: For every entry of a chromosome, randomly generate a real number *rand* from [0,1], if $rand < P_m$, then reverse the entry value. P_m is the mutation probability.



$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & \boxed{1} & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{Mutation}} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & \boxed{0} & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4.1: Mutation

CrossoverThis project adopt single point crossover. For each pair of chromosome, if a randomly generate number $rand < P_c$ where P_c is the crossover probability,then the crossover started. Randomly choose a crossover point, and exchange the pieces from chromosomes. Two offsprings are generated after this operation.

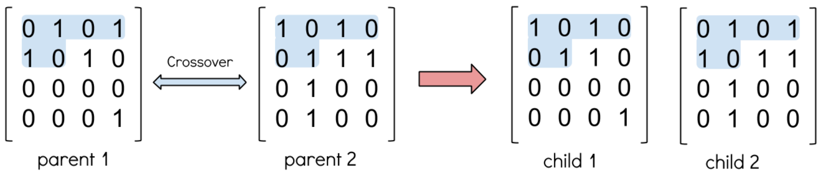


Figure 4.2: Crossover

4.2.4 Constraint Handling

In Section 3, we defined the Web service location-allocation problem with single constraint. In order to fulfill the constraint. A repair function (Algorithm 3) is developed to fix the invalid chromosomes. The function checks the Web service number constraint and fix the chromosome by randomly deploying a service to a candidate location.

Algorithm 3 Repair Algorithm

Inputs:
Population
Outputs: A repaired population

```
1: for ( do each chromosome)
2:     while  violate service number constraint do
3:         randomly choose a location $j$ and set $a_{sj} = 1$
4:     end while
```

4.3 NSPSO for service location-allocation

We applied the NSPSO developed by Li [29]. The algorithm is shown in Algorithm 4.

4.3.1 Fitness Functions

The fitness function used in the proposed algorithm is the same fitness functions with NSGA-II, defined in Section 4.2.2.

4.3.2 Particle Representation

The particle representation is the same with BPSO, a flatten binary matrxi, defined in previous Section 3.3.2.

4.3.3 Constraint Handling

The constraint handling method used in NSPSO is also “death penalty”, defined in previous Section 3.3.3.

Algorithm 4 NSPSO for Web Service Location-Allocation

Inputs:
Cost Matrix C,
Server network latency matrix L,
Service invocation frequency matrix F

Outputs: Pareto Front

```
1: Initialize the position and velocity of particles in the swarm ($Swarm$).
2: repeat
3:     Evaluate each particle with fitness functions;
4:     Identify the particles ($nonDomS$) that have non-dominated solutions in $Swarm$;
5:     Calculate crowding distance of each particle in $nonDomS$;
6:     Sort particles in $nonDomS$ based on the crowding distance;
7:     Copy all the particles in $Swarm$ to a union set ($Union$)
8:     for i = 1 to Population Size (P) do
9:         update the $pbest$ of particle $i$
10:        randomly selecting a $gbest$ for particle $i$ from the least crowded solutions in
   $nonDomS$;
11:        update the velocity and position of particle $i$;
12:        add the updated particle $i$ to $Union$ set;
13:    end for
14:    Identify different levels of nondominated fronts $F = (F_1, F_2, F_3, \dots)$ in $Union$;
15:    Empty the current $Swarm$ for the next iteration;
16:    $i = 1$;
17:    while $\mid Swarm \mid < P$ do
18:        if $\mid Swarm \mid + \mid F_i \mid \leq P$ then
19:            add $F_i$ to $Swarm$;
20:            $i = i + 1$;
21:        end if
22:        if $\mid Swarm \mid + \mid F_i \mid > P$ then
23:            calculate crowding distance in $F_i$;
24:            sort particle in $F_i$;
25:            add the ($P- \mid Swarm \mid$) least crowded particles to $Swarm$;
26:        end if
27:    end while
28: until maximum iterations is researched
29: return the positions of particles in $F_1$; =0
```

4.4 Experiment Design

We conduct 14 experiments over three algorithms: BPSO, NSGA-II and NSPSO. In order to assess and compare the quality of these solutions, hypervolume method [3] is applied. Hypervolume calculate the volume below the non-dominated curve with respective to a reference point ((1, 1) in this project). Hypervolume measures both diversity as well as convergence.

The common parameters are shown in Table 4.1. Specifically, we treat both objective equally important, so the weight parameter w in BPSO is set to 0.5. Each algorithm will be conducted 40 runs to meet the statistical minimum requirements.

Table 4.1: Parameter Settings

Method	Population	Maximum Iteration	c1	c2	w	P_m	P_c
BPSO			2	2	1	-	-
NSGA-II	50	50	-	-	-	0.2	0.8
NSPSO			2	2	1	-	-

BPSO is a single objective algorithm. So a non-dominated solutions need to be extracted from current solutions. Therefore, we first apply fast non-dominated sorting over the solutions provided by BPSO and generate a non-dominated solution set. This method was used in many researches [19, 42, 43].

We use two methodologies to compare these algorithms. 1. The best solutions. This methodology compares the best solutions from all 40 runs. It generates a best solution set by applying fast non-dominated sorting over 40 runs solutions. 2. The average solutions. This methodology apply hypervolume method on each solution. The mean and standard deviation of 40 hypervolume value will be calculated.

4.5 Experiment Results

The average solutions are shown in Table 4.2. The best solutions are shown in Figure 4.3.

In average solutions, the hypervolume of the first 10 solutions show the performance of BPSO dominate the other two algorithms. In the last four problems, NSPSO dominated the performance. In the best solutions, it shows the solutions from BPSO has better convergence in the first two problems as they are closer to the original point. As the problem size getting larger, the advantage of convergence gradually diminished. On the other hand, the diversity of BPSO is clearly much worse than NSGA-II and NSPSO as it could only covers a small region of Pareto front.

In comparison between NSGA-II and NSPSO, NSPSO has better performance in all problems except the first one. We could see the trend from Figure 4.3. In the first problem, NSGA-II dominated NSPSO in both convergence and diversity. In the problem 2, NSGA-II dominates NSPSO in most of the region except the region on the bottom right corner. In other problems, the advantage of NSPSO become clear. The diversity of solutions is the major difference between NSPSO and NSGA-II. Especially, when the data set getting very large, NSPSO could still maintain a diverse of solutions while the other algorithms could only provide some narrow solutions.

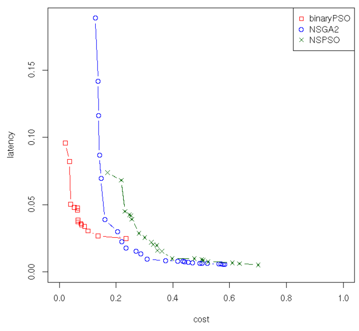
Another observation is that, there is a decreasing trend in the average solutions among all three algorithms. It indicates all three algorithms have bad scalability.

4.6 Discussion

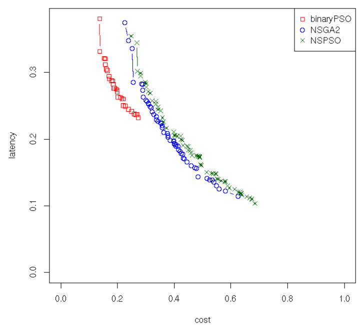
The experiments show in terms of convergence, the performance of BPSO is better than NSPSO and NSGA-II in handling small problems. In terms of diversity, NSPSO has a big advantage in most cases. There are two major problems that we are not satisfied with the current results. Firstly, the solutions are not diverse enough. From the Figure

Table 4.2: Comparison between NSPSO, NSGA-II and BPSO

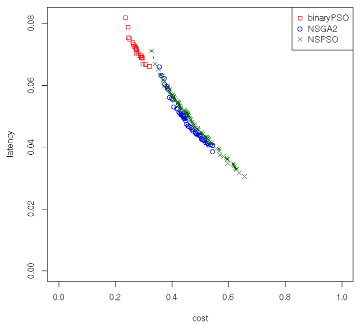
Dataset	Method	Ave-Hypervolume ± Std-Hypervolume
problem 1	NSPSO	0.76 ± 1.87E-02
	NSGA-II	0.83 ± 1.34E-02
	BPSO	0.89 ± 1.56E-02 ↑
problem 2	NSPSO	0.61 ± 1.01E-02 ↑
	NSGA-II	0.60 ± 1.08E-02
	BPSO	0.61 ↑ ± 1.23E-02
problem 3	NSPSO	0.61 ± 1.11E-02
	NSGA-II	0.59 ± 7.36E-03
	BPSO	0.69 ± 7.44E-03↑
problem 4	NSPSO	0.63 ± 1.12E-02
	NSGA-II	0.61 ± 7.45E-03
	BPSO	0.71 ± 8.00E-03↑
problem 5	NSPSO	0.61 ± 9.30E-03
	NSGA-II	0.58 ± 4.67E-03
	BPSO	0.67 ± 6.83E-03↑
problem 6	NSPSO	0.59 ± 6.96E-03
	NSGA-II	0.55 ± 6.41E-03
	BPSO	0.63 ± 5.20E-03↑
problem 7	NSPSO	0.60 ± 8.27E-03
	NSGA-II	0.56 ± 4.96E-03
	BPSO	0.63 ± 5.98E-03↑
problem 8	NSPSO	0.61 ± 7.88E-03
	NSGA-II	0.58 ± 5.06E-03
	BPSO	0.65 ± 7.79E-03↑
problem 9	NSPSO	0.60 ± 8.86E-03
	NSGA-II	0.56 ± 3.88E-03
	BPSO	0.62 ± 4.89E-03↑
problem 10	NSPSO	0.58 ± 7.87E-03
	NSGA-II	0.53 ± 4.93E-03
	BPSO	0.58 ± 4.89E-03↑
problem 11	NSPSO	0.57 ± 8.70E-03 ↑
	NSGA-II	0.52 ± 2.61E-03
	BPSO	0.55 ± 3.36E-03
problem 12	NSPSO	0.58 ± 9.44E-03 ↑
	NSGA-II	0.52 ± 3.26E-03
	BPSO	0.56 ± 3.31E-03
problem 13	NSPSO	0.59 ± 7.72E-03 ↑
	NSGA-II	0.53 ± 2.89E-03
	BPSO	0.56 ± 3.82E-03
problem 14	NSPSO	0.59 ± 8.89E-03 ↑
	NSGA-II	0.53 ± 2.49E-03
	BPSO	0.57 ± 3.03E-03



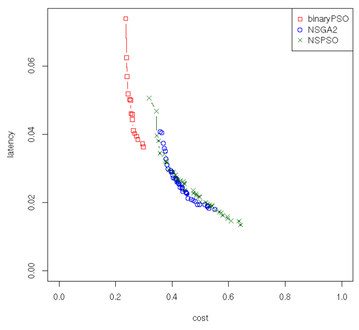
(a) Problem 1



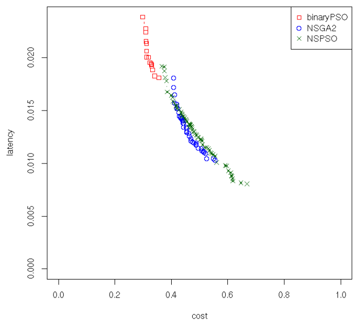
(b) Problem 2



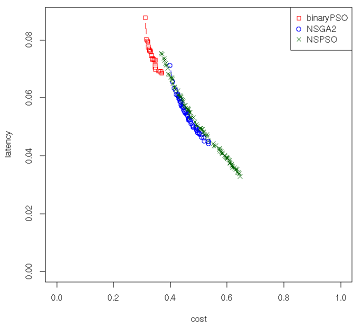
(c) Problem 3



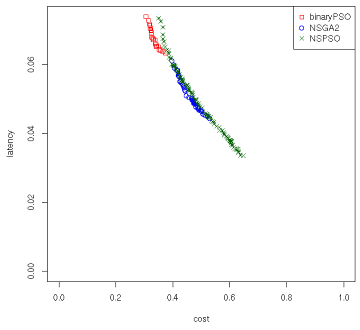
(d) Problem 4



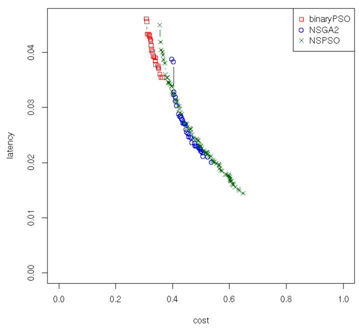
(e) Problem 5



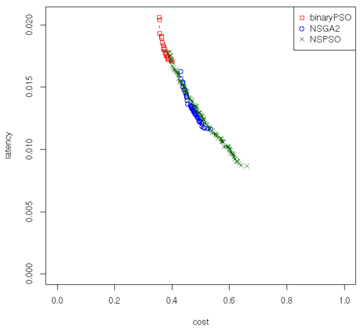
(f) Problem 6



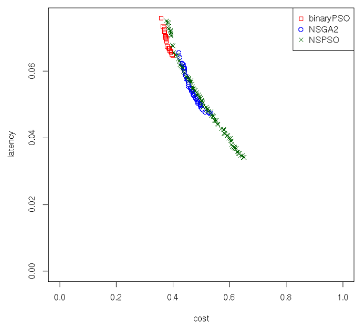
(g) Problem 7



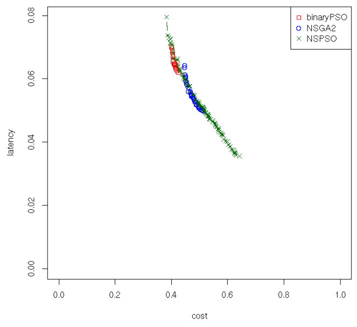
(h) Problem 8



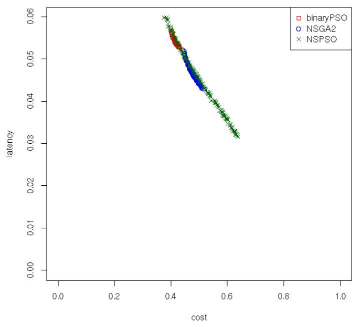
(i) Problem 9



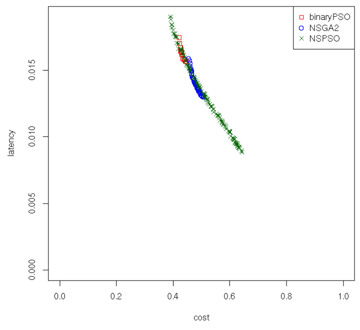
(j) Problem 10



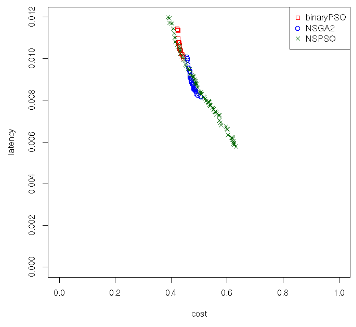
(k) Problem 11



(l) Problem 12



(m) Problem 13



(n) Problem 14

Figure 4.3: NSGA-II and NSPSO Experiments

4.3, even the best performed NSPSO could only cover half of the Pareto front. Secondly, all three algorithms are suffering from scalability issue. The scalability is major problem since the number of Web services are increasing everyday. An algorithm with low scalability would be useless when apply in the real world tasks.

4.7 Summary

In this chapter, we applied the Web service location allocation model in developing two multi-objective EC algorithms: NSGA-II and NSPSO. It shows the model is easily adapted in the multi-objective algorithms. We conducted experiments with the three algorithms over 14 problems. The results show NSPSO provides diverse solutions in most of the cases. The results also show all three algorithms has a low scalability that could not handle big dataset very well.

Chapter 5

An Improved MOPSOCD for service location-allocation

5.1 Introduction

This chapter developed an improved version of continuous multi-objective PSO with crowding distance to solve the Web service location allocation problem. We introduce a new mechanism, rounding function to the original MOPSOCD. The rounding function method, especially a dynamic rounding function is a new mechanism that makes a continuous version algorithm compatible with binary problems. This mechanism provides not only a transformation between binary and continuous representation, but also better solutions than the original version.

Raquel and et al. [35] proposed a MOPSOCD that extends the MOPSO. It has two desired features: crowding distance and mutation. These features make the algorithm has a strong ability to avoid stucking at local optima while maintain an uniformly non-dominated set.

5.2 Algoritm

The selection of *pbest* and *gbest* is one of the key step in MOPSOCD. *pbest* is the personal best solution of each particle in population. The *pbest* is updated only if the new particle dominates the current one, otherwise it remains unchanged.

In a multi-objective PSO, any non-dominated solutions in the archive can be a *gbest*, therefore, it is important to ensure the particles move to a unexplored area. The *gbest* is selected from non-dominated solutions with highest crowding distance values. It ensure the swarm move to a least crowed area.

5.2.1 Fitness Function

The fitness functions include cost fitness and latency fitness are the same fitness functions with NSGA-II, defined in Section 4.2.2.

Algorithm 5 MOPSOCD for Web Service Location-Allocation

```
Inputs:
Cost Matrix $C$,
Server network latency matrix $L$,
Service invocation frequency matrix $F$
Outputs: Pareto Front: the $Archive$ set
1: Initialize a population $P$ with random real values $\in (0, 1)$
2: Initialize velocity[i] = 0
3: Each individual $i$ in $P$ using the rounding function and then applies fitness functions
4: Initialize personal best of each individual $i$.
5: Initialize $GBEST$
6: Initialize $Archive$ with non-dominated vectors found in $P$
7: repeat
8:     Compute the crowding distance values of each non-dominated solution in the $Archive$
9:     Sort the non-dominated solutions in $Archive$ in descending crowding distance values
10:    for ( do each particle)
11:        Randomly select the global best guide for P[i] from a specified top portion of the sorted archive A and store its position to GBEST.
12:        Compute the new velocity
13:        Update its position
14:        If it goes beyond the boundaries, then multiply its velocity by -1
15:        If($t < (MAXT * PMUT)$), apply Mutation
16:        Rounding and evaluating fitness
17:        Update its $PBESTS$
18:    end for
19:    Insert new nondominated solution into $Archive$, remove dominated solutions from $Archive$
20: until maximum iterations is researched
21: return $Archive$
```

5.2.2 Particle Representation

The major difference between MOPSOCD and the other three algorithms is the particle representation. Instead of using a binary representation, we adopt a continuous representation to adapt the continuous algorithm. We also employ the flatten method as describe in the previous section.

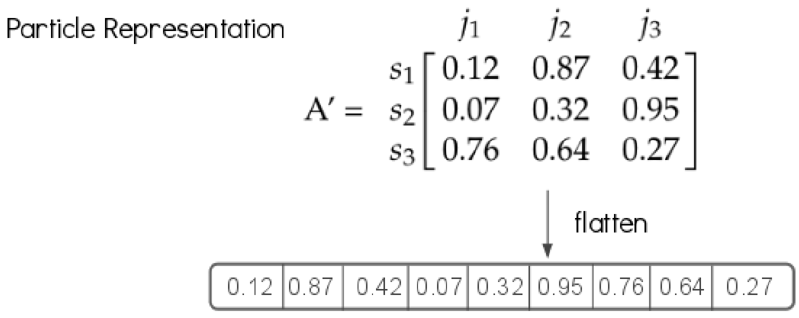

Particle Representation
$$A' = \begin{array}{c|ccc} & j_1 & j_2 & j_3 \\ s_1 & 0.12 & 0.87 & 0.42 \\ s_2 & 0.07 & 0.32 & 0.95 \\ s_3 & 0.76 & 0.64 & 0.27 \end{array}$$
flatten
0.12 0.87 0.42 0.07 0.32 0.95 0.76 0.64 0.27

Figure 5.1: Matrix Flatten

5.2.3 Constraint Handling

The constraint handling method used by MOPSOCD is a ranking of violations. A solution I is considered constraint-dominate a solution J if any of the following conditions is true:

1. solution I is feasible, solution J is not.
2. Both solutions are infeasible, solution I has less violations.
3. Both solutions are feasible, solution I dominate solution J

The particle with less violations always consider a better solution. If there is only one constraints defined, this constraint handling method provide similar effect with death penalty method.

5.2.4 Rounding Functions

The MOPSOCD is designed as a continuous version PSO. Instead of changing this algorithm to a binary version, we still use the continuous version. In order to be compatible with the binary model, the continuous representation particle needs to be transformed to a binary representation during the evaluation of fitness. That is, in the initial stage, particles are initialized in real value. The updates of velocity and position are as usual. During the evaluation of fitness, the first step is to transform the real representation particle to binary. Then, evaluate the binary particle using the fitness functions.

The rounding function is a way that used in mapping real value particle to a discrete value. The common strategy is to round a real value to its closest integer number. The round-down strategy is adopted in [28] to solve integer programming problem. [14] uses a real value representation of chromosome for GA. Then the real value chromosome is rounded to integer and binary representations in order to achieve a mixed integer optimization of array antenna pattern and micro-strip antenna. [30] adopts rounding and interval mapping strategy to solve 0-1 discrete, integer optimization and mixed optimization problem. [2] uses a random-round function which randomly returns round-up value or round-down value.

Static Rounding Function

A static rounding function is a straightforward strategy. A parameter threshold t is introduced in static rounding function. The value of a particle entry is either round up or round down according to t. The threshold value t is rather ad-hoc that based on empirical study.

$$a_{sj} = \begin{cases} 1 & \text{if } a'_{sj} > t \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

Dynamic Rounding Function

The threshold plays an important role in constraining the particle swarm in the search space. The static rounding function has the following drawbacks, firstly, the parameter threshold t needs to be defined. The value of threshold t is problem specific, therefore, it is hard to estimate the performance before obtaining the results. Secondly, the influence of different threshold values are not completely studied. Based on the above reasons, a dynamic rounding threshold is proposed. A dynamic rounding function has two steps. In the first step, it adjusts the value of threshold t according to the current generation. The second step is the same static rounding function. Three dynamic rounding functions are considered. Equation 5.2 is a linear function. Equation 5.3 is a quadratic function. Equation 5.4 is a reciprocal function. The reason why we design three dynamic functions is that we would like to compare the impact of different trajectories of threshold. t is the value of threshold, g is the current generation. The lower boundary of a threshold l, upper boundary u need to be predefined. The performance of these rounding functions is studied in the Section 5.4.1.

$$t = \frac{l-u}{\text{max generation}} g + u \tag{5.2}$$

$$t = \frac{l-u}{(\text{max generation})^2} g^2 + u \tag{5.3}$$

$$t = u - \frac{u-l}{\text{max generation} - g} (g \neq \text{max generation}) \tag{5.4}$$

5.2.5 Transfer Learning

Human have the ability to learn a technique or knowledge and apply in different fields. As the increased dimensionality of the problem, the performance of evolutionary computation drops. It is necessary to build a system that has the ability to reuse the learned knowledge. Transfer learning is a process to reuse the knowledge in solving unseen tasks [**?**]. In this section, we proposed an adaptive threshold that embodied transfer learning process.

Figure 5.7 shows the process of a transfer learning process. Initially, the threshold t is set to a upper bound u (e.g. 0.7). Then the PSO running with this setting for a pre-defined interval i (e.g. 10 generations). In the beginning of next interval (e.g. 11 generation), the threshold t is changed according to a function (Equation 5.7) and remain steadily until next interval. Repeat this process until the lower bound l is researched. The optimization may looks like forcing the swarm "jump" to a different area. But the process is equvalent to initialize a new set of population with the old one. Therefore the knowledge are inherited. A underlying assumption is that, if the particle swarm could converge within an interval i, then it is better to explore a different direction. Therefore, in the next interval, the swarm will move on and be directed by an adjacent threshold value. The potential problem of the method is that it is hard to know whether the PSO is converged. The transfer learning rounding function is shown in Equation 5.5 where t' denotes the current threshold value.

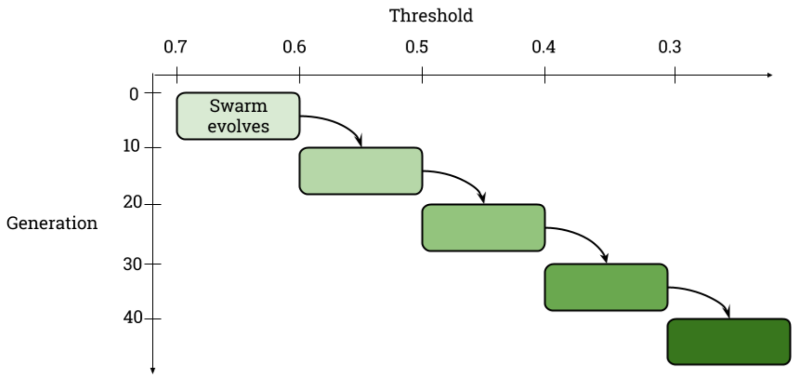


Figure 5.2: Transfer Learning Rounding Function

$$t = \begin{cases} = t' - \frac{u-l}{(\frac{\text{max generation}}{i} - 1)} & \text{if (current generation} \mod i) = 0 \\ = t' & \text{otherwise} \end{cases} \tag{5.5}$$

5.3 Experiment Design

A set of experiments have been conducted over three major features of the proposal algorithm. The first feature considers the static threshold. The influence of the selection of different values of static threshold are studied in the first experiment. The second feature considers the dynamic rounding functions. Three different types of rounding function are examined in the second experiment. The third feature is the transfer learning with an adaptive threshold. Its performance will be studied in the third experiment. An experiment of a combination of static rouding function is conducted and discussed. Lastly,we conducted an experiment considers the overall performance of a MOPSOCD with dynamic rounding function in comparison with other three algorithms: PSO, NSPSO and NSGA-II.

5.3.1 Performance Metrics

The IGD [41] is a modified version of generational distance [40, 38] as a way of estimating how far are the elements in the true Pareto front from those in the Pareto front produced by our algorithm. It calculates the sum of the distances from each point of the true non-dominated front to the nearest point of the Pareto front produced by our algorithm. The lower the IGD metric, the better the solution quality. A true Pareto front is needed when calculate the IGD value. For our problem, the true Pareto front is unknown, therefore, a approximated true Pareto front is produced by combining all the solutions produced by 4 algorithms (MOPSOCD, NSGA2, NSPSO, BPSO) and apply a non-dominated sorting over it. The approximated true Pareto front dominated all the other solutions. We are going to use hypervolume and IGD as the evaluation metrics.

5.3.2 Experiments on Rounding Functions

This section designs 4 experiments to study the effect of different types of rounding functions. 4 datasets (problem 2 $\sim$ 5) will be used, chosen from the Table 3.1.

Static Rounding Function

There are two questions that the we would like to answer with this experiment. The first question is what is the influence of threshold. The second question is how to select a static threshold.

In order to answer these two questions, a set of experiments have been conducted using different static threshold values to evaluate the performance of the proposed algorithm. The threshold value is ranged from 0.3 to 0.7. The algorithm is conducted over problem 2 to problem 5. The parameters of the algorithm are set as follows, w = 0.4, mutation probability P_m = 0.5, $c1$ = 1, $c2$ = 1, archive size is 250, population size is 50 and the max number of iteration is 50. For each experiments, the proposed algorithm has been independently run 40 times. The result is compared using the best result of each settings. To obtain the best result of 40 runs, the results of all 40 runs are combined, then a fast non-dominated sorting is applied over the combined results.

Dynamic Rounding Function

In Section 5.2.4, three dynamic rounding functions are proposed. The performance of different rounding functions is the main purpose of the experiment. One major question is which dynamic rounding function provides the best results. The parameters of the PSO algorithms are set as follow: w = 0.4, mutation probability P_m = 0.5, $c1$ = 1, $c2$ = 1, archive size is 250, population size is 50 and the max number of iteration is 50. The upper bound of dynamic threshold u = 0.7 and the lower bound of dynamic threshold l = 0.3. The results are compared using hypervolume. We use hypervolume indicator to evaluate the output of each run and the final result is the mean of 40 hypervolume values.

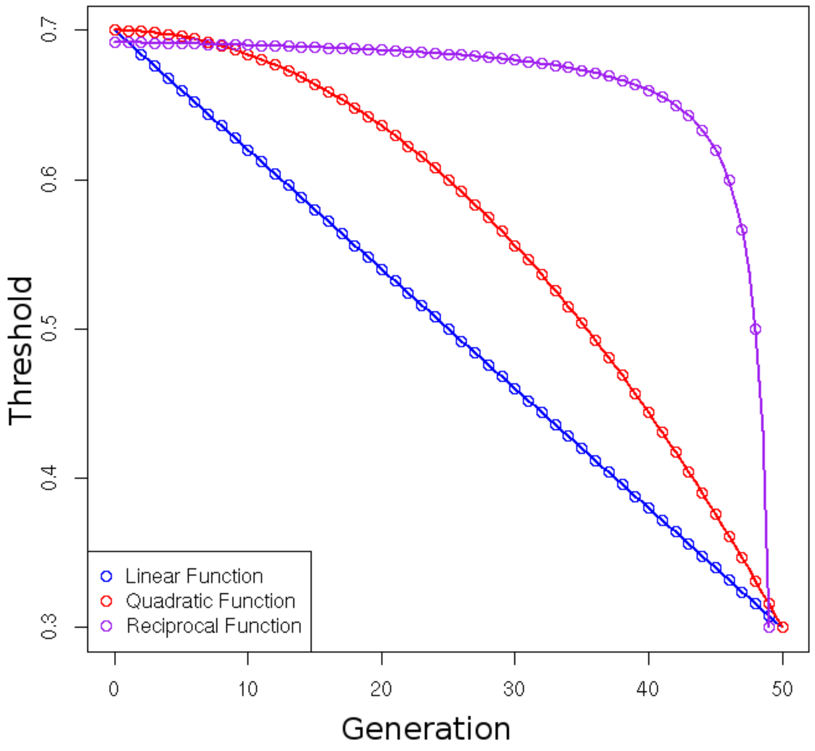


Figure 5.3: Dynamic thresholds

Figure 5.3 shows threshold t changes along with the generation. It is easy to notice that the points on linear and quadratic function are uniformly distributed. Points on reciprocal are unevenly distributed.

Transfer Learning Rounding Function

The performance of transfer learning rounding function is studied in this experiment. The parameters of the PSO algorithms are set as follow: $w = 0.4$, mutation probability $P_m = 0.5$, $c1 = 1$, $c2 = 1$, archive size is 250, population size is 50 and the max number of iteration is 50. The upper bound of dynamic threshold $u = 0.7$ and the lower bound of dynamic threshold $l = 0.3$. The interval is set to 10. The results are compared with dynamic functions using hypervolume.

A Combination of Static Rounding Function

In the Section 5.3.2, we designed an experiment to discover the effect of static threshold. A combination of static rounding funtion is an idea based on that experiment. The idea is simply obtaining the result produced by different thresholds and combine them into a dataset. Finally, employs a fast non-dominated sorting over the it. The experimental settings are the same with Section 5.3.2 and the result will be evaluate by hypervolume and compared with dynamic rounding functions.

5.3.3 The Improved MOPSOCD versus NSPSO, NSGA-II and BPSO

In this section, we compare the performance between the improved MOPSOCD with other three algorithms.

The common parameter settings are shown in Table 5.1. Specifically, we apply quadratic dynamic rounding function with [0.3, 0.7] boundaries in MOPSOCD. The archive size is 250. The results are compared using hypervolume and IGD.

Table 5.1: Parameters

Method	Population	Maximum Iteration	c1	c2	w	P_m	P_c
MOPSOCD	50	50	1	1	0.4	0.5	-
NSPSO			2	2	1	-	-
NSGA-II			-	-	-	0.2	0.8
BPSO			2	2	1	-	-

We assume both objectives are equally important, therefore, a weight for weighted-sum is set to 0.5. BPSO produces a bag of solution that is not a Pareto front. In order to compare BPSO with multi-objective algorithms, we first combine the solutions produced by 40 runs and then apply fast non-dominated sorting to obtain the Pareto front .

5.4 Experimental Results and Discussion

5.4.1 Experiments on Rounding Functions

Static Rounding Function

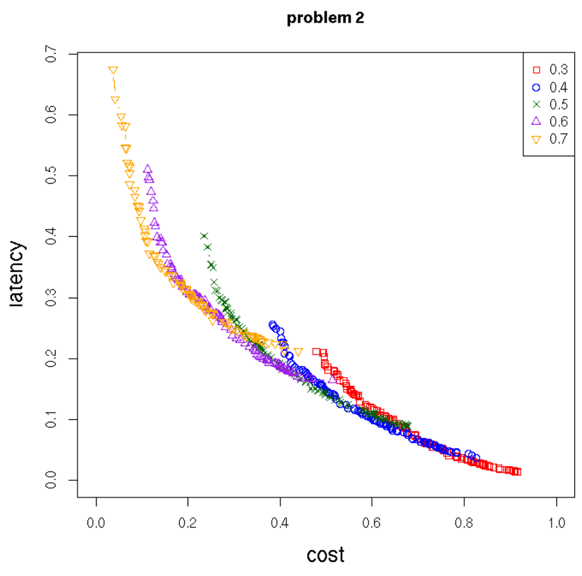
The experimental results are shown in Figure 5.4. It clearly show a pattern where the solution produced by different static threshold covers a part of Pareto front but none of them is able to cover the complete Pareto front.

The static threshold value has a huge impact of the final result. The threshold value could "clamp" the particle swarm to search a certain area. Take threshold value equals 0.7 as an example (Figure 5.5), when the rounding function rounds a real value to a binary value with respect the threshold 0.7, it means 70% of probability rounds this value to 0.

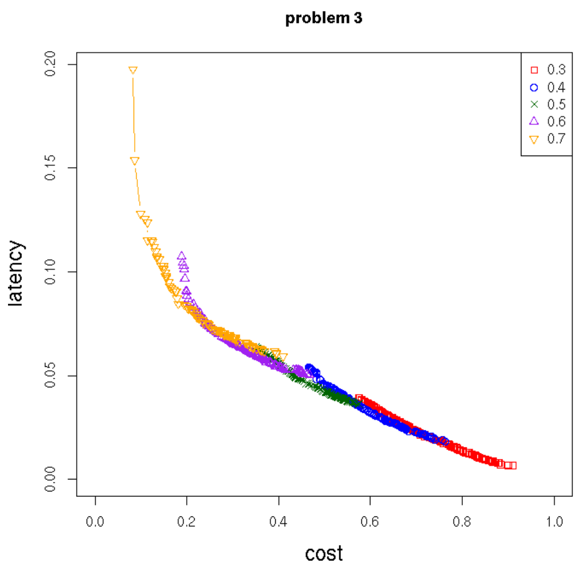
$$A' = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 0.12 & 0.87 & 0.42 \\ 0.07 & 0.32 & 0.95 \\ 0.76 & 0.64 & 0.27 \end{bmatrix} \end{array} \xrightarrow{\text{0.7 threshold}} A' = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 5.5: Rounding

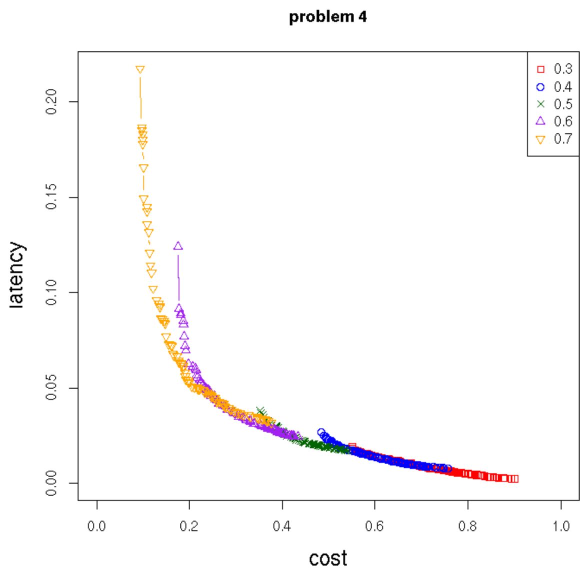
The meaning of that is 70% of probability that do not deploy the service in this candidate location. Intuitively, the optimization with threshold of 0.7 would consider optimizing cost over latency by deploying less services. The swarm are under the direction of this "savings" policy. This effect is clearly shown on Figure 5.5, where the solutions with 0.7 threshold scattered along the area with lower cost and higher latency. On the



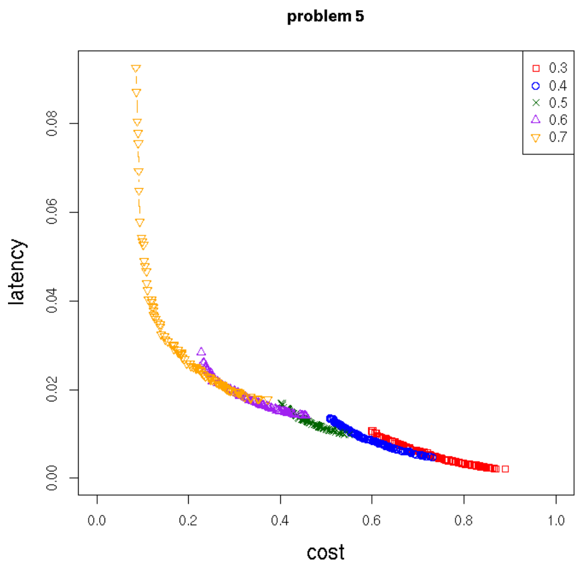
(a)



(b)



(c)



(d)

Figure 5.4: Static Rounding Function Experiments

other hand, the solutions with $t < 0.5$ thresholds are encouraged to optimize latency over cost, therefore, more services are deployed.

The experimental results clearly answered the first question: the influence of different thresholds. However, it does not offer a guide of selection of a proper threshold. Simply because none of the solutions could cover the entire Pareto front. Worth noting that, a solution of combining all results is ideal, as it largely improve the diversity of the Pareto front. This observation inspires the development of dynamic threshold.

Dynamic Rounding Function

The result is shown in Table 5.6. In this table, "Ave-", "Std-" illustrate the average and standard deviation of 3 approaches over the 40 independent runs.

Table 5.2 shows the average performance of three dynamic functions that evaluated by hypervolume. The meaning of hypervolume values denotes the volume above the non-dominated curve with respect to a reference point. Because of the task is to minimize both objectives, the larger the value, better the convergence. Convergence is one of the major measurement of multi-objective optimization. It indicates how close is the non-dominated set to the Pareto front. The results indicate the reciprocal function dominated the three dynamic functions in all test cases. This effect could be visually observed by ploting the best results (Figure 5.6). Figure 5.6 shows the reciprocal function slightly dominated the other two dynamic functions. However the problem of reciprocal function is that a gap existed in most cases. The reason of the gap is because of the ununiformity of reciprocal function (Figure 5.3).

Table 5.2: hypervolume of dynamic functions and transfer function

	Linear	Quadratic	Reciprocal
problem 2	0.72 ± 0.011	0.73 ± 0.008	$0.74 \pm 0.013 \uparrow$
problem 3	0.80 ± 0.012	0.81 ± 0.012	$0.815 \pm 0.013 \uparrow$
problem 4	0.82 ± 0.012	0.86 ± 0.016	$0.87 \pm 0.014 \uparrow$
problem 5	0.80 ± 0.014	0.85 ± 0.023	$0.86 \pm 0.020 \uparrow$

The experimental results show that in terms of convergence, reciprocal function produces the best result as it closer to the true Pareto front. However, it also shows the the major disadvantage of reciprocal function, it can not produce an entire Pareto front. In contrast, quadratic function performs a little worse in convergence but obtains an uniformly distributed non-dominated set. Linear function is dominated by quadratic function in both aspects.

The better convergence with reciprocal function can be explained, as there is minor change in threshold in the most generations, the swarm has longer time to search the same direction. On the other hand, with linear and quadratic function, the constant changing in direction could leads to premature convergence.

In comparison between quadratic function and linear function, the only difference is that changing in quadratic function is smoother than linear. The searching process is in a continuous space, therefore, sudden changes and jumps are considered to be harmful.

With these experiments results, it is still not easy to answer *Which dynamic rounding function produces the best results*. In the perspective of algorithm design, convergence

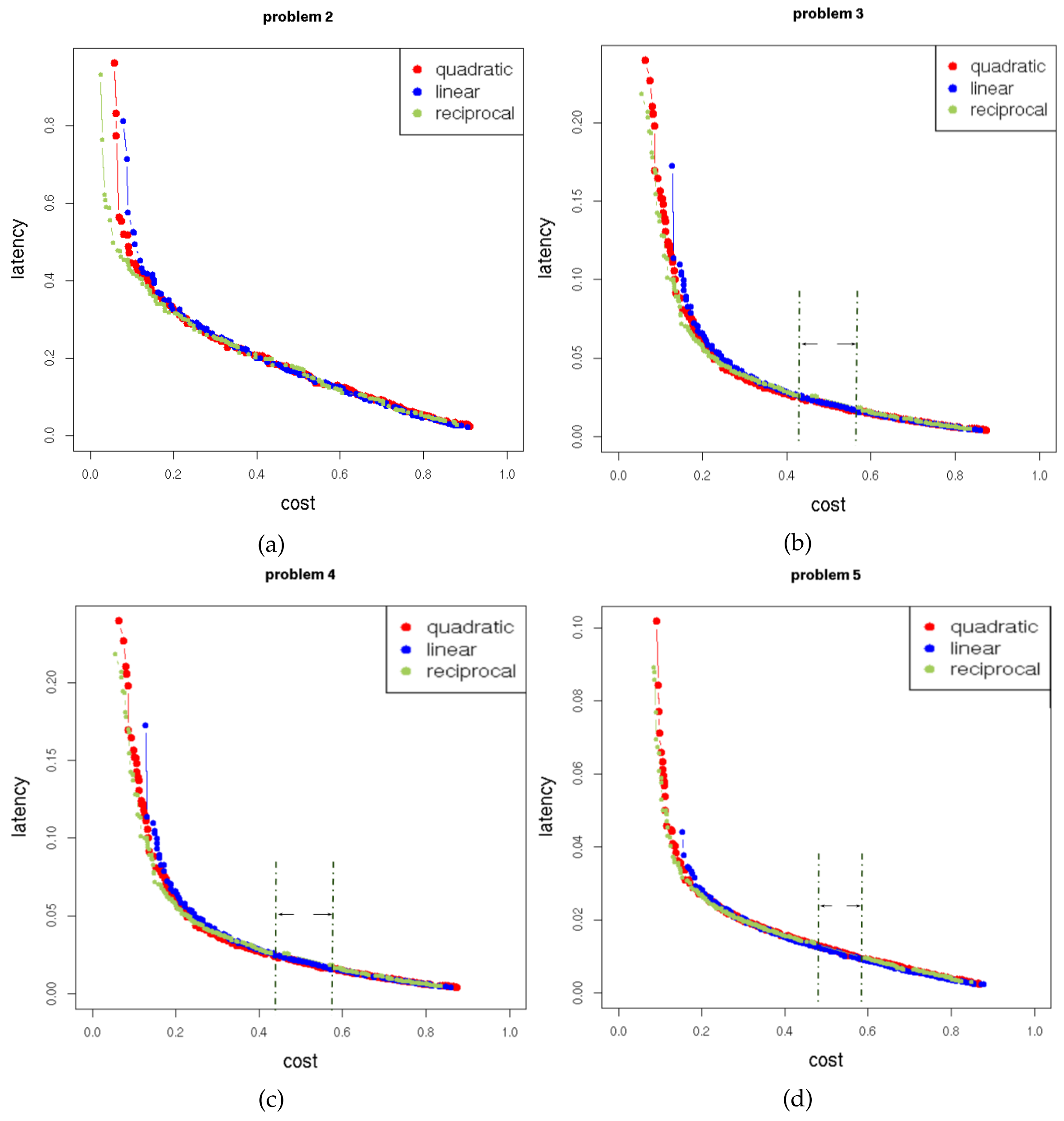


Figure 5.6: Dynamic Rounding Function Experiments

and diversity are both important. The little advantage of convergence in reciprocal function might be considered as trivial, but the gap in the Pareto front could not be neglected. Therefore, quadratic function is a better choice. On the other hand, from the perspective of Web service location allocation, the gap might be trivial since it can be complemented by the nearby solutions. However, better convergence means high quality allocation plan which is the major goal of this project.

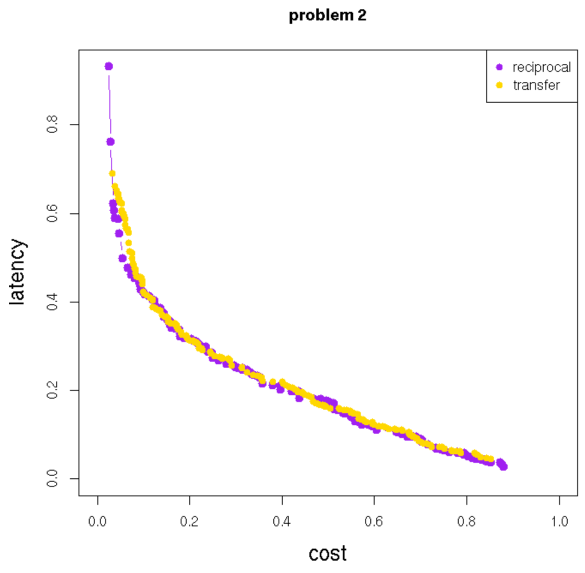
Transfer Learning with an Adaptive Threshold

We compared the performance of transfer learning with reciprocal rounding function. The Table 5.3 clearly show reciprocal function dominates transfer learning in the most cases. However, Figure 5.7 shows transfer function dominate in problem 3 and has

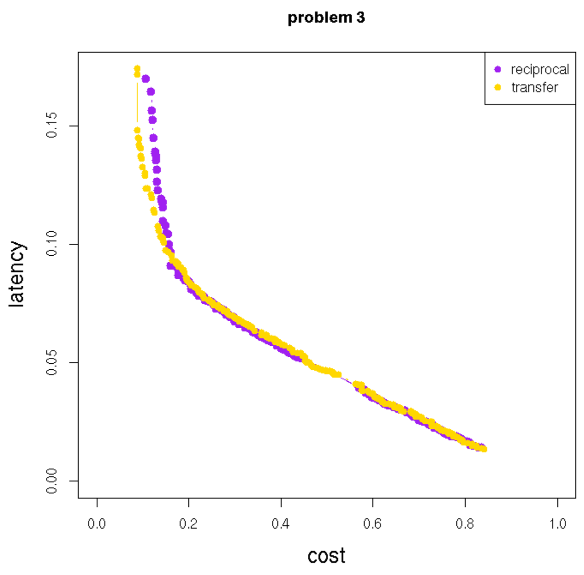
Table 5.3: Transfer Learning Experiments

	Reciprocal	Transfer
problem 2	$0.74 \pm 0.013 \uparrow$	0.72 ± 0.011
problem 3	0.815 ± 0.013	$0.83 \pm 0.014 \uparrow$
problem 4	$0.87 \pm 0.014 \uparrow$	0.85 ± 0.014
problem 5	$0.86 \pm 0.020 \uparrow$	0.85 ± 0.022

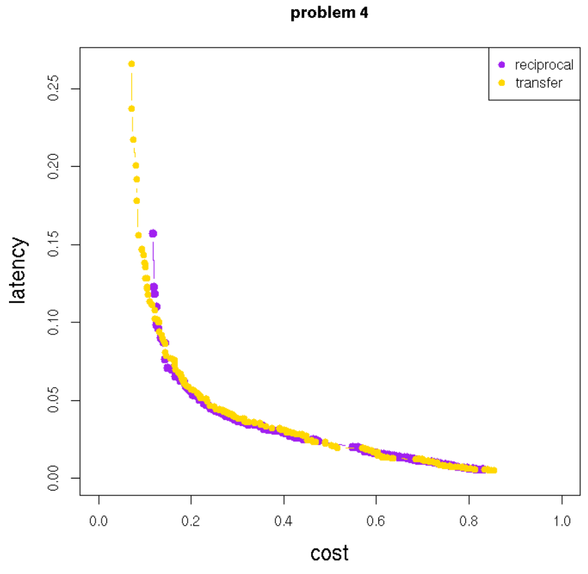
better diversity in problem 4. The results show another desired feature of transfer learning. It could provide a uniformly distributed non-dominated set. Overall, the performance of transfer learning are very closed to reciprocal function.



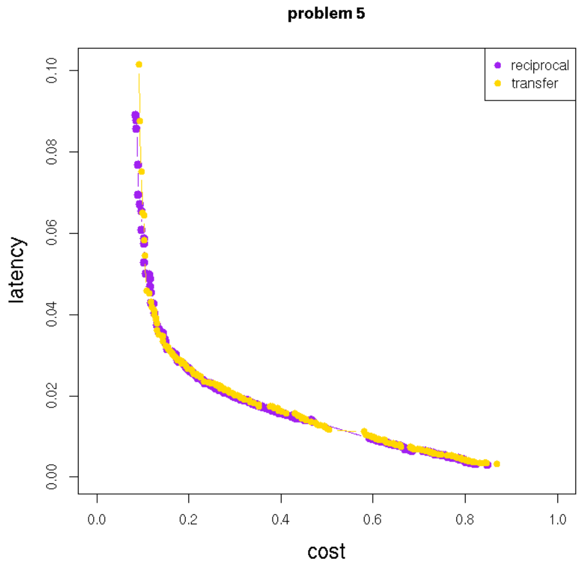
(a)



(b)



(c)

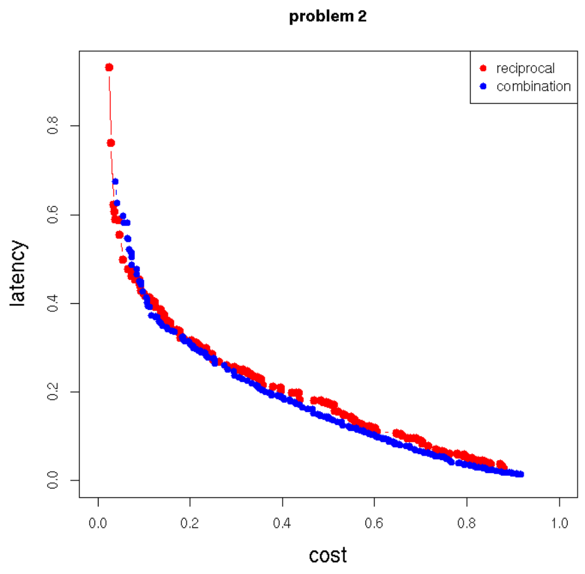


(d)

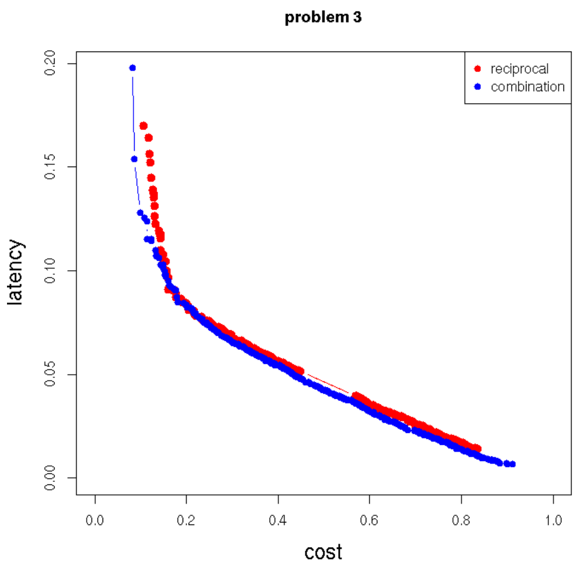
Figure 5.7: Transfer Learning Rounding Function Experiments

A Combination of Static Function

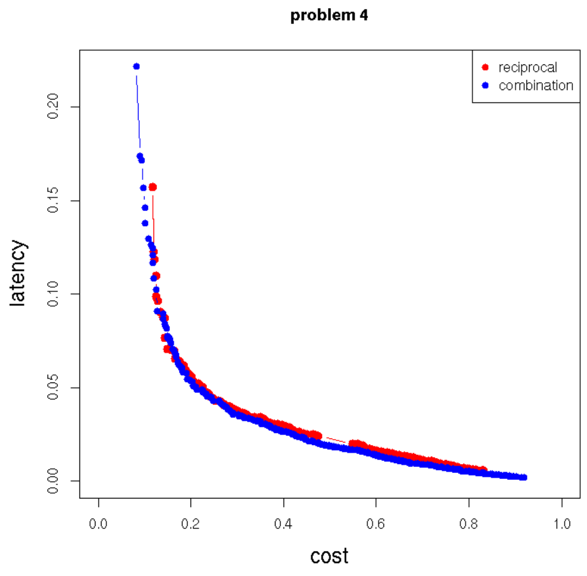
In this experiment, we combined all solutions from 5 static rounding function mentioned in Section 5.3.2 and applied a fast non-dominated sorting over it. The performance is compared with reciprocal rounding function in Figure 5.8. As the figure shows, the combined non-dominated set dominates all problems. The solution is not only diverse but also uniformaly distributed. However, the major problem is that this method takes five times longer than using reciprocal function.



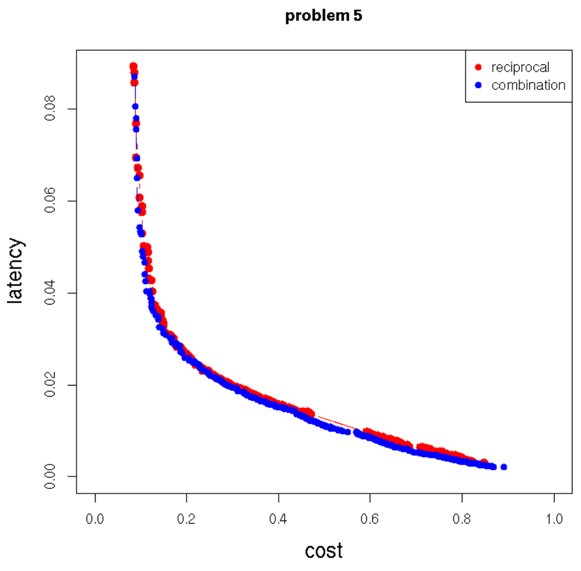
(a)



(b)



(c)



(d)

Figure 5.8: Combination of Static Function Experiments

5.4.2 An Improved MOPSOCD versus NSPSO, NSGA-II and BPSO

As can be seen from Table 5.4, on all datasets except one, MOPSOCD dominated other algorithms in both hypervolume and IGD measurement. The only exception is problem 1, where BPSO has the best hypervolume value. On all datasets, only MOPSOCD remains a good performance on hypervolume where there are obvious decreasing trends in other three algorithms along with the number of variable increases. On all dataset, MOPSOCD achieved a considerably better performance in IGD than other three algorithms which indicates a high coverage.

Compared to NSPSO and NSGA-II, MOPSOCD with dynamic rounding function achieved significantly better convergence and diversity. The first reason is that with dynamic rounding function, MOSPCOD could achieve a much better diversity. In contrast, NSGA-II and NSPSO are easy to stuck at local optima. The second reason is MOPSOCD keeps an external archive. Although three algorithms maintain a same size population, they produce different size of solutions. MOSPCOD outputs a 250 size of archive, other two algorithms output a 50 size of population.

Compared to BPSO, the convergence in problem 1 is worse than BPSO. The first reason probably is that BPSO runs 50 generations with the same weight for both objectives, it has more time to search a direction. On the other hand, with dynamic rounding function, MOSPCOD might not complete converged. The second reason is related to the variable size, where BPSO has better performance in small dataset, when the number of dataset increases, the performance drops rapidly. In contrast, MOPSOCD with dynamic rounding function does not affect by variable size, namely, good scalability.

In terms of execution time, MOPSOCD does not have much advantage as shown in Table 5.5. NSGA-II dominated in most problems except the first two problems.

5.4.3 Discussion

In this chapter, we proposed an improved continuous multi-objective PSO with crowding distance to solve the Web service location allocation problem. We major contribution is that, we provide a rounding function and a transfer learning technique make a continuous algorithm compatible with binary problem. These technique can also be applied in other continuous algorithms. The improved MOPSOCD shows 2 major desirable features. 1. The solution set has a good diversity which covers most of the Pareto front. 2. The improved MOPSOCD has a good scalability. The solution quality does not affect by the problem size.

5.4.4 Summary

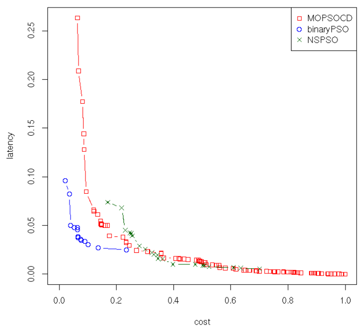
This chapter proposed an improved version of MOPSOCD to solve the Web service location-allocation problem. We introduce a rounding function technique makes the continuous algorithm compatible with binary problem. Meanwhile, a transfer learning process is embodied with an adaptive threshold. The proposed algorithm was compared with previous approaches. The results show that, on most datasets, the improved MOPSOCD can achieve much better result in both convergence and diversity than other three methods NSPSO, NSGA-II and BPSO. Additionally, we found the improved MOPSOCD has ideal scalability. The performance of MOSPCOD with dynamic rounding function does not affect by the size of variable. That is a significantly improvement.

Table 5.4: Comparison between MOPSOCD, NSPSO, NSGA-II and BPSO

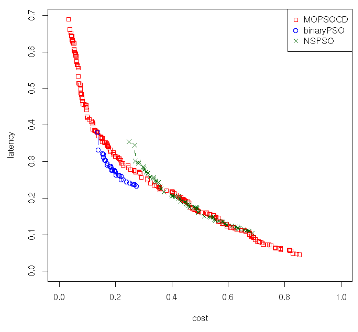
Dataset	Method	Ave-Hypervolume ± Std-Hypervolume	Ave-IGD ± Std-IGD
problem 1	MOPSOCD	0.83 ± 4.45E-02	3.73E-02 ± 1.03E-02 ↑
	NSPSO	0.76 ± 1.87E-02	0.16 ± 3.45E-02
	NSGA-II	0.83 ± 1.34E-02	0.19 ± 3.21E-02
	BPSO	0.89 ± 1.56E-02 ↑	0.46 ± 2.45E-02
problem 2	MOPSOCD	0.73 ± 1.14E-02 ↑	3.15E-02 ± 7.92E-03 ↑
	NSPSO	0.61 ± 1.01E-02	0.15 ± 1.46E-02
	NSGA-II	0.60 ± 1.08E-02	0.19 ± 1.81E-02
	BPSO	0.61 ± 1.23E-02	0.42 ± 1.54E-02
problem 3	MOPSOCD	0.81 ± 1.26E-02 ↑	7.03E-03 ± 1.92E-03 ↑
	NSPSO	0.61 ± 1.11E-02	0.10 ± 6.35E-03
	NSGA-II	0.59 ± 7.36E-03	0.16 ± 7.25E-03
	BPSO	0.69 ± 7.44E-03	0.30 ± 8.94E-03
problem 4	MOPSOCD	0.83 ± 1.16E-02 ↑	5.80E-03 ± 1.37E-03 ↑
	NSPSO	0.63 ± 1.12E-02	0.11 ± 8.55E-03
	NSGA-II	0.61 ± 7.45E-03	0.17 ± 9.11E-03
	BPSO	0.71 ± 8.00E-03	0.30 ± 8.62E-03
problem 5	MOPSOCD	0.84 ± 1.39E-02 ↑	3.74E-03 ± 1.02E-03 ↑
	NSPSO	0.61 ± 9.30E-03	0.11 ± 7.93E-03
	NSGA-II	0.58 ± 4.67E-03	0.17 ± 6.89E-03
	BPSO	0.67 ± 6.83E-03	0.24 ± 5.02E-03
problem 6	MOPSOCD	0.81 ± 1.40E-02 ↑	5.81E-03 ± 1.95E-03 ↑
	NSPSO	0.59 ± 6.96E-03	0.11 ± 5.06E-03
	NSGA-II	0.55 ± 6.41E-03	0.18 ± 8.01E-03
	BPSO	0.63 ± 5.20E-03	0.28 ± 5.88E-03
problem 7	MOPSOCD	0.79 ± 1.50E-02 ↑	7.13E-03 ± 1.70E-03 ↑
	NSPSO	0.60 ± 8.27E-03	0.10 ± 4.58E-03
	NSGA-II	0.56 ± 4.96E-03	0.17 ± 6.48E-03
	BPSO	0.63 ± 5.98E-03	0.27 ± 7.92E-03
problem 8	MOPSOCD	0.81 ± 1.54E-02 ↑	8.77E-03 ± 1.90E-03 ↑
	NSPSO	0.61 ± 7.88E-03	0.12 ± 5.81E-03
	NSGA-II	0.58 ± 5.06E-03	0.19 ± 6.17E-03
	BPSO	0.65 ± 7.79E-03	0.27 ± 5.86E-03
problem 9	MOPSOCD	0.83 ± 1.62E-02 ↑	3.58E-03 ± 1.53E-03 ↑
	NSPSO	0.60 ± 8.86E-03	0.11 ± 5.33E-03
	NSGA-II	0.56 ± 3.88E-03	0.17 ± 3.56E-03
	BPSO	0.62 ± 4.89E-03	0.22 ± 3.22E-03
problem 10	MOPSOCD	0.80 ± 1.28E-02 ↑	4.30E-03 ± 1.75E-03 ↑
	NSPSO	0.58 ± 7.87E-03	0.11 ± 4.97E-03
	NSGA-II	0.53 ± 4.93E-03	0.19 ± 5.62E-03
	BPSO	0.58 ± 4.89E-03	0.25 ± 3.91E-03
problem 11	MOPSOCD	0.79 ± 1.15E-02 ↑	5.19E-03 ± 1.81E-03 ↑
	NSPSO	0.57 ± 8.70E-03	0.12 ± 4.22E-03
	NSGA-II	0.52 ± 2.61E-03	0.20 ± 2.74E-03
	BPSO	0.55 ± 3.36E-03	0.24 ± 3.36E-03
problem 12	MOPSOCD	0.80 ± 1.02E-02 ↑	3.12E-03 ± 6.71E-04 ↑
	NSPSO	0.58 ± 9.44E-03	0.12± 4.15E-03
	NSGA-II	0.52 ± 3.26E-03	0.20 ± 3.08E-03
	BPSO	0.56 ± 3.31E-03	0.24 ± 2.50E-03
problem 13	MOPSOCD	0.83 ± 1.19E-02 ↑	2.63E-03 ± 6.73E-04 ↑
	NSPSO	0.59 ± 7.72E-03	0.12 ± 3.46E-03
	NSGA-II	0.53 ± 2.89E-03	0.20 ± 3.04E-03
	BPSO	0.56 ± 3.82E-03	0.22 ± 2.01E-03
problem 14	MOPSOCD	0.84 ± 1.50E-02 ↑	3.66E-03 ± 1.75E-03 ↑
	NSPSO	0.59 ± 8.89E-03	0.13 ± 3.85E-03
	NSGA-II	0.53 ± 2.49E-03	0.22 ± 3.24E-03
	BPSO	0.57 ± 3.03E-03	0.25 ± 1.83E-03

Table 5.5: Execution time

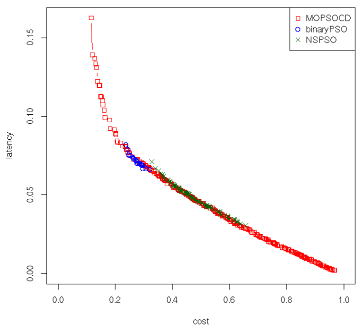
	method	Ave-time ± Std-time
problem 1	BPSO	17.99 ± 0.26
	MOPSOCD	12.98 ± 0.18 ↑
	NSPSO	19.00 ± 0.17
	NSGA-II	15.35 ± 0.15
problem 2	BPSO	23.55 ± 0.27
	MOPSOCD	16.18 ± 0.26 ↑
	NSPSO	25.52 ± 0.27
	NSGA-II	15.38 ± 0.31
problem 3	BPSO	103.65 ± 1.87
	MOPSOCD	94.98 ± 7.28
	NSPSO	111.86 ± 1.11
	NSGA-II	74.34 ± 0.61 ↑
problem 4	BPSO	181.20 ± 4.40
	MOPSOCD	175.99 ± 9.67
	NSPSO	182.09 ± 1.86
	NSGA-II	147.98 ± 1.30 ↑
problem 5	BPSO	137.03 ± 0.87
	MOPSOCD	89.74 ± 8.53
	NSPSO	161.31 ± 0.95
	NSGA-II	84.17 ± 1.03 ↑
problem 6	BPSO	208.63 ± 2.23
	MOPSOCD	172.80 ± 7.68
	NSPSO	236.23 ± 2.72
	NSGA-II	157.52± 1.62 ↑
problem 7	BPSO	234.73 ± 6.42
	MOPSOCD	202.68 ± 10.46
	NSPSO	242.94 ± 9.00
	NSGA-II	159.26 ± 1.31 ↑
problem 8	BPSO	476.76 ± 22.40
	MOPSOCD	531.64 ± 43.14
	NSPSO	444.41 ± 22.86
	NSGA-II	375.05 ± 4.11 ↑
problem 9	BPSO	293.43 ± 3.01
	MOPSOCD	198.81 ± 7.11
	NSPSO	334.62 ± 2.81
	NSGA-II	181.30 ± 1.99 ↑
problem 10	BPSO	507.72 ± 4.19
	MOPSOCD	449.91 ± 26.00
	NSPSO	539.51 ± 4.06
	NSGA-II	381.18 ± 3.06 ↑
problem 11	BPSO	1,237.30 ± 42.06
	MOPSOCD	1,262.79 ± 91.65
	NSPSO	1,328.17 ± 12.67
	NSGA-II	1,036.53 ± 35.38 ↑
problem 12	BPSO	3,631.14 ± 17.70
	MOPSOCD	4,326.22 ± 478.14
	NSPSO	3,395.47 ± 100.51
	NSGA-II	3,326.94 ± 38.21 ↑
problem 13	BPSO	1,416.63 ± 0.26
	MOPSOCD	1,155.21 ± 28.85
	NSPSO	1,507.92 ± 25.74
	NSGA-II	1,098.08 ± 17.36 ↑
problem 14	BPSO	3,617.53 ± 34.13
	MOPSOCD	3,284.66 ± 124.13
	NSPSO	3,759.51± 61.49
	NSGA-II	3,372.53 ± 31.05 ↑



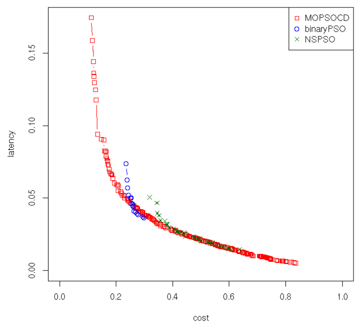
(a) Problem 1



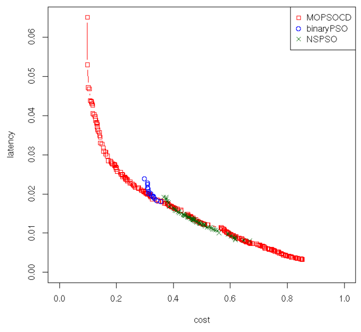
(b) Problem 2



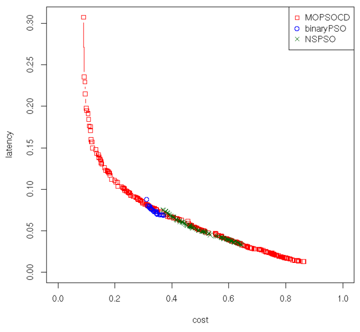
(c) Problem 3



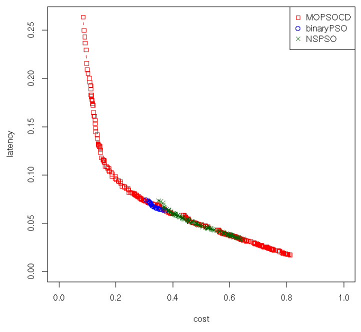
(d) Problem 4



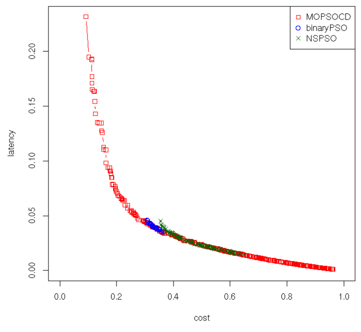
(e) Problem 5



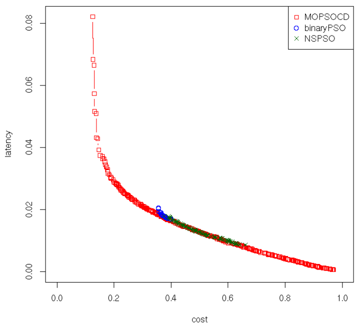
(f) Problem 6



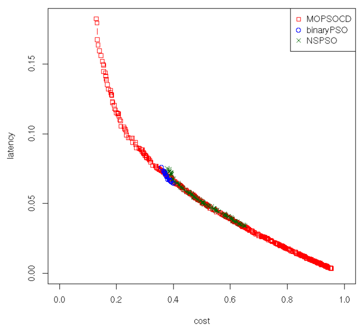
(g) Problem 7



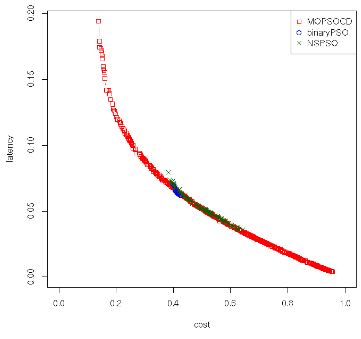
(h) Problem 8



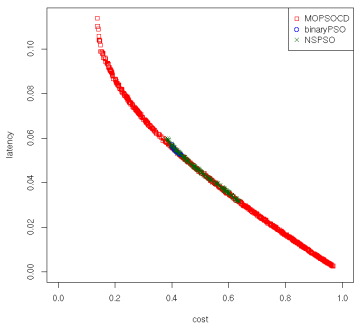
(i) Problem 9



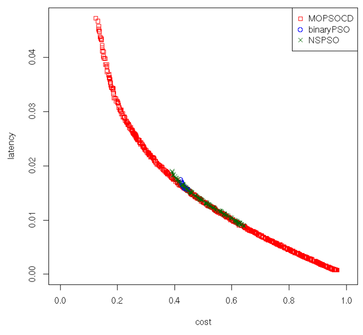
(j) Problem 10



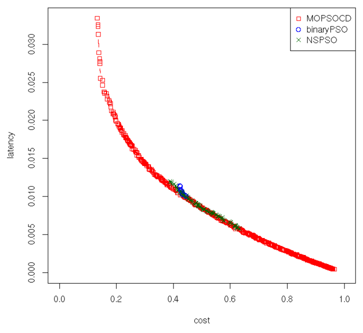
(k) Problem 11



(l) Problem 12



(m) Problem 13



(n) Problem 14

Figure 5.9: MOSPCOD, NSPSO and BPSO Experiments

Chapter 6

Conclusions

The aim of this project is to model the Web service location allocation problem. It achieves this goal by modeling the input and output data to a matrix representation. This model overcomes the drawbacks of the model proposed in previous research. In addition, this model can be eaily adapted to various EC algorithms. Then, it validates the model by developing a BPSO approach. The BPSO employed an linear aggregation method to combine two objectives. It uses a death penalty constraint handling approach. We conducted 6 experiments. The results shown BPSO adapted the model well and provided optimized solutions.

We applied the Web service location allocation model in developing two multi-objective EC algorithms: NSGA-II and NSPSO. It shows the model is easily adapted in the multi-objective algorithms. We conducted experiments with the three algorithms over 14 problems. The results show NSPSO provides diverse solutions in most of the cases. The results also show all three algorithms has a low scalability that could not handle big dataset very well.

We achieved the second objective by developing a single objective to develop a multi-objective PSO based approach to the Web service location-allocation problem.

Lastly, we proposed an improved version of MOPSOCD to solve the Web service location-allocation problem. We introduce a rounding function technique makes the continuous algorithm compatible with binary problem. Meanwhile, a transfer learning process is embodied with an adaptive threshold. The proposed algorithm was compared with previous approaches. The results show that, on most datasets, the improved MOPSOCD can achieve much better result in both convergence and diversity than other three methods NSPSO, NSGA-II and BPSO. Additionally, we found the improved MOPSOCD has ideal scalability. The performance of MOSPCOD with dynamic rounding function does not affect by the size of variable. That is a significantly improvement.

Bibliography

[1] ABOOLIAN, R., SUN, Y., AND KOEHLER, G. J. A locationallocation problem for a web services provider in a competitive market. *European Journal of Operational Research 194*, 1 (2009), 64 – 77.

[2] ANGHINOLFI, D., AND PAOLUCCI, M. A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. *European Journal of Operational Research 193*, 1 (2009), 73 – 85.

[3] AUGER, A., BADER, J., BROCKHOFF, D., AND ZITZLER, E. Theory of the hypervolume indicator: Optimal μ-distributions and the choice of the reference point. In *Proceedings of the Tenth ACM SIGEVO Workshop on Foundations of Genetic Algorithms* (New York, NY, USA, 2009), FOGA '09, ACM, pp. 87–102.

[4] BALLANI, H., COSTA, P., KARAGIANNIS, T., AND ROWSTRON, A. Towards predictable datacenter networks. ACM SIGCOMM.

[5] CARAMIA, M. Multi-objective optimization. In *Multi-objective Management in Freight Logistics*. Springer London, 2008, pp. 11–36.

[6] COELLO, C., PULIDO, G., AND LECHUGA, M. Handling multiple objectives with particle swarm optimization. *Evolutionary Computation, IEEE Transactions on 8*, 3 (June 2004), 256–279.

[7] COELLO, C. A. C. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering 191*, 11 (2002), 1245–1287.

[8] COELLO COELLO COELLO, C., AND TOSCANO PULIDO, G. A micro-genetic algorithm for multiobjective optimization. In *Evolutionary Multi-Criterion Optimization*, E. Zitzler, L. Thiele, K. Deb, C. Coello Coello, and D. Corne, Eds., vol. 1993 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2001, pp. 126–140.

[9] DAN, A., JOHNSON, R. D., AND CARRATO, T. Soa service reuse by design. In *Proceedings of the 2Nd International Workshop on Systems Development in SOA Environments* (New York, NY, USA, 2008), SDSOA '08, ACM, pp. 25–28.

[10] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on 6*, 2 (2002), 182–197.

[11] DESAI, S., BAHADURE, S., KAZI, F., AND SINGH, N. Article: Multi-objective constrained optimization using discrete mechanics and nsga-ii approach. *International Journal of Computer Applications 57*, 20 (2012), 14–20. Full text available.

[12] GUO, C., LU, G., WANG, H., YANG, S., KONG, C., SUN, P., WU, W., AND ZHANG, Y. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In *ACM CONEXT 2010* (November 2010), Association for Computing Machinery, Inc.

[13] GUZEK, M., BOUVRY, P., AND TALBI, E.-G. A survey of evolutionary computation for resource management of processing in cloud computing [review article]. *Computational Intelligence Magazine, IEEE 10*, 2 (May 2015), 53–67.

[14] HAUPT, R. L. Antenna design with a mixed integer genetic algorithm. *Antennas and Propagation, IEEE Transactions on 55*, 3 (March 2007), 577–582.

[15] HE, K., FISHER, A., WANG, L., GEMBER, A., AKELLA, A., AND RISTENPART, T. Next stop, the cloud: Understanding modern web service deployment in ec2 and azure. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (New York, NY, USA, 2013), IMC '13, ACM, pp. 177–190.

[16] HUANG, H., MA, H., AND ZHANG, M. An enhanced genetic algorithm for web service location-allocation. In *Database and Expert Systems Applications*, H. Decker, L. Lhotsk, S. Link, M. Spies, and R. Wagner, Eds., vol. 8645 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 223–230.

[17] HUANG, X. Usageqos: Estimating the qos of web services through online user communities. *ACM Trans. Web 8*, 1 (Dec. 2013), 1:1–1:31.

[18] HUFFAKER, B., FOMENKOV, M., PLUMMER, D., MOORE, D., AND CLAFFY, K. Distance Metrics in the Internet. In *IEEE International Telecommunications Symposium (ITS)* (Brazil, Sep 2002), IEEE, pp. 200–202.

[19] ISHIBUCHI, H., NOJIMA, Y., AND DOI, T. Comparison between single-objective and multi-objective genetic algorithms: Performance comparison and performance measures. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on* (2006), pp. 1143–1150.

[20] JAMIN, S., JIN, C., KURC, A., RAZ, D., AND SHAVITT, Y. Constrained mirror placement on the internet. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2001), vol. 1, pp. 31–40 vol.1.

[21] JOHANSSON, J. M. On the impact of network latency on distributed systems design. *Inf. Technol. and Management 1*, 3 (2000), 183–194.

[22] KANAGARAJAN, D., KARTHIKEYAN, R., PALANIKUMAR, K., AND DAVIM, J. Optimization of electrical discharge machining characteristics of wc/co composites using non-dominated sorting genetic algorithm (nsga-ii). *The International Journal of Advanced Manufacturing Technology 36*, 11-12 (2008), 1124–1132.

[23] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on* (Nov 1995), vol. 4, pp. 1942–1948 vol.4.

[24] KENNEDY, J., AND EBERHART, R. A discrete binary version of the particle swarm algorithm. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on* (Oct 1997), vol. 5, pp. 4104–4108 vol.5.

[25] KESSACI, Y., MELAB, N., AND TALBI, E.-G. A pareto-based genetic algorithm for optimized assignment of vm requests on a cloud brokering environment. In *Evolutionary Computation (CEC), 2013 IEEE Congress on* (June 2013), pp. 2496–2503.

[26] KNOWLES, J. D., AND CORNE, D. W. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary computation 8*, 2 (2000), 149–172.

[27] LARUMBE, F., AND SANSO, B. Optimal location of data centers and software components in cloud computing network design. In *Cluster, Cloud and Grid Computing*

(CCGrid), 2012 12th IEEE/ACM International Symposium on (May 2012), pp. 841–844.

[28] LASKARI, E., PARSOPOULOS, K., AND VRAHATIS, M. Particle swarm optimization for integer programming. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on* (2002), vol. 2, pp. 1582–1587.

[29] LI, X. A non-dominated sorting particle swarm optimizer for multiobjective optimization. In *Genetic and Evolutionary Computation GECCO 2003*, E. Cant-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U.-M. OReilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska, and J. Miller, Eds., vol. 2723 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 37–48.

[30] LIU, D., FENG, Q., AND WANG, W.-B. Discrete optimization problems of linear array synthesis by using real number particle swarm optimization. *Progress In Electromagnetics Research 133* (2013), 407–424.

[31] ORGANIZATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, Apr. 2007.

[32] PAPAZOGLOU, M. P., AND HEUVEL, W.-J. Service oriented architectures: Approaches, technologies and research issues. *The VLDB Journal 16*, 3 (July 2007), 389–415.

[33] PHAN, D. H., SUZUKI, J., CARROLL, R., BALASUBRAMANIAM, S., DONNELLY, W., AND BOTVICH, D. Evolutionary multiobjective optimization for green clouds. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation* (New York, NY, USA, 2012), GECCO '12, ACM, pp. 19–26.

[34] RAN, S. A model for web services discovery with QoS. *SIGecom Exch. 4*, 1 (2003), 1–10.

[35] RAQUEL, C. R., AND NAVAL, JR., P. C. An effective use of crowding distance in multiobjective particle swarm optimization. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation* (New York, NY, USA, 2005), GECCO '05, ACM, pp. 257–264.

[36] SUN, Y. *A Location Model for Web Services Intermediaries*. PhD thesis, Gainesville, FL, USA, 2003. AAI3120151.

[37] SUN, Y., AND KOEHLER, G. J. A location model for a web service intermediary. *Decis. Support Syst. 42*, 1 (2006), 221–236.

[38] VAN VELDHUIZEN, D., AND LAMONT, G. On measuring multiobjective evolutionary algorithm performance. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on* (2000), vol. 1, pp. 204–211 vol.1.

[39] VANROMPAY, Y., RIGOLE, P., AND BERBERS, Y. Genetic algorithm-based optimization of service composition and deployment. In *Proceedings of the 3rd International Workshop on Services Integration in Pervasive Environments* (2008), SIPE '08, ACM, pp. 13–18.

[40] VELDHUIZEN, D. A. V., AND VELDHUIZEN, D. A. V. Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations. Tech. rep., Evolutionary Computation, 1999.

[41] VILLALOBOS-ARIAS, M., PULIDO, G., AND COELLO, C. A proposal to use stripes to maintain diversity in a multi-objective particle swarm optimizer. In *Swarm*

Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE (June 2005), pp. 22–29.

[42] XUE, B., ZHANG, M., AND BROWNE, W. Particle swarm optimization for feature selection in classification: A multi-objective approach. *Cybernetics, IEEE Transactions on 43*, 6 (Dec 2013), 1656–1671.

[43] XUE, B., ZHANG, M., AND BROWNE, W. N. Multi-objective particle swarm optimisation (pso) for feature selection. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation* (2012), GECCO '12, ACM, pp. 81–88.

[44] ZHANG, Y., ZHENG, Z., AND LYU, M. Exploring latent features for memory-based QoS prediction in cloud computing. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on* (2011), pp. 1–10.

[45] ZHENG, Z., ZHANG, Y., AND LYU, M. Distributed QoS evaluation for real-world web services. In *Web Services (ICWS), 2010 IEEE International Conference on* (2010), pp. 83–90.

[46] ZHOU, J., AND NIEMELA, E. Toward semantic QoS aware web services: Issues, related studies and experience. In *Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on* (2006), pp. 553–557.