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

Evolutionary Multi-Objective Optimization for Web Service Location Allocation

Boxiong Tan

Supervisors: Hui Ma, Yi Mei, Mengjie Zhang

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

Abstract

With the ever increasing number of functional similar web services being available on the Internet, the market competition is becoming intense. Web service providers realized that good Quality of Service (QoS) is a key of business success and low network latency is a critical measurement of good QoS. Because network latency is related to geometric location, a straightforward way to reduce network latency is to allocate services to proper locations. Hence, it is necessary to provide an effective web services allocation algorithm to WSPs. In this project, we model the Web service location allocation problem as a multi-objective optimization problem - minimizing overall network latency and total cost. We use two PSO-based algorithms with different strategies - an aggregating strategy and a Pareto front strategy to solve the problem. We further develop a new PSO-based algorithm with rounding function approach to improve the quality of solutions. The result shows that the new algorithm could provide diverse solutions. In addition, the new algorithm has good performance regardless of increasing problem size.

Acknowledgments

I would like to thank my three supervisors, Hui Ma, Yi Mei and Mengjie Zhang for their thoughtful supervision and guidance; my family and friends for their encouragement and support throughout the years.

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 start with 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 [48].

The arising of web service 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) [16]. 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 [50].

A web service is a software system allowing to expose web services via the Internet. The SOA promotes the composition of coarse-grained web services to build more complex web applications using standards such as WS-BPEL [47]. Because of the convenience, low cost and capacity [1] to be composed into high-level business processes, web service technology is becoming increasingly popular.

With the ever increasing number of functional similar web services being available on the 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 web service meets specified requirements or user needs [62], 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 [34]. Study [33] shows that network latency is a significant component of web service response delay. Ignoring network latency will underestimate response time by more than 80 percent [56], since network latency is related to network topology as well as physical distance [32]. 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 provided by Web Server Hosting Providers (WSHPs). WSPs need to allocate their services wisely so that the overall network latency is minimized. According to a popular web traffic analyzing company Alexa, 96% of top one million web services were hosted in heterogeneous server clusters or co-location centers [28] that were widely distributed across different geographic regions. Hence, it is necessary to provide an effective web services allocation guide to WSPs so that they can be benefited.

The Web service location allocation problem is very challenging because it is a combinatorial optimization problem. A combinatorial optimization has a non-continuous and rugged search space, which makes it difficult to search for the optimal solutions. Furthermore, Web service location allocation problem is essential a multi-objective optimization problem [9] for which there are two conflicting objectives, to minimize latency and total cost. Previous researches [1, 56] use integer programming and greedy algorithm to solve this problem. However, these approaches are either easy to be stuck at local optima or performs poorly when problem size increases.

Multi-objective Evolutionary Optimization Algorithm (MOEA) methodologies are ideal for solving multi-objective optimization problems [20], since MOEAs work with a population of solutions. With an emphasis on moving towards the true Pareto-optimal region, a MOEA algorithm can be used to find multiple Pareto-optimal solutions in one single simulation run [35]. Specifically, [23, 26] discover that Particle Swarm Optimization (PSO)-based multi-objective optimization algorithms has the same or better effectiveness as the Genetic Algorithm (GA)-based multi-objective optimization algorithms but with significantly better computational efficiency (less function evaluations). Therefore, PSO-based algorithms are chosen to solve the problem.

1.2 Objectives

The overall goal is to develop a new PSO-based approach to the Web service location allocation problem by considering two potentially conflicting objectives - minimizing cost and minimizing network latency.

To accomplish this goal, we design three objectives:

1. Develop an aggregating approach using PSO which combines two objectives into a single fitness function and investigate whether the aggregating approach can do a good job for balancing the two conflicting objectives.

2. Develop a Pareto front approach using multi-objective nondominated-sorting PSO (NSPSO). It is the first time that NSPSO is used to solve Web service location allocation problem, and we investigate whether the Pareto front approach outperforms the aggregating approach.

3. Develop a new multi-objective PSO approach by considering better diversity and performance when problem size increases. We investigate whether this approach can produce better performance than the aggregating and Pareto front approaches.

1.3 Major Contributions

The project has three contributions.

1. This project shows that an aggregating approach converge well and be able to achieve the optimal solution in small problems.

2. This project shows that a NSPSO Pareto front approach achieve better performance than aggregating approach. It can provide a set of solutions that are distributed more diversely than the aggregating approach.

3. This project presents a new multi-objective PSO-based approach with two major improvements. Firstly, We develop a rounding function mechanism which transforms a continuous algorithm to a binary algorithm. The dynamic rounding function provides better results with good diversity. Secondly, we develop an adaptive threshold mechanism which embodies the idea of transfer learning. Similar to dynamic rounding function, it also provides the ability to solve binary problem and promising results. Another desired feature of dynamic and adaptive rounding functions is that they perform well regardless of the sizes of problems.

1.4 Organization

The report is organized as follows. Chapter 2 provides background information. Chapter 3 presents a detailed Web service location allocation problem description and model formulation. It also presents a BPSO with an aggregating approach. Chapter 4 presents a NSPSO-based approach. Chapter 5 presents a new binary 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

Service-oriented computing (SOC) provides a new paradigm for building distributed computing applications over the Internet. It gradually changes the way of software design, distribute and consume. In SOC, web services are the building blocks to support effective, low-cost development of applications in different environments [8]. SOC depends on service-oriented architecture (SOA) to organize applications and infrastructures into a set of interacting web services. The recent development of web service provides a common framework that allows data to be shared and reused across applications, enterprises and community boundaries. That is, web services become on-line resources.

In order to provide good Quality of Service (QoS), the fundamental objectives are optimizing the properties of QoS such as availability, security and response time. Availability [41] represents the probability that a service in available, a large value denotes the service is always ready to use while a small value denotes the service is unstable. Security [4] is a measurement of web service of providing access control, encrypting messages. Security is now becoming more important because web service invocation occurs over the public Internet. Response time is a major measurement of service performance while low latency values indicate low response time. Latency is the round trip time between sending a request and receiving the response. Latency is now the main impediment to improving performance [21]. Since network latency is related to network topology as well as physical distance [32], it is hard to reduce latency by analysing the underlying network topology. A straightforward way to reduce latency is that treat network as a black box and allocate services according to round trip experiments [10]. This is the main reason that WSPs need to choose service locations carefully.

Some service users may have specific requirements such as fast response time and strong availability. These constraints must be considered before launching web services. WSPs may also have constraints such as an overall cost constraint. In different locations, WSHPs might charge different prices. There are two ways to reduce the overall cost, deploying less services or deploying services in locations with low price. In addition, a common constraint is the service number constraint which ensures every service is deployed at least to one location.

In this project, we consider minimizing latency and overall cost as the objectives because response time is one of the most important service performance measurements and the overall cost is the major concern of WSPs. For the sake of simplicity, we only consider a service number constraint.

We introduce some terminologies used in a Web service location allocation context. To solve the Web service location allocation problem, some basic information must be provided by *Web Service Providers (WSPs)*. A WSP must provide a list of *user centers* and a list of *candidate locations*. A user center indicates a center city of an *user concentrated area*. This step can be achieved by analyzing web service requests data from existing web services or conducting a market survey. A candidate location is the geometric location that is suitable to deploy web services (e.g. existing *Web Server Hosting Providers (WSHPs)*). The selection of a candidate location is based on other criteria such as facilities or rental fees of servers. User centers and candidate locations are not necessarily overlapped. A WSP also need to provide *service invocation frequency*, *network latency* and *web service fixed deployment cost* at each candidate location. *Service invocation frequency* denotes the number of invocations from an user center to a web service within a period of time; *network latency* denotes the average latency between user centers and candidate locations over a period of time; *web service fixed deployment 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 web services
4. Average network latency from user centers to candidate locations
5. Web service fixed deployment cost at each candidate location

It is worth noting that the data in the above list are changing over time and 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 that the web service W was once popular in U became unfrequented. At this point, existing web services need to be re-allocated in order to adapt the market. Other scenarios such as, the network latencies vary in different time periods within a day. Existing web services do not need to re-allocated because the average network latency remains stable.

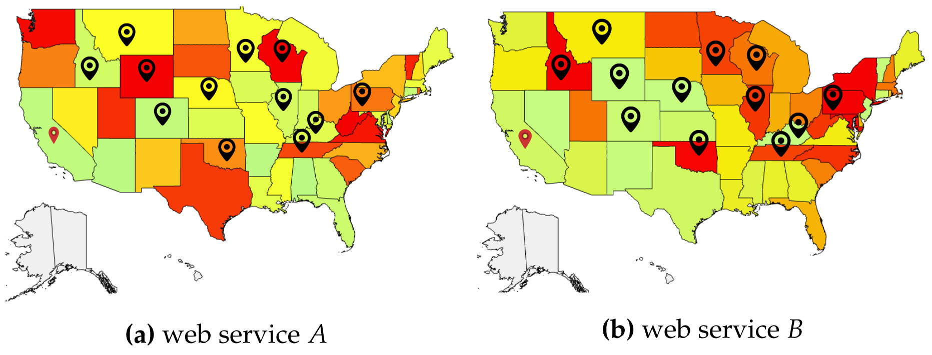

(a) web service A **(b)** web service B

Figure 2.1: User distribution of two web services

We illustrate these terminologies with a simple example (Figure 2.1). A Web service provider has N web services initially deployed in San Francisco, California (red pin on Figure 2.1). During the next a few months, the WSP constantly received complaints from all over the country about the high response time of the services. The WSP seeks some ways to improve the quality. They found a straightforward way to re-allocate the web services. Therefore, they conducted a statistical analysis over the raw data that are collected from web services. Based on the data analysis, the WSP found each web service has its own *user concentrated area* (each state) . For example, Figure 2.1 shows the popularity of web services A and B across the country. Red denotes high popularity while green denotes low. In each state, a city is selected as the *user center* (e.g. the capital city). The black pins denote the *candidate locations* which are also cities select from states. As we can see the user distributions are quite distinct for A and B. Now the WSP would choose locations for these web services. The WSP soon realized the problem is too complex to use any analytical methods for the following reasons:

1. Large number of web services
2. Large number of candidate locations and user centers
3. Trade-off between minimizing cost and minimizing network latency
4. Constraints (e.g. minimal web service number requirement)

2.2 Evolutionary Computation

Evolutionary Computation is a subfield of Artificial Intelligence. It is very well-known for its effective global search ability. These algorithms are inspired by biological mechanisms of evolution, social interactions and swarm intelligence. There are several distinguished characteristics of EC algorithms such as the use of a population and the ability of avoiding local optima. A common process of EC algorithms is as follows. Initially, the population are randomly generated in the search space. During the evolution process, the population moves according to a predefined fitness function. At the end of the evolution, the individual with the best fitness value is selected as the output.

The origins of evolutionary computation can be traced back to 1950s [6]. Since 1970s, the output of EC research has grown exponentially. The majority of current implementations of EC algorithms descend from three major approaches: *genetic algorithms*, *evolutionary programming* and *evolution strategies*.

GA [29] was introduced by Holland. There is a large number of applications uses GA [17, 18]. Evolutionary programming, introduced by Fogel [22], was originally designed to generate artificial intelligence. Evolutionary strategies as developed by Rechenberg [52] and Schwefel [54], were initially designed with the goal of solving difficult discrete and continuous, mainly experimental, parameter optimization problems [39].

In the next a few sections, we introduce the background knowledge of each algorithm we will use in this project.

2.2.1 Particle Swarm Optimization (PSO)

PSO was proposed by Kennedy and Eberhart in 1995 [36]. PSO is a population based meta-heuristic 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.

2.2.2 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) [37]. 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 $(0, 1)$.

2.2.3 Multi-Objective Evolutionary Optimization Algorithm

Multi-objective Evolutionary Optimization Algorithm (MOEA) was initially designed in the mid-1980s. Since then, people find MOEAs both efficient and effective because MOEAs deal simultaneously with a set of possible solutions which allow us to find multiple non-dominated solutions in a single run. Additionally, MOEAs are less suffering from the shape or continuity of the Pareto front (e.g. they can deal with discontinuous and concave Pareto fronts) [2]. In MOEAs, there are many strategies to deal with multi-objectives. The most popular strategy is the linear aggregation approach, which aggregates all the objective values into a single value [13]. Nonlinear aggregation approaches were also popular [14]. A Pareto front approach was first introduced by Goldberg in his seminal book on GA [24]. Goldberg suggested to use nondominated ranking and selection to move a population to the Pareto front. This idea currently is the mainstream in MOEA.

2.2.4 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 et.al [19] 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.

2.2.5 Multi-Objective PSO

Several multi-objective optimization algorithms are based on PSO such as Multi-objective PSO (MOPSO) [11], and Non Dominated Sorting PSO (NSPSO) [44]. The performance of different multi-objective algorithms is compared in [11] using five test functions. These algorithms are NSGA-II [19], PAES [40], Micro-GA [15] and MOPSO. The results show that MOPSO is 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. Apart from the good optimization ability, another major advantage for PSO-based algorithms is low computational cost. They are more effective than other EC algorithms.

Raquel et al. [51] propose a MOPSOCD extended from 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 as a single objective problem. [1, 56] try to solve the problem by using integer linear programming techniques. In particular, [56] solves this problem by employing greedy and linear relaxation.

Researches on network virtualization [7, 25] employs greedy algorithms to allocate virtual machines (VMs) in the data center so that the requirements of network bandwidth are met. [42] presents a multi-layer and integrated fashion through a convex integer programming formulation.

The major drawback of greedy algorithm is that it is easy to be stuck at local optima. Integer linear programming is well-known as not scaling very well. It performs poorly when the number of variables is large.

2.3.2 EC Approaches

Huang [30] proposes an enhanced genetic algorithm (GA)-based approach on Web service location allocation. He models the problem as a single objective problem with respect to network latency. In particular, the position of a web service in a Web service composition workflow is considered in his model. Kessaci [38] proposes MOGA-CB for minimizing cost of VMs instance and response time. [49] proposes 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 [19] with an additional local search process.

2.4 Summary

Previous researchers have studied the Web service location allocation problem with single-objective algorithm, 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 which should be addressed by multi-objective algorithms. From previous study, we found MOEAs are promising. Specifically, among many

MOEAs, 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

BPSO for Web Service Location Allocation: An Aggregating Approach

3.1 Introduction

The aim of Web service location allocation is to deploy services with a minimal cost of services to the candidate locations with low overall network latency. This problem considers two conflict objectives: minimizing cost and network latency. In an ideal situation, if WSPs deploy services to all candidate locations, web service users would receive the lowest latency. However, increasing the number of services means increasing cost. Therefore, the allocation must be considered carefully. In this chapter, we focuses on developing a binary PSO with an aggregating approach.

We first introduce the data that related to the problem and then model the problem with two objectives. In the second part, we apply a BPSO with an aggregating approach to solve the problem. In the experiment, we first design 14 problems with increasing variable sizes to test the performance of an algorithm. In addition, two small problems with small variable size are specifically designed for BPSO. We conduct experiments on BPSO. The results are discussed carefully from the perspective of convergence, evolution process and overall performance.

3.2 Problem Description and Model Formulation

In the previous study, Huang [30] developed a model of Web service location allocation. His approach considers one objective, minimizing network latency. In contrast, our approach considers the Web service location allocation as a multi-objective problem with two potentially conflicting objectives, minimizing cost and network latency. In this section, we first describe the Web service location allocation problem in detail. Then we introduce the model formulation.

3.2.1 Problem Description

For service location allocation problem, we need information of service invocation frequency, network latency, and service deployment cost. Assume a set of $S = \{s_1, s_2, \dots, s_m\}$ services are requested from a set of user centers $I = \{i_1, i_2, \dots, i_n\}$. WSPs allocate services to a set of candidate locations $J = \{j_1, j_2, \dots, j_k\}$.

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 to services. For example, f_{13} = 85 denotes service s_3 being called 85 times from i_1 in a period of time.

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 [60] [61].

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

A *service location matrix* $A = [a_{sj}]$ denotes whether the service s is deployed at a candidate location j with 1 denotes deployment, 0 denotes no deployment. Service location matrix A is the output matrix. Therefore, it is initially unknown.

A *response time matrix* $R = [r_{is}]$ denotes the response time from user centers to services. It can be calculated by using service location allocation matrix $A = [a_{sj}]$ and network latency matrix $L = [l_{ij}]$ according to Equation (3.1).

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

For example, we can calculate overall network latency by using 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.

3.2.2 Model Formulation

We set service location allocation matrix $A = a_{sj}$ be the decision variable.

$$a_{sj} = \begin{cases} 1 & \text{service } s \text{ deploy in location } j \\ 0 & \text{otherwise} \end{cases}$$

We design two objective functions and one constraint.

$$\begin{aligned} \text{minimize} \quad & Cost = \sum_{s \in S} \sum_{j \in J} c_{sj} a_{sj}, \\ & Latency = \sum_{i \in I} \sum_{s \in S} r_{is} f_{is} \\ \text{subject to} \quad & \sum_{j} a_{sj} \geqslant 1 \end{aligned} \tag{3.2}$$

Cost calculates the overall cost of deployed services, where c_{sj} is the cost of deployed service s at candidate location j, a_{sj} represents whether service s is allocated to candidate location j. The sum of the multiplication of Dc_{sj} and a_{sj} is the total deployment cost.

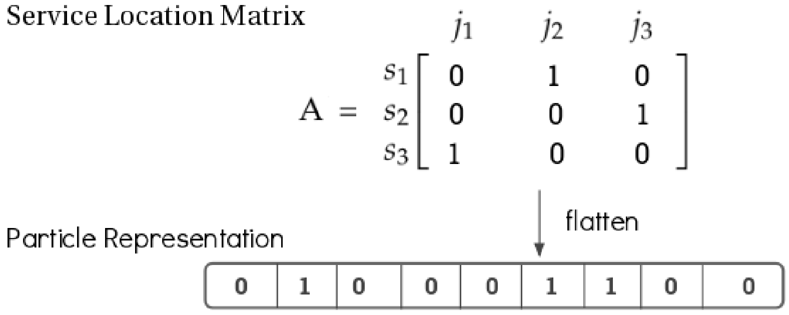
Latency calculates the overall network latency. Where r_{is} denotes the response time from a user center i to a web service s and f_{is} is the invocation frequency of a user center i to a web service s.

The constraint requires that each web service is deployed in at least one location.

3.3 Binary Particle Swarm Optimization

3.3.1 Particle Representation

As seen from the above section, Web service location allocation problem can be described with a set of matrices with F, L, C as input, A as output, $min(cost)$ and $min(latency)$ as objectives. In order to provide a baseline of optimization solutions, we adopt the original BPSO to encode allocation solutions as particles. We encode a flatten matrix as the particle representation. That is, all row vectors in a matrix are concatenated from top row to bottom row to construct a vector. Instead of directly using matrix-based representation, it uses a flattened matrix.



Service Location Matrix
$$A = \begin{array}{c|ccc} & j_1 & j_2 & j_3 \\ \hline s_1 & 0 & 1 & 0 \\ s_2 & 0 & 0 & 1 \\ s_3 & 1 & 0 & 0 \end{array}$$
flatten
Particle Representation
0 1 0 0 0 1 1 0 0

Figure 3.1: BPSO particle representation

3.3.2 Fitness Function

To measure the goodness of solutions we need fitness functions. We employ a linear aggregating method to combine all normalized objectives into one value, with a weight assigned to each objective to indicate its relative importance (Equation 3.5).

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 web services to all candidate locations. $Cost_{min}$ is the cost of allocating each of web services at a location that leads to the minimal cost and $Cost_{max}$ is the cost of allocating each web service to all the locations.

$$Cost' = \frac{Cost - Cost_{min}}{Cost_{max} - Cost_{min}} \tag{3.3}$$

$$Latency' = \frac{Latency - Latency_{min}}{Latency_{max} - Latency_{min}} \tag{3.4}$$

$$Fitness = w * Latency' + (1 - w) * Cost' \tag{3.5}$$

3.3.3 Algorithm

The key step is updating of *pbest* and *gbest*. The *pbest* of a particle is updated when its current fitness is smaller than previous *pbest*. The *gbest* is the best solution in all *pbest*. 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 reached
9: Return $Swarm$
```

3.3.4 Constraint Handling

We model the Web service location allocation problem with one constraint, it requires each web service to be deployed to at least one location. Because of the constraint, not all web service location allocation matrices are valid. Therefore, we need a constraint handling method to restrict the population generated by the algorithm. There are five categories of constraint handling summarized by Coello [12],

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" in constraint handling which is a special case of penalty function [12]. The invalid solutions are assigned the maximum fitness value. That means the penalized particle will lose chance to be the global best in the current generation. This penalty function prevents the swarm from moving to invalid regions.

$$Fitness(x_{id}) = \begin{cases} 1 & \text{if violate constraint} \\ \text{normalized fitness value} & \text{otherwise} \end{cases} \tag{3.6}$$

The invalid particles are remaining in the population. In the next generation, they could move out of invalid regions and become valid again.

3.4 Experiments and Results

3.4.1 Datasets

This project is based on both real world datasets and stimulated datasets. In this project, there are mainly three attributes 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 fourteen problems, which are either generated from existing data or generated from a normal distribution (introduced in Section *Cost*, *Service Invocation Frequency* and *Network Latency*). The problems are in increasing size and difficulty which are used as representative samples of the Web service location allocation problem.

Table 3.1: Problem set

Datasets	No. of Services	No. of Candidate Locations	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 WSP rents servers from Web Server Hosting Providers (WSHPs). The rental fee normally includes fixed deployment fees and variable fees based on usage [55]. Fixed deployment fees refer to the monthly payment for a specific server plan from a WSHP. Fixed deployment fees mainly depend on the quantity, namely the number of servers and quality features include computing power, reliability, network bandwidth. Variable fees include extra charges from exceeded storage and other limits. This project would only consider fixed deployment fees. The cost is under the following assumptions, the fixed deployment of web services in the same candidate location are the same. This project uses stimulated cost by randomly generating from a normal distribution where mean is 100, standard deviation is 20.

Service Invocation Frequency

Frequency refers to the invocation frequency from a location to a web service within a certain period of time [31]. This project uses simulated frequency data that randomly generate from an uniform distribution from 1 to 120.

Network Latency

Network latency denotes the latency between a user center and a candidate location. The network latency depends on not only the geographic locations but also the topological locations. 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 [60] [61], it measures the network latency between 339 user centers to 5825 candidate locations. The provided data is represented as a matrix.

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

The project uses subsets of the matrix. It randomly selects a subset of the matrix as the latency matrix L, where the number of user centers and candidate locations are defined by experimental design. The only selection constraint is the network latency value which can not be $Null$ value since we assume a WSP provides complete information.

For example, the experiment requires an $N * M$ latency matrix, that is, N user centers and M candidate locations. In the first step, it randomly selects an $N * M$ matrix from the data matrix. In the second step, it replaces the row which has $Null$ value with a random row. Repeat the second step until all matrix has no $Null$ value.

Small Problems

Two small problems are designed to validate that our proposed BPSO algorithm for web service allocation is capable of finding near optima solutions. The two small problems are using small datasets so that it is feasible to use a brute-force algorithm (e.g. evaluate all permutation of the solutions) to find the optimal solutions.

We define the small problems as follows:

Table 3.2: Small Problems

Datasets	No. of Services	No. of Candidate Locations	No. of User centers
Small problem 1	2	3	2
Small problem 2	4	4	3

The search space of these two small problems can be calculated as follows, because of each entry in the service location allocation matrix could be 0 or 1. The search space for problem 1 is $2^6 = 64$, problem 2 is $2^{16} = 65536$.

Small problem 1:

$$F = \begin{array}{c|cc} & s_1 & s_2 \\ i_1 & 10 & 116 \\ i_2 & 119 & 95 \end{array} \qquad L = \begin{array}{c|ccc} & j_1 & j_2 & j_3 \\ i_1 & 0 & 0.3 & 0.5 \\ i_2 & 0.4 & 0 & 0.6 \end{array} \qquad C = \begin{array}{c|ccc} & j_1 & j_2 & j_3 \\ s_1 & 98 & 72 & 144 \\ s_2 & 98 & 72 & 144 \end{array}$$

Small problem 2:

$$F = \begin{array}{c|cccc} & s_1 & s_2 & s_3 & \\ i_1 & 58 & 2 & 34 & 93 \\ i_2 & 104 & 111 & 56 & 70 \\ i_3 & 87 & 25 & 42 & 9 \end{array} \qquad L = \begin{array}{c|cccc} & j_1 & j_2 & j_3 & j_4 \\ i_1 & 0 & 0.125 & 0.044 & 0.054 \\ i_2 & 0.125 & 0 & 3.7 & 0.13 \\ i_3 & 0.044 & 3.7 & 0 & 0.36 \end{array}$$

$$C = \begin{array}{c} \\ s_1 \\ s_2 \\ s_2 \\ s_2 \end{array} \begin{array}{c} \begin{array}{cccc} j_1 & j_2 & j_3 & j_4 \end{array} \\ \begin{bmatrix} 70 & 69 & 91 & 120 \\ 70 & 69 & 91 & 120 \\ 70 & 69 & 91 & 120 \\ 70 & 69 & 91 & 120 \end{bmatrix} \end{array}$$

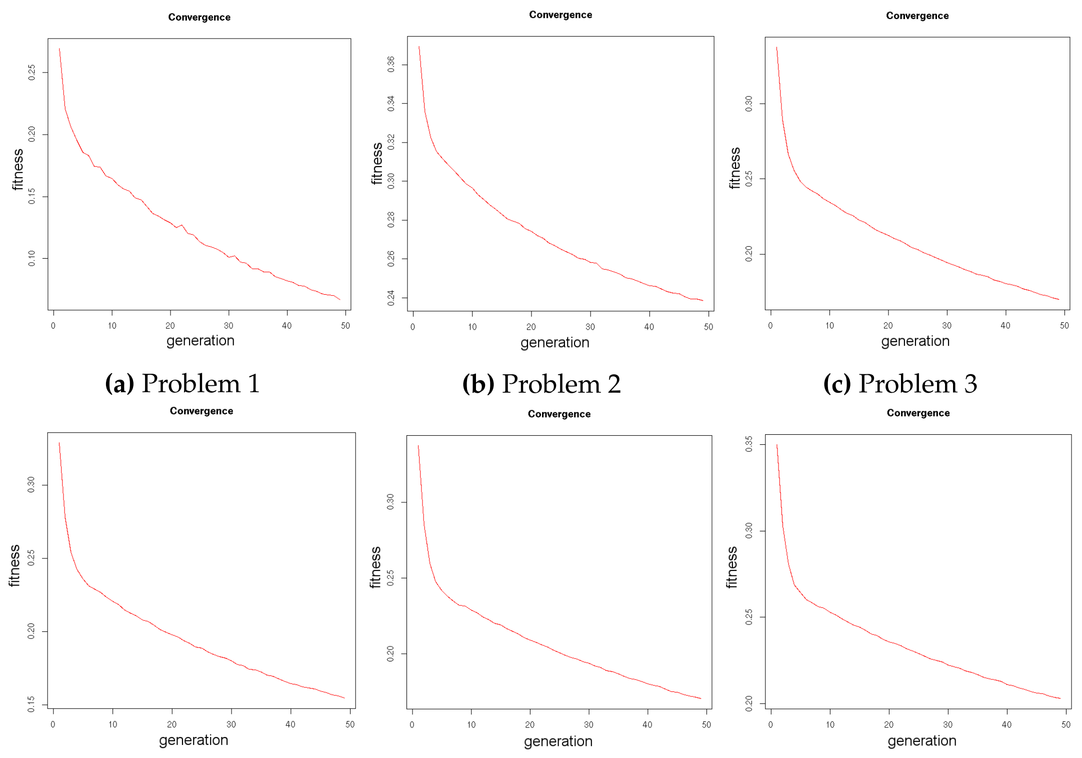
3.4.2 Experiment Design

We conduct experiment on Problems 1 $\sim$ 6 and two small problems.

The parameters of the BPSO algorithms are set as follows: $c_1 = 2$, $c_2 = 2$, 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 Results

We first present the convergence curves of BPSO in Figure 3.2. In each generation, we calculate the average of the best solutions of 40 runs . The x axis denotes the generation and y axis denotes the fitness value. As it is shown, the initial fitness value is large because the particles are randomly initialized. Along with the evolution, the fitness values follow a decreasing trend. It means that the BPSO finds better solutions in each generation.

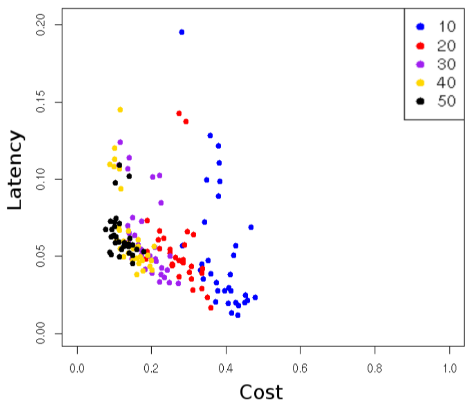


(a) Problem 1 **(b)** Problem 2 **(c)** Problem 3

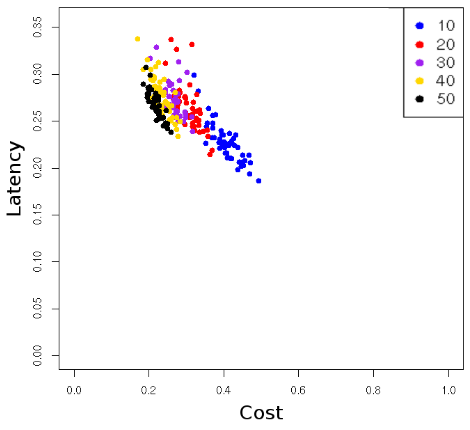
(d) Problem 4 **(e)** Problem 5 **(f)** Problem 6

Figure 3.2: Convergence curve by averaging 40 independent runs

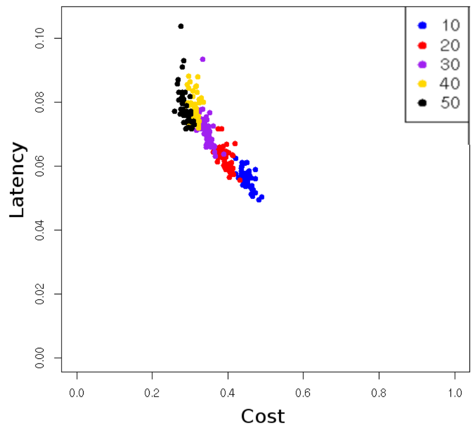
We could also see the evolutionary process by observing Figure 3.3 where it shows the solutions from 10th, 20th, 30th, 40th and 50th generation of problem 1 $\sim$ 4. It shows a pattern that the solutions are moving to a low fitness region. We could observe this pattern clearly from problem 1 and 2. As solutions move from blue region to black region. It is not so obvious in problem 3 and 4. The reason is that the BPSO could not find better solutions so the swarm schooling along the borderline which every solutions are equally good.



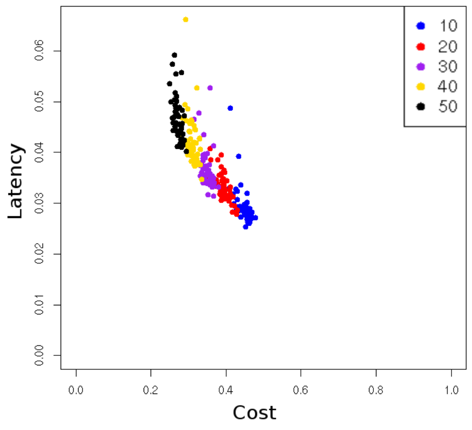
(a) Problem 1



(b) Problem 2



(c) Problem 3



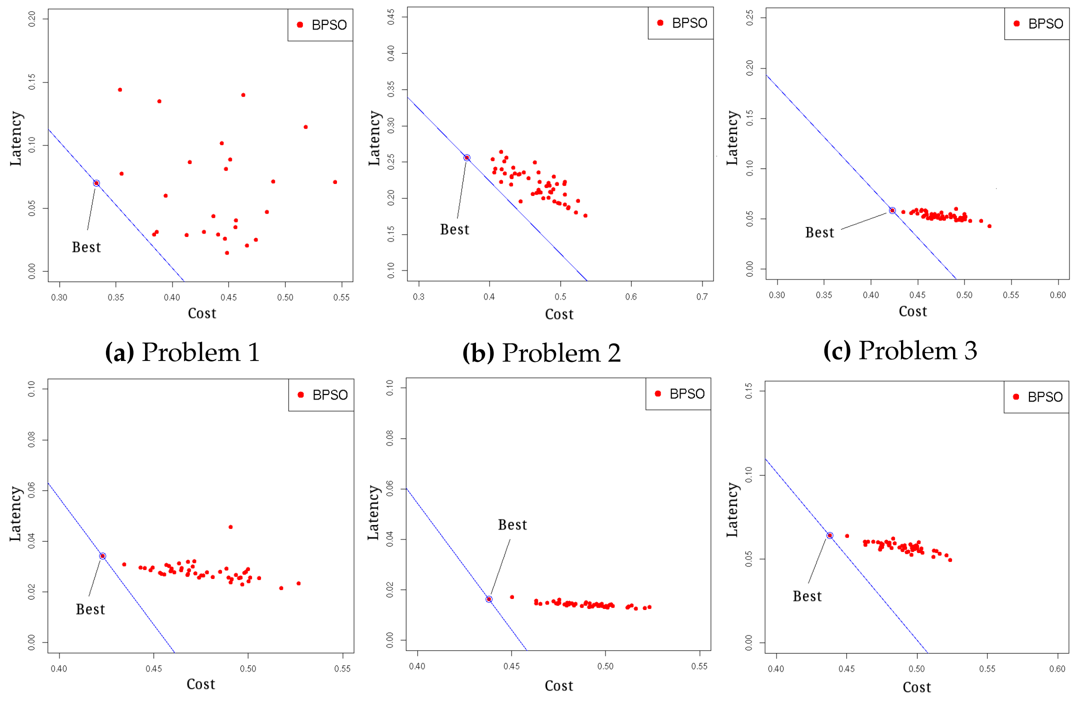
(d) Problem 4

Figure 3.3: BPSO Optimization Process

In BPSO, each particle is a solution. We plot the particles from the last generation with two objectives in Figure 3.4 . The red points denote the solutions and the marked point denotes the solution with best fitness value. We draw a blue line with $slope = 1$ passing through it.

We found that, as we set the *weight* in aggregating method to 0.5, the solutions are evolved along the direction of $y = x$. The solutions scattered along with the plane that orthogonal to the evolve direction. As there are two objectives in the problem, the plane is a line with $slope = -1$. Problems 2 $\sim$ 6 clearly shows this phenomenon. As we could observe that although the marked point has the best fitness value, it does not necessarily better than other solutions in the perspective of each objective. In other words, there are a few non-dominated solutions. In problem 3 $\sim$ 6, it is clearly that many solutions have lower latency and higher cost than the best fitness solution.

For the small problems, the results are presented in Table 3.3. We observed, for small problem 1, BPSO converged at a solution 001100, for small problem 2, it converged at 111100000000000. We convert them to a matrix representation, shown as A_1 and A_2,



(a) Problem 1 **(b)** Problem 2 **(c)** Problem 3

(d) Problem 4 **(e)** Problem 5 **(f)** Problem 6

Figure 3.4: BPSO Experiments: Solutions of Problems 1 $\sim$ 6

Table 3.3: Small problems

(a) Small problem 1

	fitness	particle
1	0.099	001100
2	0.099	001100
3	0.099	001100
4	0.099	001100
5	0.099	001100
6	0.19	001101
7 $\sim$ 50	0.099	001100

(b) Small problem 2

	fitness	particle
1	0.021	1111000000000000
2	0.021	1111000000000000
3	0.021	1111000000000000
4	0.021	1111000000000000
5	0.046	1111000001000000
6	0.0735	1111000010000000
7 $\sim$ 50	0.021	1111000000000000

$$A_1 = \begin{array}{c} \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \qquad A_2 = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{array} \begin{array}{c} \begin{array}{cccc} j_1 & j_2 & j_3 & j_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

For each small problem, we applied brute-force algorithm to search for the optimal solution. The results have proved PSO found the optimal solutions in both cases. Therefore, we know BPSO are able to find out optimal solution. For the large datasets, BPSO is able to converge towards the optimal solutions. Unfortunately, BPSO does not necessarily find optima in every problem because of many reasons,

1. Search space is too large.
2. Not run enough generation to converge.
3. The algorithm stuck at local optima.

In particular, we notice that in two small problems, all web services are deployed in the same locations. These are special cases which are not frequently happened. In a practical situation, optimal candidate locations for web services are not necessary the same. The choices are highly dependent on the given attributes (latency, invocation frequency and cost).

3.5 Discussions

WSPs may have their specific requirements, for example, financial service providers are willing to make more invention to improve the QoS while social network service providers prefer less expenditure even it means the decline of QoS. As we observed in Figure 3.4, clearly, the solutions are lack of diversity. This is the major drawback of using an aggregating method to solve multi-objective problems. The linear aggregating method tries to optimize the overall fitness and ignore each objective. As a consequence, the solutions are concentrated on a narrow region.

3.6 Summary

This chapter first introduces a model for Web service location allocation problem with two objectives. Second, it presents a BPSO approach which employs an linear aggregating method to combine two objectives. The BPSO uses a "death penalty" approach. Third, the chapter presents eight experiments over the BPSO including six normal size problems and two small problems. The results shown that BPSO can solve the problem we modeled and provides good solutions. However, the drawback of BPSO is that it couldn't provide a variety of choices to let service providers to choose according to their preferences. In the next step, we are going to apply a multi-objective PSO approach to treat each objective separately.

Chapter 4

A Pareto-Front Multi-Objective to Web Service Location Allocation

4.1 Introduction

In the previous chapter, we modeled the Web service location allocation problem and developed a BPSO with a linear aggregating approach. The results show BPSO provides good solutions but with insufficient diversity. In order to improve diversity, in this chapter, we develop a NSPSO-based approach with two separate fitness functions. In addition, in order to show the effectiveness of NSPSO, we also employ a common multi-objective evolutionary algorithm - NSGA-II. These algorithms are not new, but it is the first time to apply them to the Web service location allocation problem.

The goal of this chapter is to develop a NSPSO-based approach that provides solutions with a better diversity.

4.2 NSPSO for Web Service Location Allocation

We applied the NSPSO developed by Li [44]. The algorithm is shown in Algorithm 2. The particle representation is a flatten binary matrix which is the same as BPSO (Section 3.3.1) . The constraint handling method is also "death penalty" as in BPSO (Section 3.3.4).

4.2.1 Fitness Functions

In the previous section, we defined two objective functions *Cost* and *Latency*. Instead of combining two objective functions as one fitness function, In NSPSO, we would address them as two fitness functions.

$$CostFitness = \frac{Cost - Cost_{min}}{Cost_{max} - Cost_{min}} \tag{4.1}$$

$$LatencyFitness = \frac{Latency - Latency_{min}}{Latency_{max} - Latency_{min}} \tag{4.2}$$

4.3 NSGA-II for Service Location Allocation

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

Algorithm 2 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 non-dominated 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 reached
29: return the positions of particles in $F_1$;
```

location allocation problem is a binary problem, we adopt the discrete version. Similar to NSPSO, NSGA-II also adopted the two objective functions as the fitness functions.

4.3.1 Chromosome Representation

A chromosome can be used to represent a service location allocation matrix without making any changes.

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

4.3.2 Genetic Operators

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

Algorithm 3 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 $F_1$ ;
```

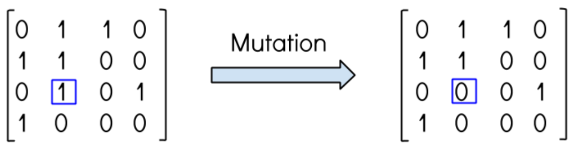


Figure 4.1: Mutation

Crossover This project adopt single point crossover. For each pair of chromosome, if a randomly generated 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 both chromosomes. Two offspring are generated after this operation.

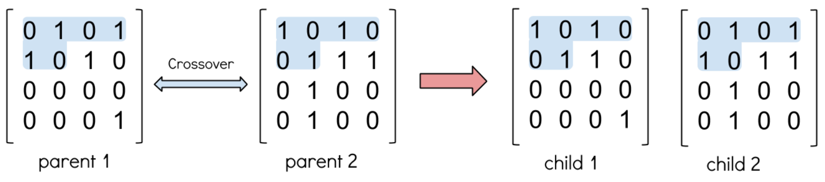


Figure 4.2: Crossover

4.3.3 Constraint Handling

In Section 3, we defined the Web service location allocation problem with a single constraint. In order to fulfill the constraint, we develop a repair function (Algorithm 4) to fix the invalid chromosomes. The reason that we use a repair function instead of "death penalty" is that "death penalty" is considered harmful in GA-based approaches. In GA-based approaches, chromosomes with bad fitness value have large probability being removed from the population in the next generation. Hence, the algorithm would lose chance to search these areas. In contrast, in PSO-based approaches, particles with bad fitness value remain in the population. The invalid particles could move out of invalid regions in the next generation. The function checks the constraint and fix an invalid chromosome by randomly deploying a web service to a candidate location.

Algorithm 4 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
```

4.4 Experiment Design

We conduct experiments using the 14 problem cases described in Section 3.4.1 over three algorithms: BPSO (Algorithm 1), NSGA-II (Algorithm 3) and NSPSO (Algorithm 2). In order to assess and compare the quality of their solutions, hypervolume method [5] is applied. Hypervolume calculates the volume between the non-dominated curve with respective to a reference point ((1, 1) in this project). Hypervolume measures both diversity as well as convergence.

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

BPSO with an aggregating approach outputs a solution set without using a non-dominated sorting. In order to compare with other multi-objective algorithms, the non-dominated solutions are extracted from its output.

To show the performance of the three algorithms, we compare the *best solutions* generated by the algorithm and the *average solutions* generated by the algorithms. A *best solution* set is generated by applying fast non-dominated sorting over a combination of 40 runs of solutions. We compare different algorithms by plotting best solutions from algorithms. In *average solutions*, we apply hypervolume method on the solutions of 40 runs. The mean and standard deviation of 40 hypervolume values are calculated.

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	-	-

4.5 Results

The *average solutions* are shown in Table 4.2. The *best solutions* are shown in Figure 4.3. In *average solutions*, BPSO dominates the other two algorithms in the first ten problems. In the last four problems, NSPSO dominates the performance. In the *best solutions*, it shows the solutions from BPSO has better convergence for the first two problems as they are closer

Table 4.2: Comparison between NSPSO, NSGA-II and BPSO: The mean and standard deviation of the hypervolume values over the 40 independent runs

Dataset	Method	Hypervolume (avg ± sd)
	NSPSO	0.76 ± 0.019
problem 1	NSGA-II	0.83 ± 0.013
	BPSO	**0.89 ± 0.016**
	NSPSO	**0.61 ± 0.011**
problem 2	NSGA-II	0.60 ± 0.012
	BPSO	**0.61**± 0.01
	NSPSO	0.61 ± 0.011
problem 3	NSGA-II	0.59 ± 0.007
	BPSO	**0.69 ± 0.007**
	NSPSO	0.63 ± 0.011
problem 4	NSGA-II	0.61 ± 0.007
	BPSO	**0.71 ± 0.008**
	NSPSO	0.61 ± 0.009
problem 5	NSGA-II	0.58 ± 0.005
	BPSO	**0.67 ± 0.007**
	NSPSO	0.59 ± 0.007
problem 6	NSGA-II	0.55 ± 0.006
	BPSO	**0.63 ± 0.005**
	NSPSO	0.60 ± 0.008
problem 7	NSGA-II	0.56 ± 0.005
	BPSO	**0.63 ± 0.006**
	NSPSO	0.61 ± 0.008
problem 8	NSGA-II	0.58 ± 0.005
	BPSO	**0.65 ± 0.008**
	NSPSO	0.60 ± 0.009
problem 9	NSGA-II	0.56 ± 0.004
	BPSO	**0.62 ± 0.005**
	NSPSO	0.58 ± 0.008
problem 10	NSGA-II	0.53 ± 0.005
	BPSO	**0.58 ± 0.005**
	NSPSO	**0.57 ± 0.009**
problem 11	NSGA-II	0.52 ± 0.003
	BPSO	0.55 ± 0.003
	NSPSO	**0.58 ± 0.009**
problem 12	NSGA-II	0.52 ± 0.003
	BPSO	0.56 ± 0.003
	NSPSO	**0.59 ± 0.008**
problem 13	NSGA-II	0.53 ± 0.003
	BPSO	0.56 ± 0.004
	NSPSO	**0.59 ± 0.009**
problem 14	NSGA-II	0.53 ± 0.002
	BPSO	0.57 ± 0.003

to the original point. As the problem size getting larger, BPSO's advantage of convergence gradually diminishes. In addition, the diversity of BPSO is clearly much worse than NSGA-II and NSPSO as its solutions could only cover a small region of the Pareto front.

In comparison between NSGA-II and NSPSO, NSPSO has a better performance in all problems except the first one (Table 4.2). NSPSO's advantage in diversity become more and more clear as the problem size increases. In the first and second problem, NSGA-II dominates NSPSO in convergence. From problem 3, NSPSO outperforms NSGA-II in both measurements. The diversity of solutions is the major difference between NSPSO and NSGA-II. Especially, when the data set is getting very large, NSPSO could still maintain a diverse of solutions while the other algorithms could only provide some solutions within a narrow area.

Another observation is that, there is a decreasing trend in the average solutions among all three algorithms with the problem size increasing.

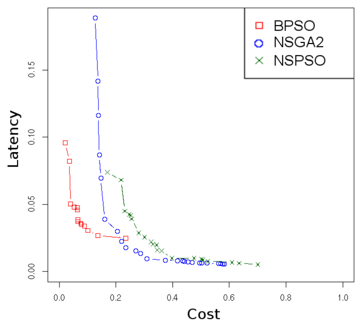
4.6 Discussions

The experiments show that 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 noticed from the current results. Firstly, the solutions are not diverse enough. Figure 4.3, especially, Problems 3 $\sim$ 9 show that NSPSO could not cover the Pareto front. It is because NSPSO lacks a mechanism to avoid being stuck at local optima. Secondly, all three algorithms are suffering from decreasing of performance when problem size increases. It is a major problem since the number of web services and locations are increasing everyday. An algorithm without scalability would be less useful for service allocation task with a big search space.

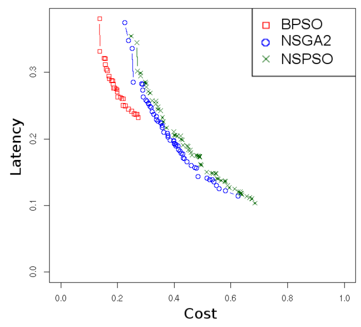
4.7 Summary

In this chapter, we proposed a NSPSO-based approach to solve the Web service location allocation with two fitness functions. In order to show the quality of its solutions, we compare it with a NSGA-II approach. We conducted experiments with the three algorithms over 14 problems. The results show NSPSO provides diverse solutions in most cases. The results also show all three algorithms have poor performances when handling big datasets. In the next step, we are going to solve this problem by developing a new BMOPSOCD approach.

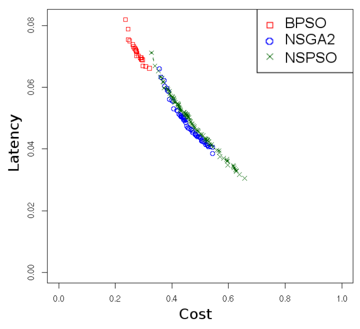
Figure 4.3: NSPSO, NSGA-II and BPSO Experiments: The non-dominated solutions among the set obtained by 40 independent runs



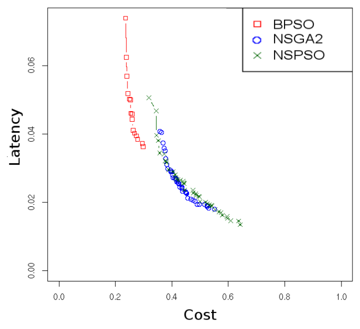
(a) Problem 1



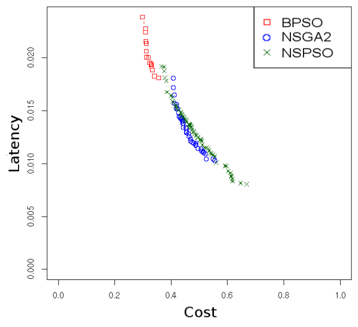
(b) Problem 2



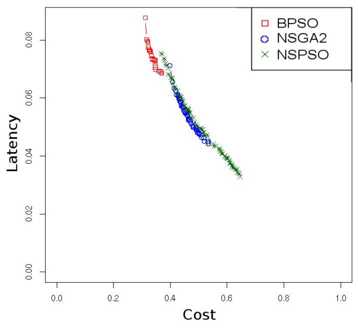
(c) Problem 3



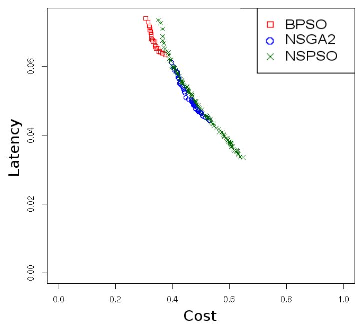
(d) Problem 4



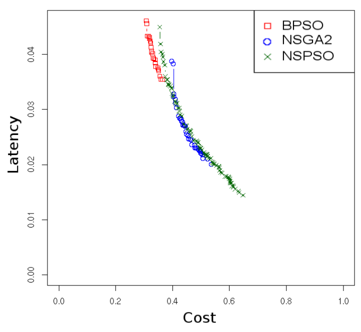
(e) Problem 5



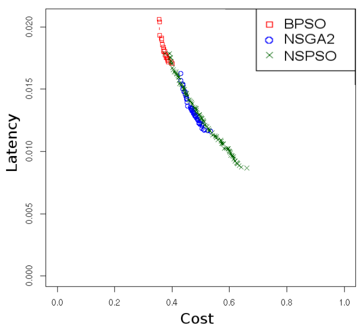
(f) Problem 6



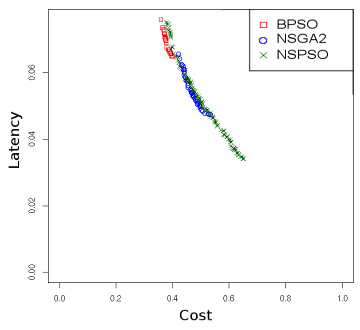
(g) Problem 7



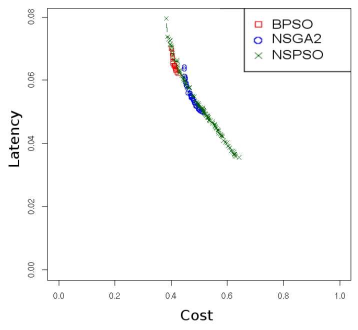
(h) Problem 8



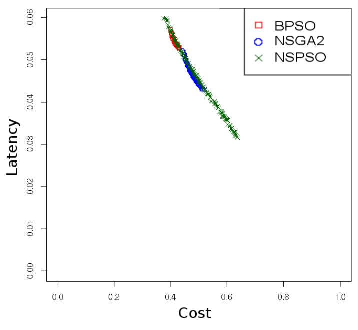
(i) Problem 9



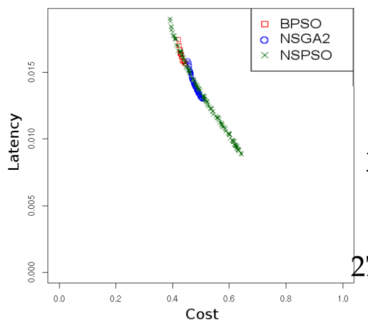
(j) Problem 10



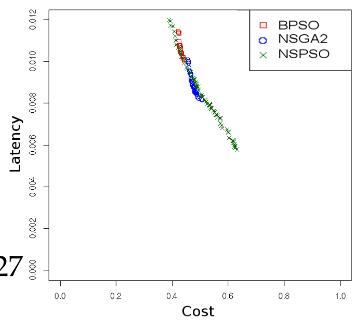
(k) Problem 11



(l) Problem 12



(m) Problem 13



(n) Problem 14

Chapter 5

A new BMOPSOCD for Service Location Allocation

5.1 Introduction

In the previous chapter, we find that there are two major shortcomings in BPSO, NSPSO and NSGA-II. The first one is their performances drop rapidly when the dataset increases. The second shortcoming is that the solutions are not diverse enough. In order to further improve the performance, this chapter develops a new binary multi-objective PSO with crowding distance approach to solve the Web service location allocation problem. We introduce a new mechanism, the rounding function to the original MOPSOCD. The rounding function method is a mechanism that makes a continuous algorithm compatible with discrete problems.

We develop this approach based on MOPSOCD [51] because it has two desired features: archive and mutation. These features make the algorithm to have a strong ability to avoid stucking at local optima while maintaining an uniformly non-dominated set.

5.2 BMOPSOCD

The selection of *pbest* and *gbest* is one of the key steps in BMOPSOCD. *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 the BMOPSOCD, any non-dominated solutions in the archive can be a *gbest*, therefore, it is important to ensure that the particles move to an unexplored area. The *gbest* is selected from non-dominated solutions with highest crowding distance value. It ensures the swarm to move to a least crowded area.

5.2.1 Particle Representation

The major difference between BMOPSOCD and the other three algorithms is the particle representation. Although it is a binary approach, the particle remains a continuous representation. We also employ the flatten method as described in the previous section.

Algorithm 5 BMOPSOCD 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 non-dominated solution into $Archive$, remove dominated solutions from $Archive$
20: until maximum iterations is reached
21: return $Archive$ =0
```

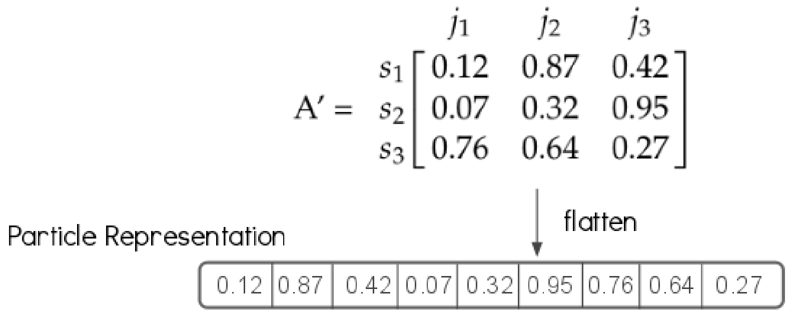


Figure 5.1: BMOPSOCD particle representation

5.2.2 Fitness Function

Two fitness functions are used to achieve two objectives, cost fitness and latency fitness. They are the same for NSPSO, defined in Section 4.2.1.

5.2.3 Constraint Handling

The constraint handling method used by BMOPSOCD is a ranking of violations. A solution I is considered to 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 dominates solution J.

The particle with less violations is always considered as a better solution. If there is only one constraint, this constraint handling method provides similar effect with death penalty method.

5.2.4 Rounding Functions

The original MOPSOCD is designed as a continuous version PSO. Instead of changing the particle to a binary representation, we still use the continuous representation. In order to be compatible with the binary problem, 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 values. 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 used to map a real value particle to a discrete value particle. The common strategy is to round a real value to its closest integer number. The round-down strategy is adopted in [43] to solve integer programming problem. [27] 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. [45] adopts rounding and interval mapping strategy to solve 0-1 discrete, integer optimization and mixed optimization problem. [3] uses a random-round function which randomly returns round-up value or round-down value.

A Static Rounding Function

A static rounding function is a straightforward strategy. A parameter threshold t is introduced in the 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 Functions

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 predefined. 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. Because of 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 as static rounding function, either round up or round down according to t. 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 that we design three dynamic functions is that we would like to compare the impact of different trajectories of dynamic thresholds. t is the value of threshold, g is the current generation. The lower boundary of a threshold is l, upper boundary is u. They are predefined. The performance of these rounding functions is studied in the Section 5.4.1.

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

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

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

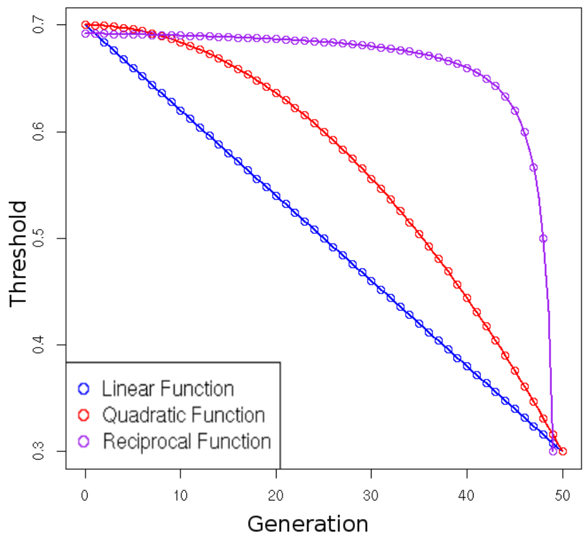


Figure 5.2: Curve of three dynamic thresholds

5.2.5 An Adaptive Threshold Approach

Human have the ability to learn a technique or knowledge and apply in different fields. As the dimensionality of the problem increases, 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 [46]. In this section, we proposed an adaptive threshold that embodied in the transfer learning process.

Figure 5.3 shows the evolutionary process with an adaptive threshold. Initially, the threshold t is set to a upper boundary u (e.g. 0.7). Then the PSO runns with this setting for a predefined 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 5.5) and remain steadily until next interval. Repeat this process until the lower boundary l reached. The optimization may look like forcing the swarm "jump" to a different area. But the process is equivalent to initializing a new set of population with the old one. Therefore the knowledge are inherited. An 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 explore a different area and is 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.

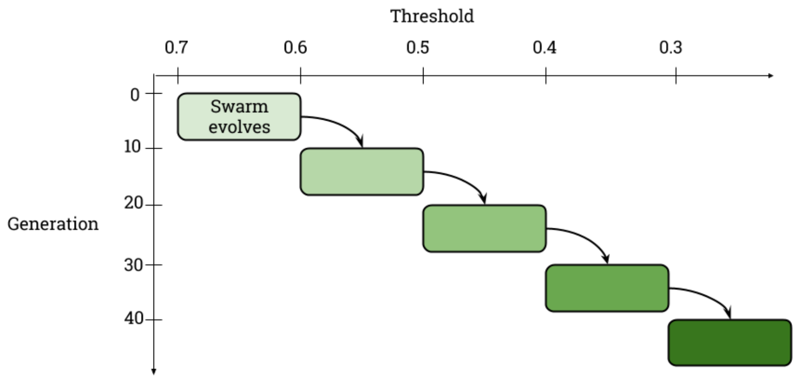


Figure 5.3: Evolutionary process with an adaptive threshold

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

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 adaptive threshold. Its performance is studied in the third experiment. An experiment of a combination of static rounding function is conducted and discussed. Lastly, we conduct an experiment considering the overall performance of a BMOPSOCD with a dynamic rounding function in comparison with other three algorithms: PSO, NSPSO and NSGA-II (presented in Chapter 4).

5.3.1 Performance Metrics

The IGD [59] is a modified version of generational distance [58, 57] as a way of estimating how far the elements in the true Pareto front are from those in the Pareto front produced by our algorithm. It calculates the sum of the distances from each point of the true Pareto front to the nearest point of the non-dominated set that produced by an algorithm. The lower the IGD, the better quality the solution is. A true Pareto front is needed when calculating 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 (BMOPSOCD, NSGA-II, NSPSO, BPSO) and then apply a non-dominated sorting over it. The approximated true Pareto front dominates all the other solutions. We use hypervolume and IGD as the evaluation metrics.

5.3.2 Experiments on Rounding Functions

This section designs four experiments to study the effect of different types of rounding functions. Four datasets (Problems 2 $\sim$ 5) are used, chosen from the Table 3.1.

Static Rounding Function

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

In order to answer these two questions, a set of experiments is 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 parameters of the algorithm 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. For each experiment, the proposed algorithm has been independently run 40 times. The results are compared using the *best result* approach. 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 performances of different rounding functions are 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 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. The upper boundary of dynamic threshold $u = 0.7$ and the lower boundary of dynamic threshold $l = 0.3$. The results are compared using *average solutions*.

Figure 5.2 shows threshold t changes along with the generations. It is easy to notice that the points on linear and quadratic functions are uniformly distributed. Points on reciprocal are unevenly distributed.

An Adaptive Threshold Approach

The performance of the adaptive threshold approach is studied in this experiment. The parameters of the PSO algorithms 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. The upper boundary of dynamic threshold $u = 0.7$ and the lower boundary of dynamic threshold $l = 0.3$. The interval is set to 10. The results are compared with dynamic functions with *average solution* approach.

A Combination of Static Rounding Function

In the Section 5.3.2, we design an experiment to discover the effect of static thresholds. The idea of combination of static rounding funtion is based on that experiment. The idea is obtaining the result produced by different thresholds and combining them into a dataset. Finally, employs a fast non-dominated sorting over the it. The experimental settings are the same as in Section 5.3.2 and the result is evaluated with hypervolume and compared with reciprocal dynamic rounding.

5.3.3 BMOPSOCD versus NSPSO, NSGA-II and BPSO

In this section, we compare the performance of BMOPSOCD 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 BMOPSOCD. 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
BMOPSOCD			1	1	0.4	0.5	-
NSPSO	50	50	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 a non-dominated solution set.

5.4 Experimental Results and Discussion

5.4.1 Experiments on Rounding Functions

Static Rounding Function

The experiment results of static threshold are shown in Figure 5.4. It clearly show a pattern where the solution produced by different static thresholds cover a part of the 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 of 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.

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

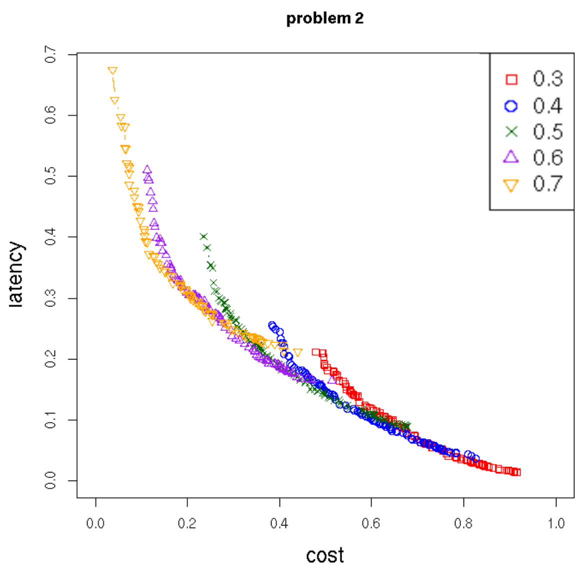
Figure 5.5: Rounding process

It means there are 70% of chance avoid deploying a web service in a location. Intuitively, the optimization with threshold of 0.7 would considers optimizing cost over latency by deploying less web 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 other hand, the solutions with $t < 0.5$ thresholds are encouraged to optimize latency over cost, therefore, more web services are deployed.

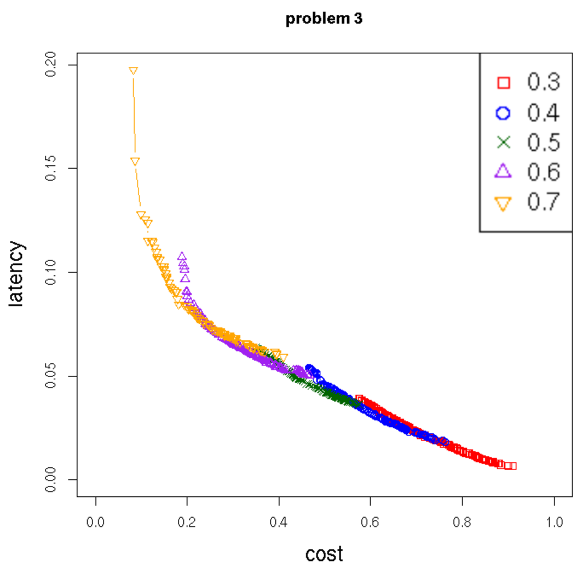
The experimental results clearly answered the first question: the influence of different thresholds. However, it is not offer a guide of selection of a proper threshold, because none of the solutions could cover the entire Pareto front. Worth noting that, a solution of combining all results is ideal,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

Dynamic Rounding Functions

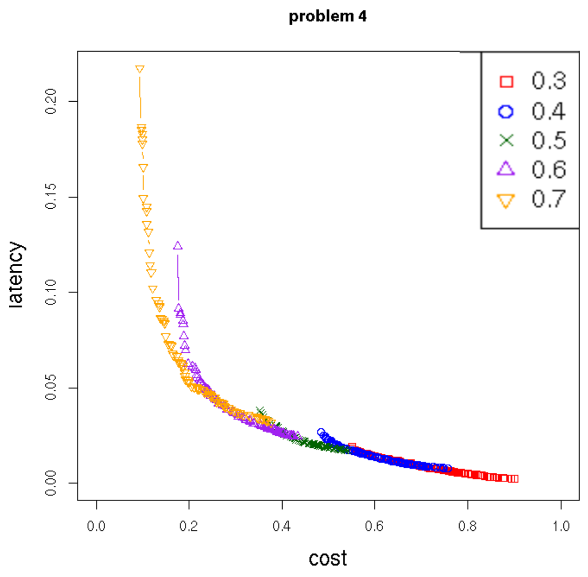
Table 5.2 shows the average performance of three dynamic functions that evaluated by hypervolume. The results indicate the reciprocal function dominated the three dynamic functions in all test cases. This effect could be visually observed by plotting the best results



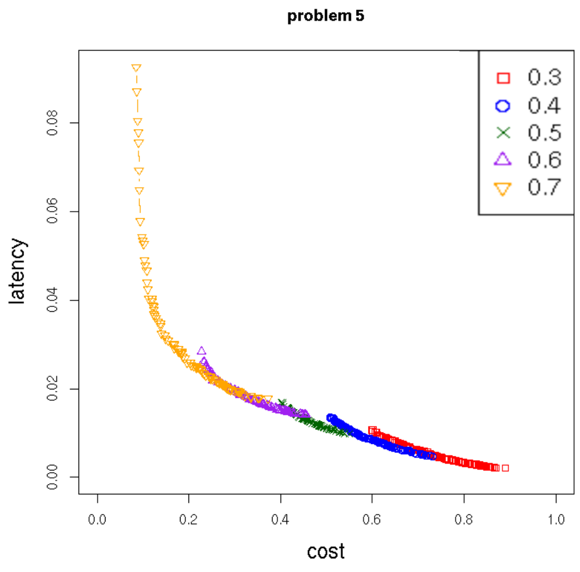
(a)



(b)



(c)



(d)

Figure 5.4: Static Rounding Function Experiments: Solutions from Problems 2 $\sim$ 5 with different static threshold values

(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 the solutions are not complete. There are gaps in Problems 3 $\sim$ 5. The gap is created because of the uniformity of the reciprocal function (Figure 5.2).

Table 5.2: The mean and standard deviation of the hypervolume values over the 40 independent runs

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

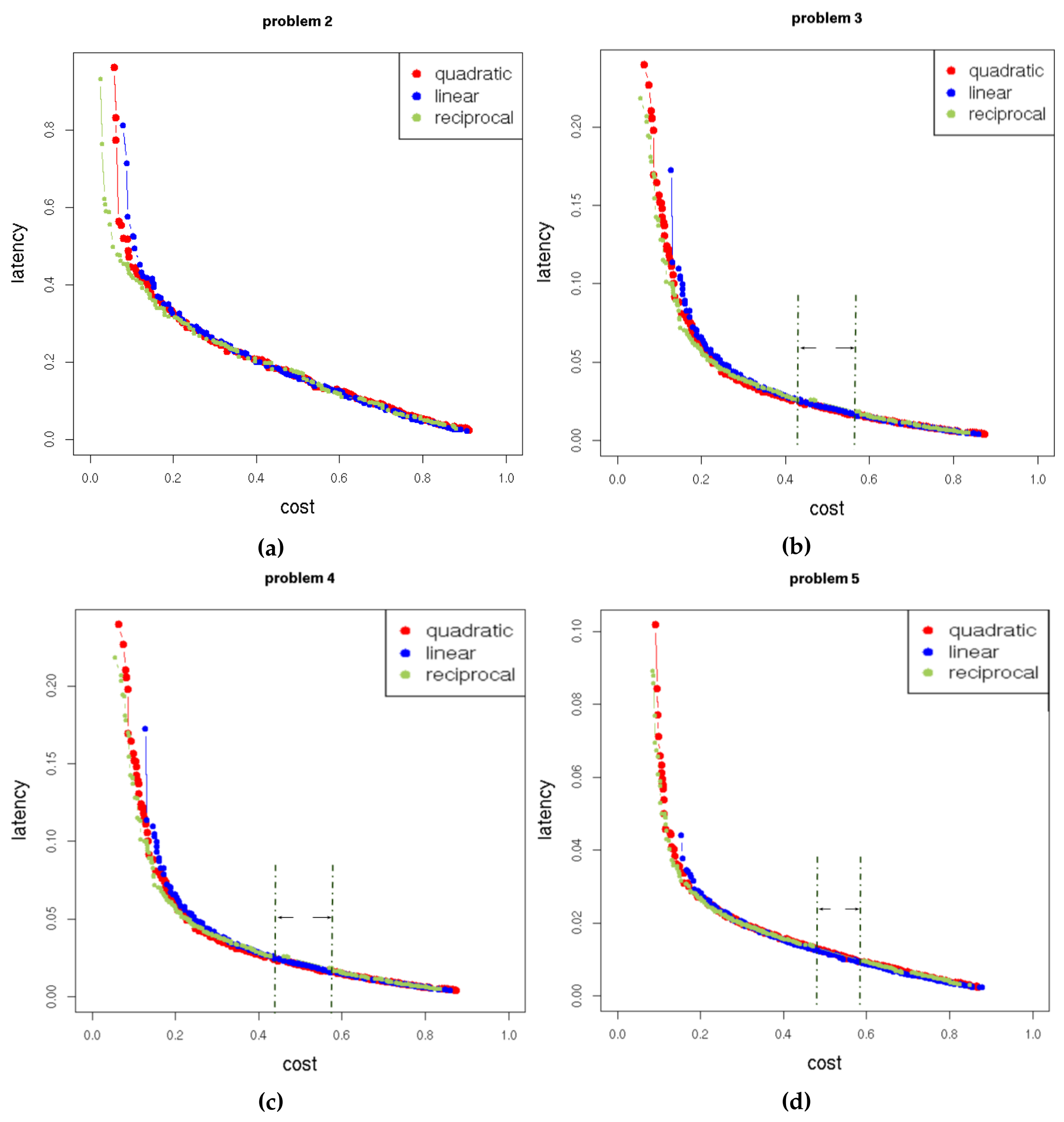


Figure 5.6: Dynamic Rounding Function Experiments : The non-dominated solutions among the sets obtained by 40 independent runs of BMOPSOCD with different dynamic functions

According to Table 5.2, the experimental results show that in terms of convergence, reciprocal function produces the best result. However, it also shows the major disadvantage of reciprocal function, which 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 along 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 are considered to be harmful.

Table 5.3: A comparison between Reciprocal function and Adaptive function, the mean and standard deviation of hypervolume values over the 40 independent runs

	Reciprocal	Adaptive
problem 2	**0.74 ± 0.013**	0.72 ± 0.011
problem 3	0.815 ± 0.013	**0.83 ± 0.014**
problem 4	**0.87 ± 0.014**	0.85 ± 0.014
problem 5	**0.86 ± 0.020**	0.85 ± 0.022

With these experiment results, it is still not easy to answer *Which dynamic rounding function produces the best results*. In the perspective of algorithm design, convergence and diversity are both important. The little advantage of convergence in reciprocal function might be considered as trivial, but the gap in the non-dominated set 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.

An Adaptive Threshold Approach

We compared the performance of the adaptive threshold approach with reciprocal rounding function. Table 5.3 clearly shows reciprocal function dominates the adaptive threshold approach in the most cases. However, Figure 5.7 shows the adaptive threshold approach dominates in Problem 3 and has better diversity in Problem 4. The results show another desired feature of the adaptive threshold approach. It could provide an uniformly distributed non-dominated set. Overall, the performance of the adaptive threshold approach is very close to reciprocal function.

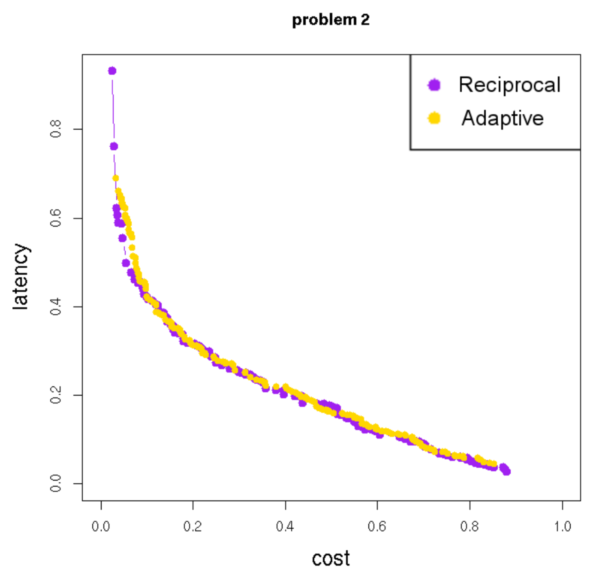
A Combination of Static Function

In this experiment, we combined all solutions from 5 static rounding functions 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 uniformly distributed. However, the major problem is that this method takes five times longer execution time than using reciprocal function.

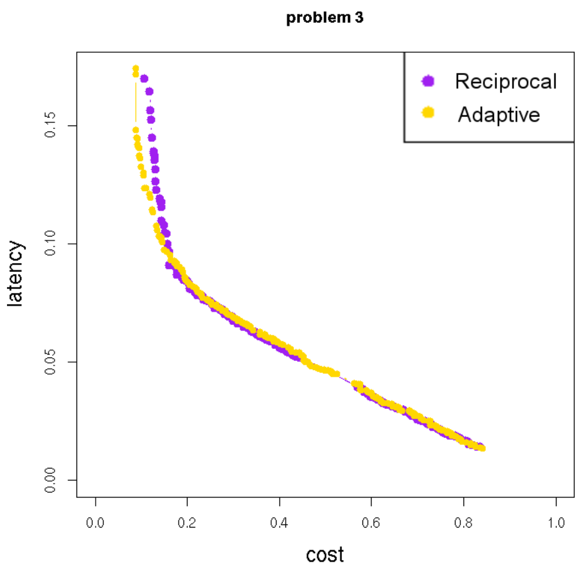
5.4.2 BMOPSOCD versus NSPSO, NSGA-II and BPSO

The result is shown in Table 5.4. In this table, "Ave-", "Std-" illustrate the average and standard deviation of four approaches over the 40 independent runs. It can be seen from Table 5.4, on all datasets except one, BMOPSOCD dominates other algorithms in both hypervolume and IGD. The only exception is Problem 1, where BPSO has the best hypervolume value. On all datasets, only BMOPSOCD 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 datasets, BMOPSOCD achieved a considerably better performance than other three algorithms in IGD which indicates a high coverage.

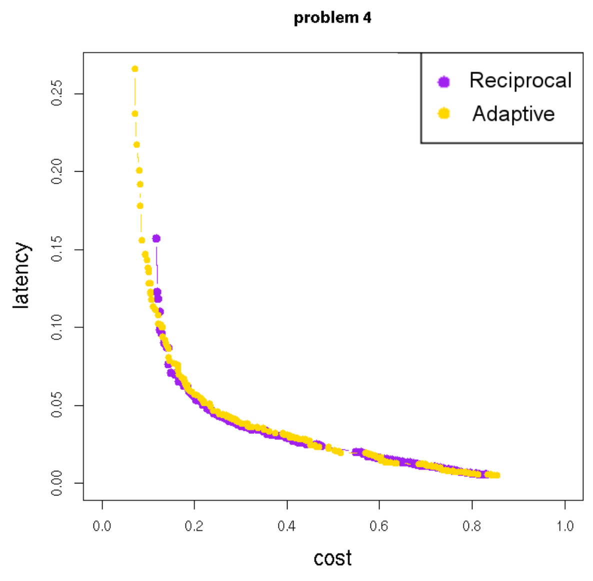
In comparison with NSPSO and NSGA-II, BMOPSOCD with dynamic rounding function achieved significantly better convergence and diversity. The first reason is that with the dynamic rounding function, BMOPSOCD could move out of local optima. In contrast, NSGA-II and NSPSO are easy to stuck at local optima. The second reason is BMOPSOCD keeps an external archive. Although three algorithms maintain a same size population, they



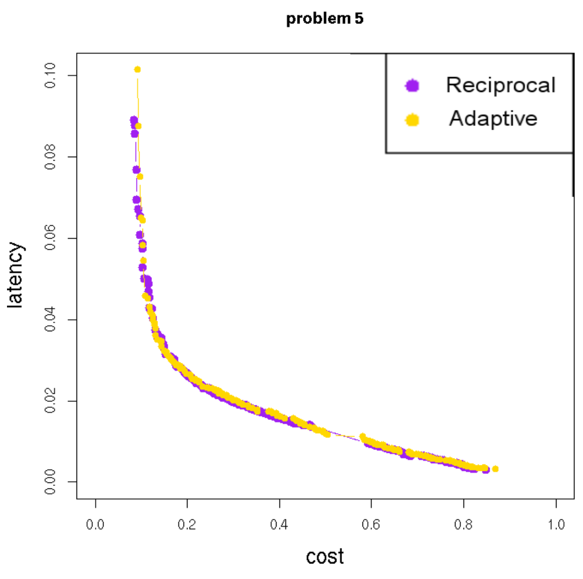
(a)



(b)



(c)



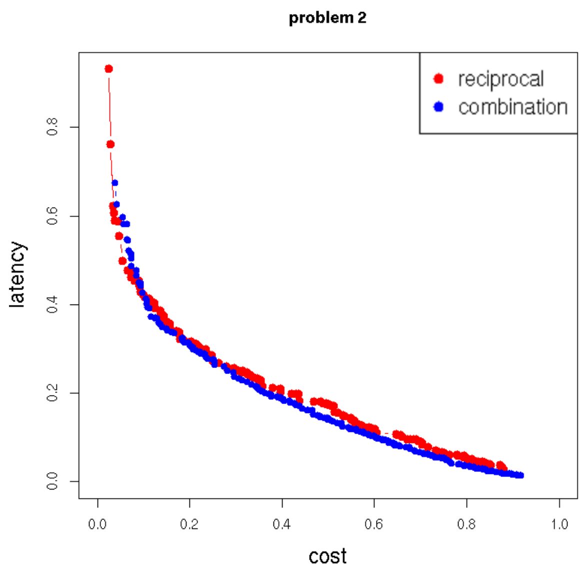
(d)

Figure 5.7: Adaptive Rounding Function Experiments: The non-dominated solutions among the sets obtained by 40 independent runs of adaptive function and reciprocal function

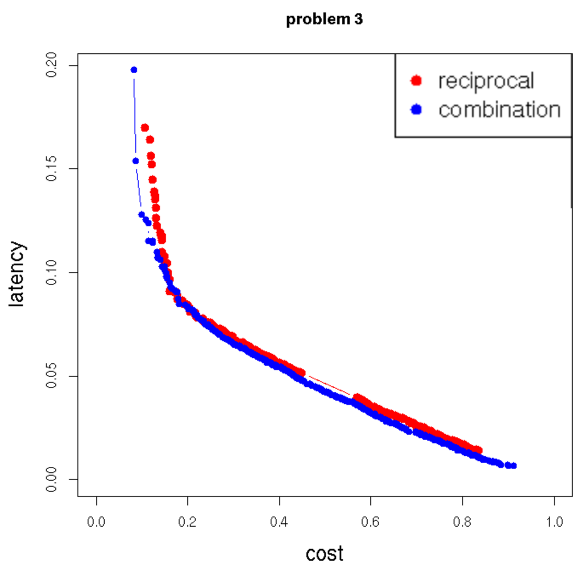
produce different sizes of solutions. MOSPCOD outputs an archive with a size of 250 while other two algorithms output a population of size 50.

In Problem 1, the convergence of BMOPSOCD is worse than BPSO. One reason is probably 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 completely converge. Another reason is related to the variable size, where BPSO has better performance in small datasets, when the number of datasets increases, the performance drops rapidly. In contrast, BMOPSOCD with the dynamic rounding function is not affected by the variable sizes.

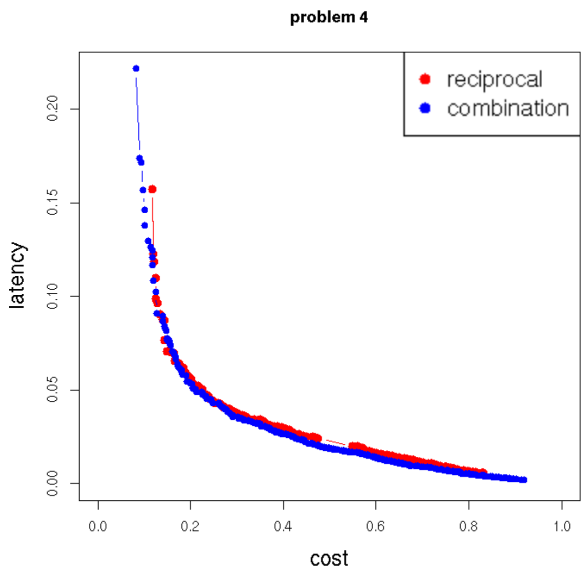
In terms of execution time, although BMOPSOCD is not as good as NSGA-II, it achieves best or the second best for 11 out of 14 problems (Table 5.5).



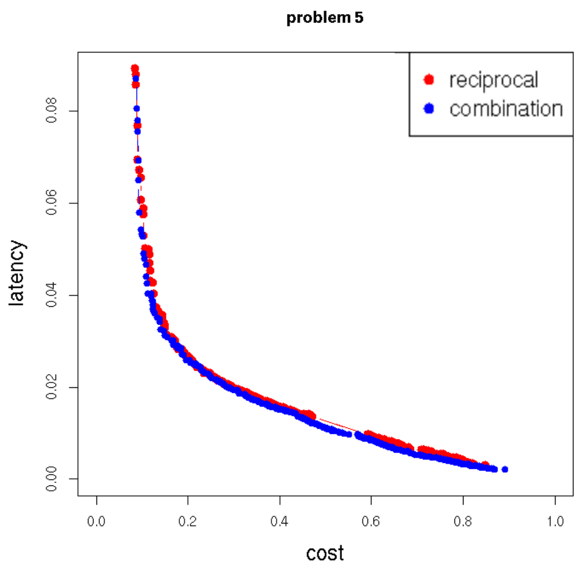
(a)



(b)



(c)



(d)

Figure 5.8: Combination of Static Function Experiments: The non-dominated solutions among the sets obtained by 40 independent runs of combination of static thresholds and reciprocal function

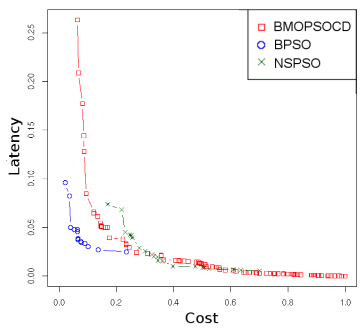
5.4.3 Discussions

In this chapter, we proposed a binary version of multi-objective PSO with crowding distance to solve the Web service location allocation problem. The major contribution is that, we provide a rounding function and an adaptive threshold technique to make a continuous algorithm compatible with a binary problem. These techniques can also be applied in other continuous algorithms. BMOPSOCD shows 2 major desirable features.

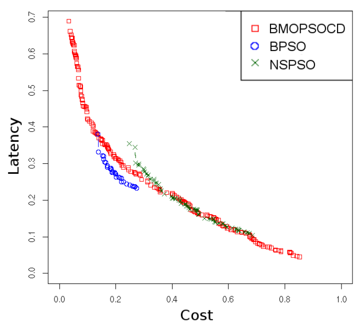
1. The solution set has a good diversity which covers most of the Pareto front.

2. BMOPSOCD could produce good solutions regardless of increasing problem size.

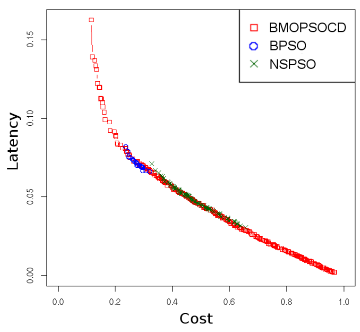
Figure 5.9: MOPSOCD, NSPSO and BPSO Experiments: The non-dominated solutions among the sets obtained by 40 independent runs of different algorithms



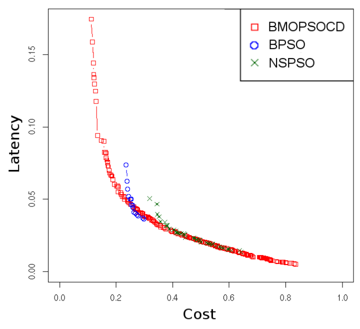
(a) Problem 1



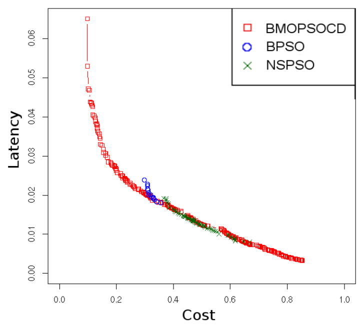
(b) Problem 2



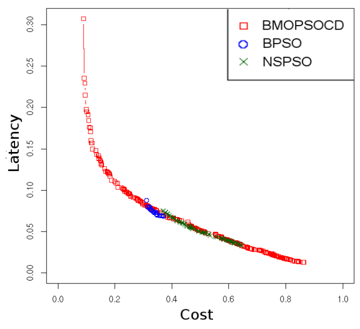
(c) Problem 3



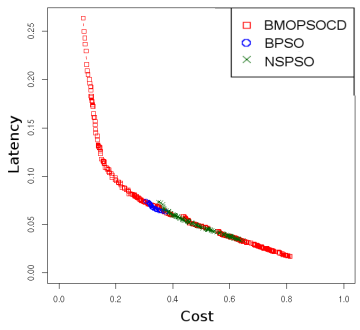
(d) Problem 4



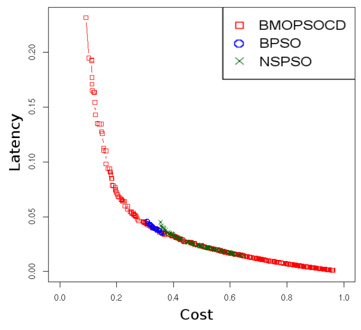
(e) Problem 5



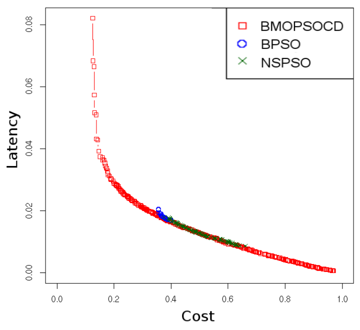
(f) Problem 6



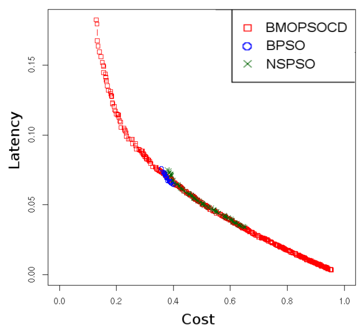
(g) Problem 7



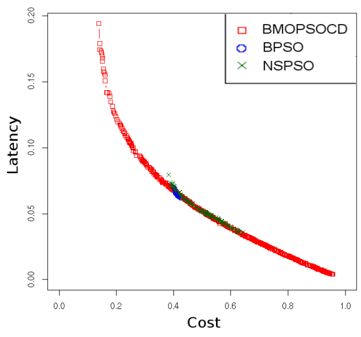
(h) Problem 8



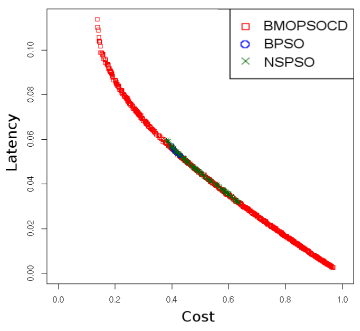
(i) Problem 9



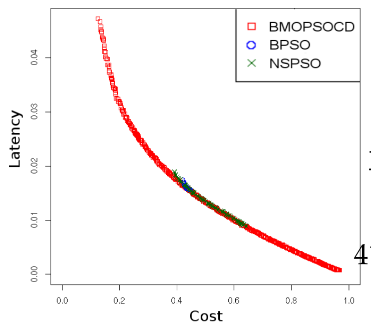
(j) Problem 10



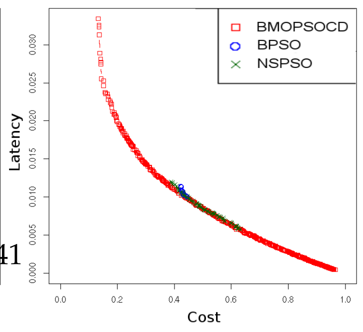
(k) Problem 11



(l) Problem 12



(m) Problem 13



(n) Problem 14

5.4.4 Summary

This chapter proposed a BMOPSOCD to solve the Web service location allocation problem. We introduce a rounding function technique enables the continuous algorithm to handle binary problems. Meanwhile, an adaptive threshold embodies the idea of transfer learning. The proposed algorithm was compared with previous approaches. The results show that, on most datasets, BMOPSOCD can achieve much better results in both convergence and diversity than other three methods, NSPSO, NSGA-II and BPSO. Additionally, the performance of MOSPCOD with dynamic rounding function is not affected by the size of variable. That is a significantly improvement.

Table 5.4: Comparison between BMOPSOCD, NSPSO, NSGA-II and BPSO: The non-dominated solutions among the sets obtained by 40 independent runs of different algorithms

Dataset	Method	Hypervolume (avg ± sd)	IGD (avg ± sd)
problem 1	BMOPSOCD	0.83 ± 0.04	**3.73E-02 ± 1.03E-02**
	NSPSO	0.76 ± 0.018	0.16 ± 3.45E-02
	NSGA-II	0.83 ± 0.013	0.19 ± 3.21E-02
	BPSO	**0.89 ± 0.015**	0.46 ± 2.45E-02
problem 2	BMOPSOCD	**0.73 ± 0.011**	**3.15E-02 ± 7.92E-03**
	NSPSO	0.61 ± 0.001	0.15 ± 1.46E-02
	NSGA-II	0.60 ± 0.011	0.19 ± 1.81E-02
	BPSO	0.61 ± 0.001	0.42 ± 1.54E-02
problem 3	BMOPSOCD	**0.81 ± 0.012**	**7.03E-03 ± 1.92E-03**
	NSPSO	0.61 ± 0.011	0.10 ± 6.35E-03
	NSGA-II	0.59 ± 0.008	0.16 ± 7.25E-03
	BPSO	0.69 ± 0.007	0.30 ± 8.94E-03
problem 4	BMOPSOCD	**0.83 ± 0.016**	**5.80E-03 ± 1.37E-03**
	NSPSO	0.63 ± 0.012	0.11 ± 8.55E-03
	NSGA-II	0.61 ± 0.008	0.17 ± 9.11E-03
	BPSO	0.71 ± 0.008	0.30 ± 8.62E-03
problem 5	BMOPSOCD	**0.84 ± 0.014**	**3.74E-03 ± 1.02E-03**
	NSPSO	0.61 ± 0.009	0.11 ± 7.93E-03
	NSGA-II	0.58 ± 0.005	0.17 ± 6.89E-03
	BPSO	0.67 ± 0.007	0.24 ± 5.02E-03
problem 6	BMOPSOCD	**0.81 ± 0.014**	**5.81E-03 ± 1.95E-03**
	NSPSO	0.59 ± 0.007	0.11 ± 5.06E-03
	NSGA-II	0.55 ± 0.006	0.18 ± 8.01E-03
	BPSO	0.63 ± 0.005	0.28 ± 5.88E-03
problem 7	BMOPSOCD	**0.79 ± 0.015**	**7.13E-03 ± 1.70E-03**
	NSPSO	0.60 ± 0.008	0.10 ± 4.58E-03
	NSGA-II	0.56 ± 0.005	0.17 ± 6.48E-03
	BPSO	0.63 ± 0.006	0.27 ± 7.92E-03
problem 8	BMOPSOCD	**0.81 ± 0.015**	**8.77E-03 ± 1.90E-03**
	NSPSO	0.61 ± 0.008	0.12 ± 5.81E-03
	NSGA-II	0.58 ± 0.005	0.19 ± 6.17E-03
	BPSO	0.65 ± 0.007	0.27 ± 5.86E-03
problem 9	BMOPSOCD	**0.83 ± 0.016**	**3.58E-03 ± 1.53E-03**
	NSPSO	0.60 ± 0.009	0.11 ± 5.33E-03
	NSGA-II	0.56 ± 0.004	0.17 ± 3.56E-03
	BPSO	0.62 ± 0.005	0.22 ± 3.22E-03
problem 10	BMOPSOCD	**0.80 ± 0.012**	**4.30E-03 ± 1.75E-03**
	NSPSO	0.58 ± 0.008	0.11 ± 4.97E-03
	NSGA-II	0.53 ± 0.005	0.19 ± 5.62E-03
	BPSO	0.58 ± 0.005	0.25 ± 3.91E-03
problem 11	BMOPSOCD	**0.79 ± 0.012**	**5.19E-03 ± 1.81E-03**
	NSPSO	0.57 ± 0.009	0.12 ± 4.22E-03
	NSGA-II	0.52 ± 0.003	0.20 ± 2.74E-03
	BPSO	0.55 ± 0.003	0.24 ± 3.36E-03
problem 12	BMOPSOCD	**0.80 ± 0.01**	**3.12E-03 ± 6.71E-04**
	NSPSO	0.58 ± 0.009	0.12 ± 4.15E-03
	NSGA-II	0.52 ± 0.003	0.20 ± 3.08E-03
	BPSO	0.56 ± 0.003	0.24 ± 2.50E-03
problem 13	BMOPSOCD	**0.83 ± 0.012**	**2.63E-03 ± 6.73E-04**
	NSPSO	0.59 ± 0.008	0.12 ± 3.46E-03
	NSGA-II	0.53 ± 0.003	0.20 ± 3.04E-03
	BPSO	0.56 ± 0.004	0.22 ± 2.01E-03
problem 14	BMOPSOCD	**0.84 ± 0.015**	**3.66E-03 ± 1.75E-03**
	NSPSO	0.59 ± 0.009	0.13 ± 3.85E-03
	NSGA-II	0.53 ± 0.002	0.22 ± 3.24E-03
	BPSO	0.57 ± 0.003	0.25 ± 1.83E-03

Table 5.5: Execution time

	method	time (avg ± sd)
problem 1	BPSO	17.99 ± 0.26
	BMOPSOCD	**12.98 ± 0.18**
	NSPSO	19.00 ± 0.17
	NSGA-II	15.35 ± 0.15
problem 2	BPSO	23.55 ± 0.27
	BMOPSOCD	**16.18 ± 0.26**
	NSPSO	25.52 ± 0.27
	NSGA-II	15.38 ± 0.31
problem 3	BPSO	103.65 ± 1.87
	BMOPSOCD	94.98 ± 7.28
	NSPSO	111.86 ± 1.11
	NSGA-II	**74.34 ± 0.61**
problem 4	BPSO	181.20 ± 4.40
	BMOPSOCD	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
	BMOPSOCD	172.80 ± 7.68
	NSPSO	236.23 ± 2.72
	NSGA-II	**157.52± 1.62**
problem 7	BPSO	234.73 ± 6.42
	BMOPSOCD	202.68 ± 10.46
	NSPSO	242.94 ± 9.00
	NSGA-II	**159.26 ± 1.31**
problem 8	BPSO	476.76 ± 22.40
	BMOPSOCD	531.64 ± 43.14
	NSPSO	444.41 ± 22.86
	NSGA-II	**375.05 ± 4.11**
problem 9	BPSO	293.43 ± 3.01
	BMOPSOCD	198.81 ± 7.11
	NSPSO	334.62 ± 2.81
	NSGA-II	**181.30 ± 1.99**
problem 10	BPSO	507.72 ± 4.19
	BMOPSOCD	449.91 ± 26.00
	NSPSO	539.51 ± 4.06
	NSGA-II	**381.18 ± 3.06**
problem 11	BPSO	1,237.30 ± 42.06
	BMOPSOCD	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
	BMOPSOCD	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
	BMOPSOCD	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
	BMOPSOCD	**3,284.66 ± 124.13**
	NSPSO	3,759.51± 61.49
	NSGA-II	3,372.53 ± 31.05

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The goal of this project is to develop a PSO-based approach to Web service location allocation problem by considering two objectives - minimizing cost and network latency. The goal was successfully achieved by accomplishing three objectives.

First, we develop a BPSO approach which employs an linear aggregation method to combine two objectives. It uses a "death penalty" constraint handling approach. We conducted eight experiments including six normal size problems and two small problems. The results shown BPSO adapted the model well and provided good solutions. It also shows the BPSO is trying to look for the optimal solution. However because the search spaces are too large for most the problems, BPSO fails to find the optimal solution. The major drawback of BPSO with aggregation approach is that its solutions are not diverse enough. Therefore, in the next step, we treat each objective separately and develop a NSPSO-basd approach.

Second, We apply NSPSO with a Pareto front to solve this problem. In order to overcome the drawback of BPSO with aggregation approach, we take each objective as a fitness funciton. In order to make a comparison with other approaches, we also adopt a common MOEA - NSGA-II to solve this problem. We conduct 14 experiments over three algorithms: BPSO, NSPSO and NSGA-II. the results show NSPSO provides the best diversity among three algorithms. Therefore the objective is achieved. However, the results also reveal three algorithms could not handle big datasets very well. In addition, although NSPSO provides a good diversity, it does not cover the complete Pareto front. Hence, a BMOPSOCD is developed in the next chapter.

Third, we proposed a BMOPSOCD approach to solve the Web service location allocation problem. The objective is to further improve the diversity and keep high quality solutions when dealing with large datasets. To achieve this objective, we introduce a rounding function mechanism which not only makes a continuous algorithm compatible with binary problems but also significantly improved the quality of solutions. Specifically, three types of rounding functions and an adaptive rounding function were developed. From the experiments, we observed that the solutions from BMOPSOCD with dynamic rounding functions have a great diversity that almost covers the whole Pareto front. Meanwhile, BMOPSOCD could produce good solutions regardless of increasing problem size.

6.2 Future Work

In this project, although we have successfully achieved the goal of developing a PSO-based algorithm that produce good solutions, there are still some limitations. Firstly, our model can be further improved by considering service composition. For now, the problem model considers each service as an atomic service. As the service composition has become a hot research area, it is expected that the composited services will become the mainstream. The service composition workflow has a significant impact on the allocation of atomic service because the data flow between services could not be neglected. Therefore, the location of each atomic service is highly related to the previous and the next service in a workflow. Secondly, more potential objectives need to be considered, for example, the availability problem. In order to avoid single point failure, WSPs normally deploy multiple services in different candidate locations to keep the availability. Green economy could also being considered. As the issue of global warming becomes a world-wide challenge, deploying a service to a location that close to a power plant has been proposed by many literatures [53]. In addition, we only considered a single constraint in this project. In a practical situation, there might be other constraints such as the overall cost and bandwidth requirements. Our future work will also address these issues.

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] ABRAHAM, A., AND JAIN, L. *Evolutionary multiobjective optimization*. Springer, 2005.

[3] 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.

[4] ANISETTI, M., ARDAGNA, C. A., DAMIANI, E., AND SAONARA, F. A test-based security certification scheme for web services. *ACM Trans. Web 7*, 2 (May 2013), 5:1–5:41.

[5] 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* (2009), FOGA '09, ACM, pp. 87–102.

[6] BÄCK, T., HAMMEL, U., AND SCHWEFEL, H.-P. Evolutionary computation: Comments on the history and current state. *Evolutionary computation, IEEE Transactions on 1*, 1 (1997), 3–17.

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

[8] BOUGUETTAYA, A., SHENG, Q. Z., AND DANIEL, F. *Web services foundations*. Springer, 2014.

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

[10] CHA, S.-J., HWANG, Y.-Y., CHANG, Y.-S., KIM, K.-O., AND LEE, K.-C. Design and evaluation of experiment methods for improving performance in gis web services. *International Journal of Multimedia and Ubiquitous Engineering 3*, 1 (2008), 27–43.

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

[12] 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.

[13] Coello, C. A. C. Evolutionary multi-objective optimization: a historical view of the field. *Computational Intelligence Magazine, IEEE 1*, 1 (2006), 28–36.

[14] Coello, C. A. C., and Christiansen, A. D. Two new ga-based methods for multiobjective optimization. *Civil Engineering Systems 15*, 3 (1998), 207–243.

[15] 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.

[16] 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* (2008), SDSOA '08, ACM, pp. 25–28.

[17] De Jong, K. A. Are genetic algorithms function optimizers? In *PPSN* (1992), vol. 2, pp. 3–14.

[18] De Jong, K. A. Genetic algorithms are not function optimizers. *Foundations of genetic algorithms 2* (1993), 5–17.

[19] 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.

[20] 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.

[21] Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., and Govindan, R. Reducing web latency: The virtue of gentle aggression. *SIGCOMM Comput. Commun. Rev. 43*, 4 (Aug. 2013), 159–170.

[22] Fogel, L. J. Autonomous automata. *Industrial Research 4*, 2 (1962), 14–19.

[23] Godinez, A. C., Espinosa, L. E. M., and Montes, E. M. An experimental comparison of multiobjective algorithms: Nsga-ii and omopso. In *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010* (2010), IEEE, pp. 28–33.

[24] Goldberg, D. E., and Holland, J. H. Genetic algorithms and machine learning. *Machine learning 3*, 2 (1988), 95–99.

[25] 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* (2010), Association for Computing Machinery, Inc.

[26] Hassan, R., Cohanim, B., De Weck, O., and Venter, G. A comparison of particle swarm optimization and the genetic algorithm. In *Proceedings of the 1st AIAA multidisciplinary design optimization specialist conference* (2005), pp. 1–13.

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

[28] 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* (2013), IMC '13, ACM, pp. 177–190.

[29] HOLLAND, J. H. Outline for a logical theory of adaptive systems. *Journal of the ACM (JACM) 9*, 3 (1962), 297–314.

[30] 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.

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

[32] 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.

[33] 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.

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

[35] 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.

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

[37] 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* (1997), vol. 5, pp. 4104–4108 vol.5.

[38] 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* (2013), pp. 2496–2503.

[39] KLOCKGETHER, J., AND SCHWEFEL, H.-P. Two-phase nozzle and hollow core jet experiments. In *Proc. 11th Symp. Engineering Aspects of Magnetohydrodynamics* (1970), Pasadena, CA: California Institute of Technology, pp. 141–148.

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

[41] KRITIKOS, K., PERNICI, B., PLEBANI, P., CAPPIELLO, C., COMUZZI, M., BENRERNOU, S., BRANDIC, I., KERTÉSZ, A., PARKIN, M., AND CARRO, M. A survey on service quality description. *ACM Comput. Surv. 46*, 1 (July 2013), 1:1–1:58.

[42] 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* (2012), pp. 841–844.

[43] 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.

[44] Li, X. A non-dominated sorting particle swarm optimizer for multiobjective optimization. In *Genetic and Evolutionary Computation GECCO 2003*, vol. 2723 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 37–48.

[45] 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.

[46] Olivas, E. S. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques*. IGI Global, 2009.

[47] Organization for the Advancement of Structured Information Standards (OASIS). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*, 2007.

[48] Papazoglou, M. P., and Heuvel, W.-J. Service oriented architectures: Approaches, technologies and research issues. *The VLDB Journal 16*, 3 (2007), 389–415.

[49] 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* (2012), GECCO '12, ACM, pp. 19–26.

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

[51] 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* (2005), GECCO '05, ACM, pp. 257–264.

[52] Rechenberg, I. *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. 1971.

[53] Schien, D., Shabajee, P., Wood, S. G., and Preist, C. A model for green design of online news media services. In *Proceedings of the 22Nd International Conference on World Wide Web* (Republic and Canton of Geneva, Switzerland, 2013), WWW '13, International World Wide Web Conferences Steering Committee, pp. 1111–1122.

[54] Schwefel, H.-P. *Evolutionsstrategie und numerische Optimierung*. Technische Universität Berlin, 1975.

[55] Sun, Y. *A Location Model for Web Services Intermediaries*. PhD thesis, 2003. AAI3120151.

[56] Sun, Y., and Koehler, G. J. A location model for a web service intermediary. *Decis. Support Syst. 42*, 1 (2006), 221–236.

[57] 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.

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

[59] 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* (2005), pp. 22–29.

[60] 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.

[61] 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.

[62] 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.