Evolutionary Multi-Objective Optimization for Web Service Location Allocation Problem

Boxiong Tan, Hui Ma, Yi Mei, and Mengjie Zhang

Abstract—With the ever increasing number of functionally similar web services being available on the Internet, the market competition is becoming intense. Web service providers (WSPs) realize 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 location, a straightforward way to reduce network latency is to allocate services to proper locations. However, Web Service Location Allocation Problem (WSLAP) is a challenging task since there are multiple objectives potentially conflicting with each other and the solution search space has a combinatorial nature. In this paper, we consider minimizing the network latency and total cost simultaneously and model the WSLAP as a multi-objective optimization problem. We develop a new PSO-based algorithm to provide a set of trade-off solutions. The results how that the new algorithm can provide a diverse set of solutions than the compared three well known multi-objective optimization algorithms. Moreover, the new algorithm performs better especially on large problems.

Index Terms—Web Service Location Allocation, Quality of Service, Evolutionary Computation, Particle Swarm Optimization.

1 Introduction

Recently, companies have been employing service-oriented computing (SOC) developed softwares in an agile and cost efficient way [1]. As the building blocks of SOC, Web services are well-defined, self-contained modules that provide standard business functionality and customers can access them via the Internet [2]. Web service providers always face a tradeoff while trying to improve the Quality of Service (QoS) [3] (e.g. service response time) and reduce the total cost. This study proposes a new algorithm to solve cost–QoS trade-off. Specifically, we use response time as the QoS criteria. Service providers can choose the most suitable allocation according to their budget and required service response time.

Web service allocation affects both service response time and total cost. Deploying services to user concentrated locations improves service response time largely because it reduces network latency [4]. On the other hand, deploying services at multiple locations invariably raised the total cost. Hence, the Web Service Location Allocation Problem (WSLAP) is essentially a multi-objective optimization problem [5] with two conflicting objectives, minimizing the response time and minimizing the total cost. Therefore, solutions of WSLAP have no single global optimum but a set of Pareto-optimal solutions. Although much research [6] discusses location-awareness, WSLAP has not been well studied.

WSLAP is an extension of P median problem [7] that has been proved as an NP-hard problem. WSLAP has a combinatorial search space. For example, to deploy 8 services at 4 locations. Each service can be deployed at multiple locations. Then, the number of deployment solution is $2^{8\times4}$. It is difficult for exhaustive algorithms to find optimal solutions in such a huge search space.

Current studies have two drawbacks. First, Aboolian and Sun [8], [9] treat the WSLAP as a single-objective problem and use traditional approaches, e.g. integer linear programming (ILP), to solve the problem. The algorithm generates only one solution based on the assumption that we know service providers' budget and required response time. Second, their methods do not scale [10].

Multi-objective evolutionary algorithms (MOEAs) can address the above drawbacks effectively. First, MOEAs maintain a population of solutions during the search, and thus are able to obtain a set of non-dominated solutions in a single run. Second, MOEAs scale much better than ILPs due to the nature of heuristic search.

To address WSLAP as a multi-objective problem, we first proposed an aggregation approach with Binary Particle Swarm Optimization (BPSO) [11]. Even though the BPSO approach can solve the problem, a single-objective algorithm can only provide one solution for each run, hence, BPSO cannot provide alternatives when service providers do not have preferences. Therefore, we further investigated a Pareto front approach [12] with Sorting Genetic Algorithm-II (NSGA-II).

Solutions from BPSO and NSGA-II suggest that the proposed multi-objective approaches suit the problem well. This is because BPSO and NSGA-II are population-based algorithms. Multi-objective Optimization Evolutionary Algorithm (MOEA) methodologies are ideal [13] for solving combinatorial optimization problems [14], [15].

However, we also found a disadvantage in both BPSO and NSGA-II. Their performances (in terms of convergence) drop rapidly when the problem size increases. In addition, the diversity of solutions is not maintained when problem size grows.

Hence this study focuses on how to maintain the performance when problem size increases. We consider a Multi-Objective Particle Swarm Optimization with Crowding Dis-

• *School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand. E-mail: {boxiong.tan, hui.ma, yi.mei, mengjie.zhang}@ecs.vuw.ac.nz*

Manuscript received x x, x; revised x x, x.

tance (MOPSOCD). Raquel et al [16] propose MOPSOCD which can produce a well-distributed set of non-dominated solutions. MOPSOCD has two desired features: *an external archive set* and *a mutation operator*. These two features enhance the ability to avoid being stuck at local optima and maintaining a distributed non-dominated set.

However, the major obstacle is that MOPSOCD is a continuous optimization algorithm whereas WSLAP is a binary problem. Therefore, we develop a new binary version - BMOPSOCD. To this end, we use *rounding functions* to transform continuous values into binary ones. We propose three types of rounding functions and study their effects.

The overall goal is to develop a new binary multi-objective PSO-based approach to the WSLAP by considering two potentially conflicting objectives - minimizing cost and minimizing network latency. More specifically, we have the following objectives:

1) To design rounding functions for transforming continuous PSO to binary PSO;
2) To develop a new multi-objective PSO approach that can produce a set of solutions with good diversity and can perform well when problem sizes increase;
3) To evaluate our proposed approach by comparing it with previous approaches on benchmark datasets.

2 Background

2.1 Multi-Objective Optimization

A multi-objective optimization problem consists of multiple objective functions. It can be stated as follows:

$$\min \ \vec{f}(\vec{x}) = (f_1(\vec{x}), \ldots, f_m(\vec{x})), \quad (1)$$

$$s.t. \ \vec{x} \in \Omega. \quad (2)$$

where Ω stands for the feasible region of $\vec{x}$.

The objective functions $(f_1(\vec{x}), \ldots, f_m(\vec{x}))$ are assumed to be conflicting, i.e. No single solution can achieve the optimal value for all the objective functions. In this case, the goal is to find a set of *Pareto-optimal* solutions. First, we introduce the concept of the *dominance relation* between solutions in multi-objective optimization.

Definition 1 (Dominance relation 1). A solution $\vec{x}_1$ is said to *dominate* another solution $\vec{x}_2$ if

1) for all $k \in \{1 \ldots, m\}$, $f_k(\vec{x}_1) \leq f_k(\vec{x}_2)$ and
2) there exists at least one $k \in \{1 \ldots, m\}$, so that $f_k(\vec{x}_1) < f_k(\vec{x}_2)$.

Definition 2 (Dominance relation 2). Two solutions $\vec{x}_1$ and $\vec{x}_2$ are said to be *non-dominated* to each other, if *neither* $\vec{x}_1$ *dominates* $\vec{x}_2$, *nor* $\vec{x}_2$ *dominates* $\vec{x}_1$.

Then, the Pareto optimality is defined as follows.

Definition 3 (Pareto optimality). A solution $\vec{x}^*$ is a *Pareto optimal solution*, if it is not dominated by any $\vec{x} \in \Omega$.

To measure the performance of a multi-objective optimization algorithm, three aspects need to be considered [17], [18]: convergence, diversity and the number of solutions. Convergence denotes the closeness between a non-dominated solution set to the theoretical Pareto front. Diversity denotes the distribution and spread of a non-dominated solution set.

2.2 Evolutionary Multi-objective Optimization

The Evolutionary Computation (EC) method is good at solving multi-objective optimization problems due to its capability of maintaining a set of solutions in its population. It finds a set of Pareto-optimal solutions in a single run. Evolutionary Multi-objective Optimization (EMO) has been extensively studied, and numerous EMO algorithms have been proposed.

NSGA-II [19] is a widely used multi-objective algorithm. NSGA-II proposed two innovative approaches: fast non-dominated sorting and crowding distance comparison-based diversity preservation. Strength Pareto Evolutionary Algorithm (SPEA2) [20] uses an external archive to store non-dominated solutions and a nearest neighbor density estimation technique to guide the search process. MOEA/D [21] decomposes a multi-objective problem into a number of scalar optimization subproblems and optimizes them simultaneously.

2.3 Particle Swarm Optimization (PSO)

Kennedy and Eberhart proposed PSO in 1995 [22]. It is a meta-heuristic algorithm inspired by the social behavior of birds. In PSO, each individual — a particle — searches the solution space. The underlying phenomenon of PSO is to use the collective knowledge to find the optimal solution.

At the initial state, each particle has a random initial position in the search space which is represented by a vector $\vec{x}_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$, where D is the dimension of the search space. A particle has a velocity as $\vec{v}_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$. The velocity is limited by a threshold $v_{\max}$ so that for any i and d, $v_{id} \in [-v_{\max}, v_{\max}]$. During the search process, each particle maintains a record of its best position so far, called the *personal best* (*pbest*). The best positions among all the personal best positions of its neighbors is the *global best* (*gbest*). The position and velocity of each particle are updated according to the following equations:

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}, \quad (3)$$

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 \cdot r_{1i} \cdot (p_{id} - x_{id}^{t}) + c_2 \cdot r_{2i} \cdot (p_{pg} - x_{id}^{i}). \quad (4)$$

Here, t is the index of iteration, d is the index of dimension. The inertia weight w is used to balance the local and global search abilities. The parameters c_1 and c_2 are the acceleration constants. r_{1i} and r_{2i} are random constants following the uniform distribution in the interval $[0, 1]$. p_{id} and p_{gd} denote the values of $pbest$ and $gbest$ in the d^{th} dimension of the i^{th} particle.

PSO was initiated to solve continuous optimization problems: position update function has been developed for real numbers. To address discrete problems, Kennedy and Eberhart developed a binary PSO [23]. We applied BPSO in WSLAP in previous study [11] and used the linear aggregation (weighted sum) approach to transform the multiple objective values into a single aggregated value. BPSO can provide a single solution and therefore it is used in the scenarios where a service provider knows their budget and required service response time.

Several multi-objective optimization algorithms are based on PSO such as Multi-Objective PSO (MOPSO) [24], and Non-Dominated Sorting PSO (NSPSO) [25]. Coello et

al [24] studied several multi-objective algorithms, NSGA-II, Micro-GA [26] and MOPSO, and shows that MOPSO is the most capable of generating the best set of non-dominated solutions close to the true Pareto front with low computational cost. To improve the diversity of non-dominated solutions, Raquel et al. [16] propose an MOPSOCD extended from the MOPSO. They select the global best from an external archive set. The selection is based on crowding distance values (larger the better). MOPSOCD is able to generate a well-distributed set of non-dominated solutions, in this paper, we develop a binary version of MOPSOCD to solve the WSLAP.

3 Problem Description

We consider the WSLAP as a multi-objective problem with two conflicting objectives, minimizing the total deployment cost and minimizing the network latency. This section explains symbols and variables of the problem.

We consider a set of user centers $\mathcal{U} = \{U_1, \ldots, U_m\}$ and a set of candidate locations $\mathcal{A} = \{A_1, \ldots, A_n\}$. A user center denotes the center of a user-centered area which allows latency estimation between user groups and services. Candidate locations are locations that are suitable to deploy web services, e.g., locations of datacenters. A service provider needs to deploy a set of web services $\mathcal{W} = \{W_1, \ldots, W_s\}$. Each service needs to be deployed to at least one location. One can deploy duplicated copies of the same Web service to multiple locations in order to reduce the service response time. For each web service $W_i \in \mathcal{W}$ and each candidate location $A_j \in A$, there is a deployment cost C_{ij} induced by deploying service W_i to candidate location A_j. Service invocation frequency F_{ki} denotes service invocation frequencies from user centers U_k to services W_i over a unit period of time. In reality, service request frequency patterns may change over time at a steady pace (e.g. in periods of month). Therefore, we consider WSLAP as an off-line problem and assume services' frequencies are fixed [27]. This study takes an average number of invocations over a period of time to represent the frequency. This way, the problem is simplified and the research is focused on developing algorithms to search effectively in the complex search space. For each user center $U_k \in \mathcal{U}$ and each candidate location $A_j \in \mathcal{A}$, a latency L_{kj} represents the response time from the location A_j to the user center U_k. Because the networking topology is complex, we treat it as a black box. We use a *service location matrix* $X = [X_{ij}]$ to represent the location allocation plan, where X_{ij} represents whether a service W_i is deployed at a candidate location A_j. Web Service Location Allocation allocates a set of services $\mathcal{W} = \{W_1, W_2, \ldots, W_s\}$ to a set of candidate locations $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ so that the total deployment cost f_1 and response time f_2 are minimized. Total deployment cost f_1 and total network latency f_2 can be calculated as follows:

$$f_1 = \sum_{i=1}^{s}\sum_{j=1}^{n} C_{ij}X_{ij}, \tag{5}$$

$$f_2 = \sum_{k=1}^{m}\sum_{i=1}^{s} F_{ki}R_{ki}, \tag{6}$$

where X_{ij} takes 1 if service W_i is allocated to location A_j, and 0 otherwise. R_{ki} stands for the shortest response time of accessing service W_i from user center U_k, which is calculated as

$$R_{ki} = \min\{L_{kj} \mid j \in \{1, 2, ..., n\} \text{ and } X_{ij} = 1\} \tag{7}$$

Here, R_{ki} is represented by the smallest latency L_{kj} choose from deployed locations.

WSLAP has the following two objective functions and one constraint.

$$\text{minimize} \quad f_1 = \sum_{i=1}^{s}\sum_{j=1}^{n} C_{ij}X_{ij}, \tag{8}$$

$$\text{minimize} \quad f_2 = \sum_{k=1}^{m}\sum_{i=1}^{s} F_{ki}R_{ki}, \tag{9}$$

$$\begin{aligned} \text{subject to} \quad & \sum_{j=1}^{n} X_{ij} \geqslant 1, \forall i \in 1, \cdots, s \\ & x_{ij} \in 0, 1, \forall i \in 1, \cdots, s, \forall j \in 1, \cdots, n, \end{aligned} \tag{10}$$

To solve the WSLAP we aim to find an allocation matrix $X = [X_{ij}]$ such that it results in minimal overall network latency and overall deployment cost.

For example, assume we are given the service invocation frequency matrix F, latency matrix L, deployment cost matrix C, and the allocation matrix X as follows, we will show how to calculate the overall deployment cost and network latency.

$$F = \begin{array}{c} \\ U_1 \\ U_2 \\ U_3 \end{array} \begin{array}{c} \begin{array}{ccc} W_1 & W_2 & W_3 \end{array} \\ \begin{pmatrix} 150 & 83 & 40 \\ 36 & 42 & 13 \\ 72 & 65 & 9 \end{pmatrix} \end{array}, \quad L = \begin{array}{c} \\ U_1 \\ U_2 \\ U_3 \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & A_2 & A_3 \end{array} \\ \begin{pmatrix} 0 & 2.8 & 7.6 \\ 2.8 & 0 & 3.1 \\ 0.8 & 1.2 & 3.4 \end{pmatrix} \end{array}$$

$$C = \begin{array}{c} \\ W_1 \\ W_2 \\ W_3 \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & A_2 & A_3 \end{array} \\ \begin{pmatrix} 120 & 95 & 60 \\ 37 & 43 & 65 \\ 68 & 27 & 46 \end{pmatrix} \end{array}, \quad X = \begin{array}{c} \\ W_1 \\ W_2 \\ W_3 \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & A_2 & A_3 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \end{array}$$

The calculation of the overall deployment cost sums up the deployment costs of all the deployments.

$$\begin{aligned} f_1 &= \sum_{1=1}^{s}\sum_{j=1}^{n} C_{ij}X_{ij} \\ &= C_{11}X_{11} + C_{12}X_{12} + C_{13}X_{13} + ... + C_{33}X_{33} \\ &= 120 + 43 + 27 + 46 \\ &= 236 \end{aligned}$$

To calculate the overall network latency, we need to first calculate the response time matrix R using the matrices L and X. For each service W_i, by checking matrix X, we can find out which locations the service has been deployed in. Then, we check matrix L to find out the corresponding latency from each deployed location A_j to each user center U_k. Finally, the smallest latency is selected to be the response

time from each service W_i to each user center U_k. In the above example, the corresponding response matrix is

$$R = \begin{array}{c} \\ U_1 \\ U_2 \\ U_3 \end{array} \begin{array}{c} \begin{array}{ccc} W_1 & W_2 & W_3 \end{array} \\ \begin{pmatrix} 0 & 2.8 & 2.8 \\ 2.8 & 0 & 0 \\ 0.8 & 1.2 & 1.2 \end{pmatrix} \end{array}$$

Finally, we can calculate the overall network latency as

$$\begin{aligned} f_2 &= \sum_{k=1}^{m}\sum_{i=1}^{s} F_{ki}R_{ki} \\ &= F_{11}R_{11} + F_{12}R_{12} + F_{13}R_{13} + ... + F_{33}R_{33} \\ &= 150 * 0 + 83 * 2.8 + 40 * 2.8 + \cdots + 9 * 1.2 \\ &= 591.6 \end{aligned}$$

The constraint requires a web service is at least deployed to one location. The example matrix X above satisfies the constraint.

4 BMOPSOCD for Web Service Location Allocation Problem

This section presents the proposed BMOPSOCD. We first define the representation of the problem followed by rounding methods that can be used to transform a continuous representation into a binary form. We then present fitness functions and constraint handling method.

4.1 Particle Representation

A solution for WSLAP is a $s \times n$ allocation matrix $X = [X_{ij}]$. We need to transform the $s \times n$ matrix into a vector y in order to use PSO to generate vector-based solutions. The element X_{ij} in X corresponds to the $(n \cdot (i-1)+j)^{th}$ element in Y. Also, for continuous PSO, each element of a particle takes value from 0 to 1, i.e., $0 \leq Y_u \leq 1$. For example, the following 3×3 matrix

$$X = \begin{array}{c} \\ W_1 \\ W_2 \\ W_3 \end{array} \begin{array}{c} \begin{array}{ccc} A_1 & A_2 & A_3 \end{array} \\ \begin{pmatrix} 0.12 & 0.87 & 0.42 \\ 0.07 & 0.32 & 0.95 \\ 0.76 & 0.64 & 0.27 \end{pmatrix} \end{array}$$

can be transformed into a vector:

$$Y = [0.12, 0.87, 0.42, 0.07, 0.32, 0.95, 0.76, 0.64, 0.27].$$

To transform continuous particle representation to binary form, rounding functions can be applied. The choice of the rounding function plays an important role in the quality of the final results. The following sections discuss different rounding methods. Note that the vector Y is used in the evolution of PSO. During the fitness evaluation phase, each Y is first decoded into a matrix X. Then, a rounding function is used to round X into a binary representation. At last, the evaluation is processed on binary solutions.

4.2 Rounding Functions

The original MOPSOCD is designed as a continuous version of PSO. Instead of changing the particle to a binary representation, we still use the continuous representation. This is because the binary position updating function is frequently criticized for being independent between the current value and the next value [28]. However, the position updating of continuous PSO is a well-defined mechanism. Therefore, we keep the continuous representation and related evolutionary operators. Furthermore, we explore using rounding functions as a mechanism of transferring a continuous representation to a binary representation.

At the initial stage, particles are initialized in real values as before. The updates of velocity and position are performed as usual. At the evaluation stage, particles in continuous representation need to be transformed to binary representation, and then to be evaluated by fitness functions. A particle does not change after evaluation; only its "binary representation" is evaluated.

The rounding function maps real values to binary values. The common strategy is to round a real value to its closest integer number. A round-down strategy is adopted in Laskari [29] to solve integer programming problem. Xue [30] defines a static rounding function for feature selection problem. Whether a feature is selected or not is determined by a predefined static threshold θ. Haupt et al [31] use a real-valued representation of chromosome for GA. Then, a real-valued chromosome is rounded to an integer representation. Nonetheless, there is no thorough study of how to choose rounding functions or a comparison between different rounding functions. Hence, we propose three types of rounding functions and study their effects.

4.2.1 Static Rounding Function

The static rounding function is shown in Equation 11: a parameter threshold θ is introduced to round up or round down a particle entry X'_{ij}. The threshold value θ is ad-hoc and is usually set based on empirical study.

$$X_{ij} = \begin{cases} 1 & \text{if } X'_{ij} > \theta \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

4.2.2 Dynamic Rounding Functions

The threshold plays an important role in searching for solutions for a given problem [11]. The static rounding function has the following drawbacks. Firstly, it needs to be predefined and problem specific. It is hard to estimate the effect of θ before observing results. Secondly, the influence of different threshold values is not completely studied. Therefore, we propose dynamic rounding functions have two steps. First, the function adjusts the value of threshold θ according to two predefined parameters: a lower bound l and an upper bound u of the threshold ($l < u$), and a dynamic parameter: the current generation t. The second step either rounds up or down the value of X_{ij} according to θ. Three dynamic rounding functions: *linear function*, *Quadratic function* and *Reciprocal function* are considered. Specifically, current generation t cannot equal max_gen in order to keep the result valid.

$$\theta_{linear} = \frac{l-u}{\text{max\_gen}}t + u \tag{12}$$

$$\theta_{quad} = \frac{l-u}{(\text{max\_gen})^2}t^2 + u \tag{13}$$

$$\theta_{recip} = u - \frac{u-l}{\text{max\_gen} - t}(t \neq \text{max\_gen}) \quad (14)$$

The reason we design three dynamic functions is to compare the impact of different trajectories of dynamic thresholds (see Figure 1. The threshold values with a Linear function are uniformly distributed, whereas Reciprocal and Quadratic functions are unevenly distributed. The performances of these rounding functions are studied in Section 5.5.4.

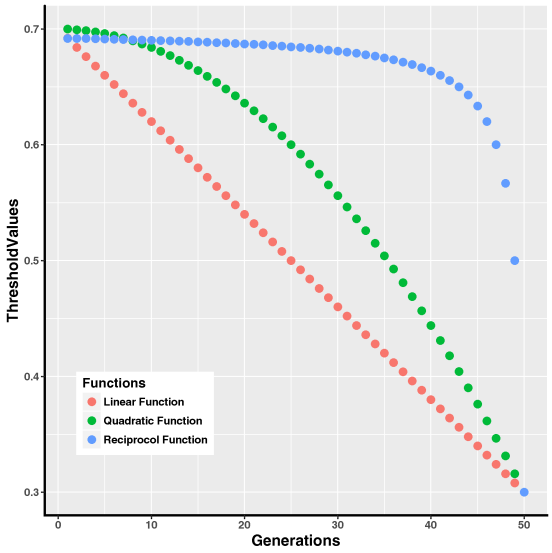


Fig. 1: Trajectories of the three dynamic thresholds

4.2.3 *Stepped Rounding Function*

As the dimensionality of the problem increases, the performance of evolutionary computation drops. It is necessary to build a system that can reuse the learned knowledge. Transfer learning is a process to reuse the knowledge in solving unseen tasks [32]. This section proposes a Stepped rounding function that is embodied in the transfer learning process.

Figure 2 shows the evolutionary process with a Stepped rounding function. Initially, the threshold θ is set to an upper bound u (e.g. 0.7). Then the PSO runs with this setting for a predefined interval of i (e.g. 10) generations. At the beginning of the next interval (e.g. 11) generation, the threshold θ is changed according to Equation 15 and remains until next interval. This process is repeated until the lower bound l is reached. The optimization is equivalent to initializing a new set of a population with the old one. Therefore, the knowledge is inherited. An underlying assumption is that, if the particle swarm has converged within an interval, then it is better to explore a different direction. Therefore, in the next interval, the swarm will explore a slightly different area and is directed by an adjacent threshold value. The Stepped rounding function is shown in Equation 15 where θ' denotes the current threshold value.

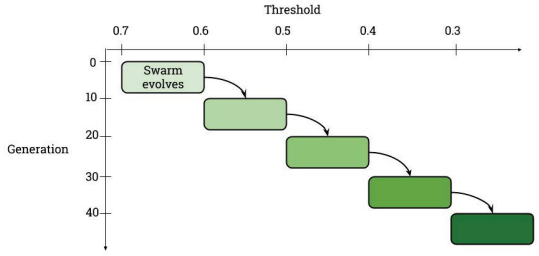


Fig. 2: Evolutionary process with a Stepped rounding function

$$\theta = \begin{cases} \theta' - \frac{u-l}{(\text{max\_gen}/i-1)} & \text{if (cur\_gen} \mod i) = 0 \\ \theta' & \text{otherwise} \end{cases} \quad (15)$$

4.3 Fitness Function and Constraint Handling

After the particles represented as vectors have been transformed to allocation matrices, two fitness functions, Equation 5 and 6, are used to evaluate the fitness of particles that represent allocation plans. Based on the fitness of solutions a set of non-dominated solutions are returned.

As we see in Section 3, WSLAP needs to satisfy the constraint defined by Equation 10, which means that each service needs to be allocated to at least one location. However, during the searching process of PSO, the constraint may be violated. That is, some services might be allocated to none of the locations during the process of searching.

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 while solution J is not,
2) Both solutions are infeasible and solution I has fewer violations,
3) Both solutions are feasible, and solution I dominates solution J.

The particle with fewer violations is always considered as a better solution. If there is only one constraint, this constraint handling method provides the same effect as the death penalty method in [33].

4.4 The BMOPSOCD algorithm for Web Service Location Allocation Problem

Algorithm 1 presents our BMOPSOCD algorithm for solving the WSLAP. The algorithm iteratively improves results by keeping good solutions and eliminating poor solutions.

First, a population of particles is random initialized from a uniform distribution and evaluated (lines 1–5). Notice that, instead of randomly generating population, we can design an initialization operator that only generates feasible solutions. However, this is not our focus. Therefore, this work only uses random initilization.

Next, non-dominated solutions are stored in the external archive (line 6). Lines 7–20 are the evolution process, where the population evolves maximal $MAXT$ generations and t is the current generation. In each generation, solutions are ranked by their crowding distances (lines 8–9). Line 10 randomly selects a global best since they are equally good.

Lines 12–13 update position and velocity values. Line 14 guarantees a particle will not leave its domain by reversing its direction. Lines 15 conducts the mutation, where P_m is the mutation rate (see Section. 5.4). Line 16–17 round the population and evaluate them with fitness functions. Line 19 updates the best solutions in the archive.

The selection of $pbest$ and $gbest$ is one of the key steps in BMOPSOCD. $pbest$ is the personal best solution of each particle in a 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 the highest crowding distance value. It ensures that the swarm move to a less crowded area.

Algorithm 1 BMOPSOCD for WSLAP

Inputs:
Cost Matrix C,
Server network latency matrix L,
Service invocation frequency matrix F
Outputs: Pareto Front: the $Archive$ set

1: Initialize each individual X_{ij} in a Population P with a random real value $\in$ (0, 1).
2: Initialize v_i = 0
3: **For each individual** i **in** P **Rounding and Evaluating fitness**
4: Initialize $pbest$ of each individual i.
5: Initialize $gbest$
6: Initialize $Archive$ with non-dominated vectors in P
7: **repeat**
8: Compute the crowding distances of each solution i in $Archive$
9: Sort solutions in $Archive$ in descending crowding distances
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 v_i using Eq.4
13: Update its position x_i using Eq.3
14: If $X_{ij} > 1$ or $X_{ij} < 0$, set its value to 1 or 0 and multiply its velocity by -1.
15: If $(t < (MAXT * P_m))$, apply Mutation
16: **Rounding and Evaluating fitness**
17: Update its $pbest$
18: **end for**
19: Insert new non-dominated solutions into $Archive$, remove dominated solutions from $Archive$
20: t increment by 1
21: **until** t reaches $MAXT$
22: return $Archive$

The fitness of particles can be evaluated using the objective functions presented in Equation 5 and 6. Line 12 updates velocity v_i as:

$$v_{id} = w * v_{id} + c_1 * r_{1i} * (p_{id} - x_{id}) + c_2 * r_{2i} * (p_{pg} - x_{id}) \quad (16)$$

where w is the inertia weight, c_1 and c_2 are the acceleration factors, r_{1i} and r_{2i} are the random variables sampled from the uniform distribution between 0 and 1, v_{id}, x_{id}, p_{id} and g_d denote the value in dimension d of $\vec{v_i}, \vec{x_i}, \vec{p_i}$ and $\vec{g}$, respectively. Note that the main difference between our algorithm and the original [16] are in Lines 3 and 16, which is the use of rounding function. For WSLAP, the decision variable is binary. Therefore, in our algorithm (Line 3 and 16) we apply a rounding function to transform continuous values to binary values.

The mutation operator (Line 15) changes the value of each dimension of an individual according to a nonlinear function [24]. The nonlinear function is related to the total number of iteration and the current number of iteration. The mutation operator is designed to increase the diversity of the initial population.

In Line 19, dominated solutions from the *archive* are removed. For example, if there are two solutions A with cost of 560 and overall latency as 1120.5 and B with cost of 780 and overall latency as 1589.9 in the archive. In this case, solution A dominates B since A's cost and latency are better than B's. Then solution B is removed from the *archive*. At the end of the algorithm, an *archive* with no dominated solutions is returned as the best solutions found during the entire evolutionary process.

5 Experiment Design

This study proposes a multi-objective approach to WSLAP to produce well-distributed Pareto-optimal solutions. Meanwhile, the performance of the proposed algorithm performs well with the increasing size of problems. The above section presents our BMOPSOCD approach to WSLAP. We develop static and dynamic rounding methods with different threshold settings. We conduct a set of experiments over three variant of the proposed algorithm. The first variant uses the static threshold. We study the influence of different threshold values; the second variant uses the dynamic rounding functions. We examine the effect of three dynamic rounding functions; The third variant is the Stepped rounding function. We compare its results with dynamic functions. In addition, we make a comparison between a dynamic function with a congregation of several static rounding functions with different thresholds. Lastly, we conduct an experiment considering the overall performance of a BMOPSOCD with a dynamic rounding function in comparison with three other algorithms: PSO, BNSPSO and NSGA-II (see [12] for details).

All algorithms were implemented in R version 3.3.3 and the experiments were conducted on an i7-4790 3.6 GHz with 8GB of RAM memory running Linux Arch 4.9.6.

5.1 Comparison Design

We use different ways to compare three variants of BMOPSOCD, and BMOPSOCD with other algorithms (e.g NSGA-II). The major difference is that we compare three variants with the solutions from a single run while we compare BMOPSOCD and other algorithms with their best solutions from 40 runs.

Explicitly, for comparing three variants of BMOPSOCD, we first find out the median hypervolume value (explained in Section 5.3) in 40 runs. Then we find out in which run that the variant of BMOPSOCD produces the median hypervolume value. Finally, we plot the solutions from that run in terms of two objectives fitness values.

For comparing BMOPSOCD and other algorithms, we first aggregate the solutions from 40 runs and then apply an non-dominated sorting on the aggregated results to find the best results from that algorithm. Finally, we plot the solutions.

All fitness values are normalized between 0 and 1 with linear normalization method described in [12]. The normalization method requires the maximum and minimum

values of cost and response time that we found during optimization.

We can obtained these extreme values with the following method. We collect the solutions from all four algorithms :NSGA-II, BPSO, NSPSO, and BMOPSOCD on the same test instances. Then, we applied non-dominated sorting on these solutions to obtain the best solutions. The third step, we can find the extreme values of cost and response time from the best solutions.

5.2 Test instances

We design 14 test instances in terms of increasing size of search space (see Table 1). The instances were generated randomly, and thus were not biased to any algorithm. The search space is decided by the number of services and candidate locations (e.g $2^{service \cdot candidate}$). Because the computation complexity is not solely determined by the search space of test instances, from instance 3 onwards, for each search space, we vary the number of user centers. The purpose is to test which factor (Number of services, candidate locations, and user centers) contributes to the computation complexity the most.

We construct test instances based on real-world WS-Dream dataset [34]. The data contains only the latencies between candidate locations and user centers, but lacks the deployment costs for candidate locations and invocation frequencies for web services. Therefore, to generate a complete WSLAP instance, we randomly generated the deployment costs for candidate locations according to a normal distribution with the mean of 100 and standard deviation of 20. We generate the invocation frequency from a uniform distribution between 1 and 120.

TABLE 1: Test instances

Test instances	No. of Services	No. of Candidate Locations	No. of user centers	Size of search space
Instance 1	20	5	10	2^{100}
Instance 2	20	10	10	2^{200}
Instance 3	50	15	20	2^{750}
Instance 4	50	15	40	2^{750}
Instance 5	50	25	20	2^{1250}
Instance 6	50	25	40	2^{1250}
Instance 7	100	15	20	2^{1500}
Instance 8	100	15	40	2^{1500}
Instance 9	100	25	20	2^{2500}
Instance 10	100	25	40	2^{2500}
Instance 11	200	25	40	2^{5000}
Instance 12	200	25	80	2^{5000}
Instance 13	200	40	40	2^{8000}
Instance 14	200	40	80	2^{8000}

5.3 Performance Metrics

We use HyperVolume (HV) and IGD as the evaluation metrics. HyperVolume indicator [18], [35] is a measure used in evolutionary multi-objective optimization, which reflects the volume enclosed by a solution set and a reference point. A larger HyperVolume value indicates a better solution set. The IGD [36] is a modified version of generational distance [37] as a way of estimating how far the elements in the true Pareto front are from those in the non-dominated set produced by an algorithm. IGD calculates the sum of the distances from each point from the true Pareto front to the nearest point from the non-dominated set produced by an algorithm. The lower the IGD, the better quality the solution is. Calculating the IGD value needs a true Pareto front. For our problem, the true Pareto front is unknown. Therefore, an approximated Pareto front is produced by combining all the solutions produced by the four compared algorithms (BMOPSOCD, NSGA-II, BNSPSO, BPSO) and then applying a non-dominated sorting to obtain the final non-dominated set. The approximated Pareto front dominates all the other solutions we found.

TABLE 2: Parameter Settings for BMOPSOCD

Parameter	value	Parameter	value
w	0.4 [38]	population size	250
P_m	0.5 [16]	archive size	250
c_1	1 [16]	iteration	50
c_2	1 [16]	HV reference point	(1, 1)

5.4 Parameter Settings

The parameters of the PSO algorithms are set as follows. We apply static inertia weight and set w to 0.4. Although it is found in [38] that inertia weight w decreasing from 0.9 to 0.4 provides excellent results, this work only focuses on rounding functions. Therefore, the influence of dynamic w has to be eliminated. Mutation probability is in range of [0, 1]. This study follows the original paper's [16] and set P_m to 0.5. The cognitive parameter and social parameter c_1 and c_2 are generally selected from the range of [0, 4]. This study sets both parameters to 1 [16] so that cognitive and social parameter have an equal influence on the swarm. The archive size and a population size are set to 250 and 50 respectively according to the original paper's [16] recommendation. Generally, the population size has a relation with the search space. A larger population size denotes a stronger search ability because more space are explored [39]. Our study focuses on the binary PSO technique instead of selecting the best parameter set. Therefore an empirical value is selected and the maximal number of iteration is set to 50 in all experiments. The "empirical value" means we try several values for the number of generation and see whether the solutions has converged (has little changes of fitness values between two consecutive generations). Since the computation is time consuming, we choose a small number of generation for all algorithms for fair comparison.

We use (1, 1) as the reference point in Hypervolume. Because we normalize two objectives between 0 and 1, reference point (1, 1) is the extreme point of objective values. Moreover, we do not have particular preference to objectives. Therefore, a balanced point (1, 1) is suitable for the reference point.

For each experiment, the proposed algorithm was run for 40 times independently. The best results of all the runs are compared. To obtain the *best result* of 40 runs, the results of all 40 runs are combined and sorted by the fast non-dominated sorting [19].

5.5 Experiments on Rounding Functions

This section designs four experiments to study the effect of different types of rounding functions. We applied Wilcoxon signed rank test on all results between variants of BMOP-SOCD. We show four test instances (Instances $2 \sim 5$) because we observe a similar pattern throughout all test instances. Therefore, we choose these four representative instances to show the results.

5.5.1 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*. We conduct several experiments to evaluate the performance of various threshold values ranging from 0 to 1 divided equally into 10 segments.

5.5.2 Result Analysis of Static Rounding Function

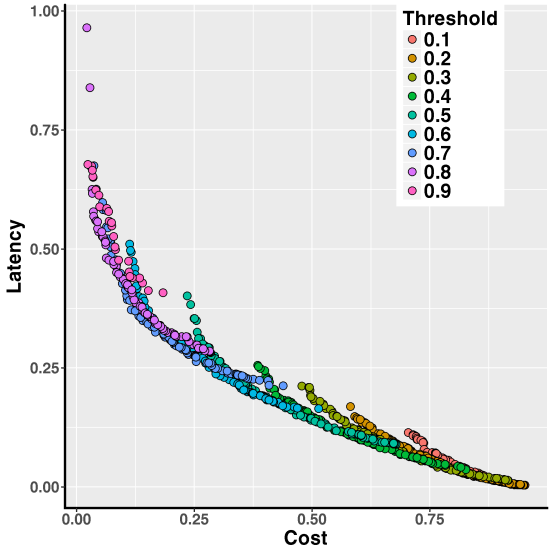


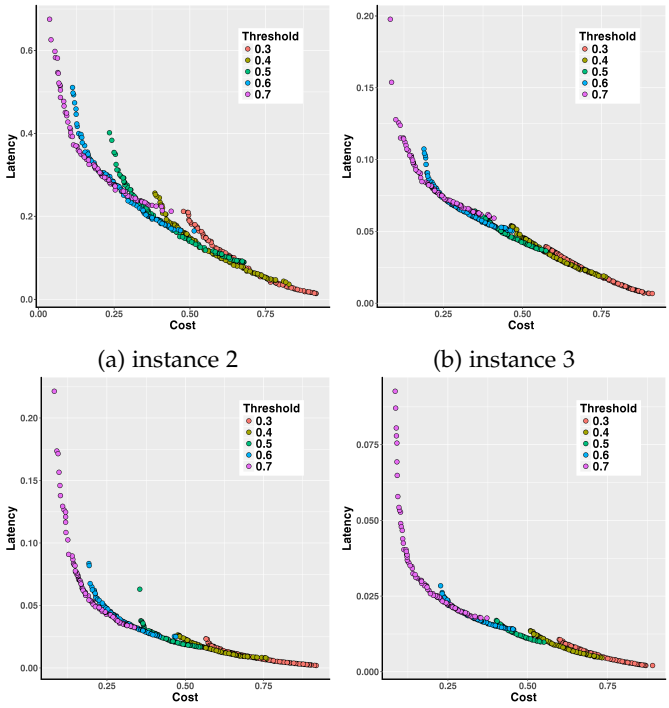
Fig. 3: Solve Test instance 2 with Static thresholds ranging from 0.1 to 0.9

A representative result is given in Figure 3, where each color represents a threshold value and each point denotes a service allocation solution in the 2D objective space. It shows that in the BMOPSOCD with a different static threshold value will lead to solutions covering a different partial region of the Pareto front.

The impact of a static threshold can be explained as a preference factor between two objectives. Take the threshold value of 0.7 as an example, when the rounding function rounds a real value to a binary value with the threshold of 0.7, it means that 70% of probability rounds this value to zero. It denotes that there are 70% of chance does not deploy a web service in a location. Intuitively, the optimization with threshold of 0.7 would consider optimizing cost over latency by deploying fewer web services. The swarm is under the guidance of a "savings" policy.

Another important point is that, the solutions generated using various thresholds, from 0.3 to 0.7, almost cover the entire Pareto front (with the zero cost-fitness eliminated because of the service number constraint). That is, the solutions from thresholds under 0.3 and greater than 0.7 have trivial improvement for a final solution set 4. Therefore, 0.3 and 0.7 can be set as the lower and upper bound of a variant threshold without losing many solutions.

The experimental results answered the first question: *the influence of different thresholds*. However, it does not offer a guideline for selection of a proper threshold, because none of the solutions could cover the entire Pareto front. The combination of all the results boosts the diversity of the non-dominated set. However, repeating the experiment with different threshold values is time consuming. It is necessary to discover a method with widely spread solutions as well as a low computational complexity. The observation of thresholds inspires the development of the dynamic rounding functions.



(a) instance 2 (b) instance 3

(c) instance 4 (d) instance 5

Fig. 4: Static Rounding Function Experiments : Solutions for Test instances 2 to 5 with different static threshold values

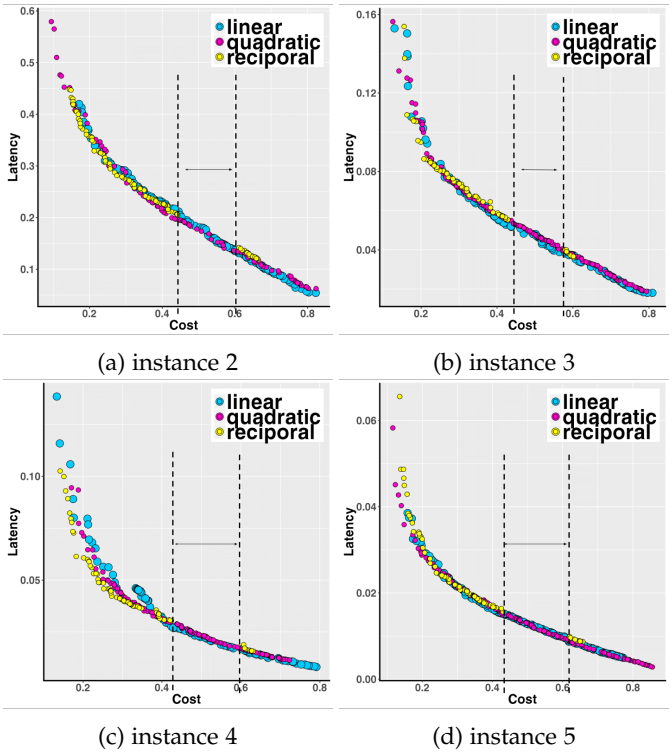
5.5.3 Dynamic Rounding Function

In Section 4.2.2, three dynamic rounding functions are proposed. In this section, we evaluate the performances of these three rounding functions to find out which dynamic rounding function provides the best results. The upper bound of dynamic threshold u is set to 0.7 and the lower bound of dynamic threshold l is set to 0.3. The results are compared in terms of mean IGD and HV.

5.5.4 Comparison between Dynamic Rounding Functions

Table 3 shows the average performance of the three dynamic functions that are evaluated by hypervolume. The results indicate that the Reciprocal function outperformed other two dynamic functions in all test cases. This could be visually observed by plotting the best results (Figure 5). Figure 5 shows that the Reciprocal function slightly dominates the other two dynamic functions. However, the problem of the

Reciprocal function is that it may miss some regions in the Pareto front, e.g. the regions between dashed-vertical lines of 0.45 and 0.55 in Figure 5. The gap is created because of the lack of uniformity (Figure 1).



(a) instance 2 (b) instance 3

(c) instance 4 (d) instance 5

Fig. 5: Dynamic Rounding Function Experiments : The non-dominated solutions among the sets obtained by 40 independent runs of BMOP-SOCD with different dynamic functions. The dashed-vertical lines mark the missing solutions from Reciprocal function.

TABLE 3: The mean and standard deviation of the hypervolume values over the 40 independent runs

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

The experimental results 3 show that in terms of convergence, the Reciprocal function produces the best result. However, it also shows the disadvantage of the Reciprocal function, which cannot provide a non-dominated set that covers the entire Pareto front. In contrast, the Quadratic function performs slightly worse in convergence but obtains a good-coverage non-dominated set. The Linear function is dominated by Quadratic function in both aspects.

The better convergence with the Reciprocal function can be explained, as there is a minor change in threshold in the most generations and the swarm has a longer time to search along the same direction. However, with the Linear and Quadratic function, the constant changing in direction does not give the algorithm enough time to find a good solution.

Between the Quadratic function and the Linear function, their changing rate of threshold value are different: Linear function changes equally, while Quadratic function changes little at the beginning and then become larger. As the swarm searches the low-cost region (as the threshold starts decreasing from 0.7) and moves to high-cost region, it has more time to search in the low-cost region with the Quadratic function. This is the key reason that the performance of Quadratic function is better than that of Linear function, because we have observed that the number of solutions in the low-cost region is much smaller than in the high-cost region, as shown in Figure 4, the low-cost regions are quite sparse compare with the low-latency regions. Therefore, it is reasonable to give the algorithm more time to search the low-cost region, while, in the low-latency region, the algorithm can spend much less effort because the solutions are rich. In general, algorithm with Quadratic function obtains better solutions in the low-cost region than with Linear function.

TABLE 4: A comparison between Reciprocal function and Stepped function, the mean and standard deviation of hypervolume values over the 40 independent runs

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

From table 3, in terms of hypervolume, the Reciprocal function has the best performance. However, we can observe that the Reciprocal function cannot provide a good coverage of the Pareto front in Figure 5. That is, the Reciprocal function missed many useful solutions because it lacks uniformity. Therefore, we cannot determine which algorithm is better solely depending on the hypervolume values. We conclude that the Quadratic function performances the best. Although the Quadratic function is worse than the Reciprocal function in terms of hypervolume value, the disadvantage of Reciprocal function cannot be ignored.

5.5.5 *Stepped Rounding Function*

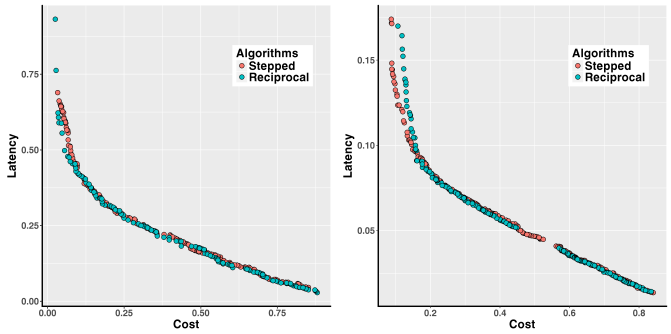
We study the performance of the Stepped rounding function in this experiment. The upper and lower bound u and l are set to 0.7 and 0.3 respectively. The interval is set to 10 generations. The results are compared with dynamic functions.

5.5.6 *Results for the Stepped Rounding function*

We compared the performance of the Stepped rounding function with the Reciprocal rounding function. The averages of HV of three rounding functions are shown in Table 4. Table 4 shows that in terms of HV, Reciprocal function performs better than the Stepped threshold approach in most cases. However, Figure 6 shows that the Stepped rounding function dominates in Instance 3 and has better diversity in Instance 4. The results indicate that the Stepped rounding function could provide a uniformly distributed non-dominated set. Overall, the performance of the Stepped rounding function is very close to the Reciprocal function and almost the same with Quadratic function.

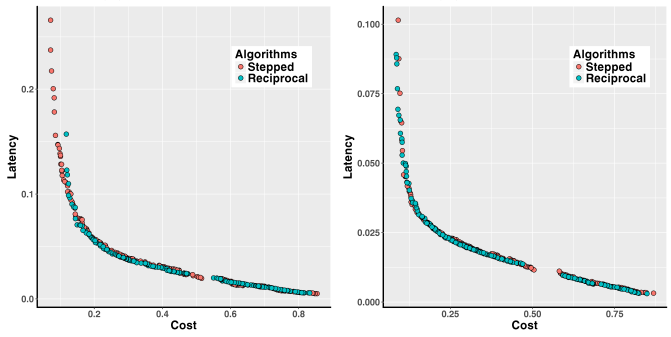
5.5.7 *Combination of Static Function*

In this experiment, we combined all solutions from 5 static rounding functions mentioned in Section 5.5.1 and applied a fast non-dominated sorting over it. The performance is compared with the Reciprocal rounding function in Figure 7. As the figure shows, the combined non-dominated set



(a) instance 2

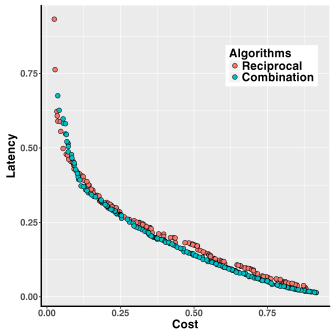
(b) instance 3



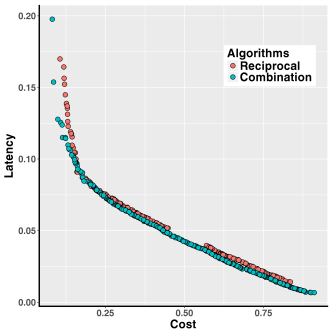
(c) instance 4

(d) instance 5

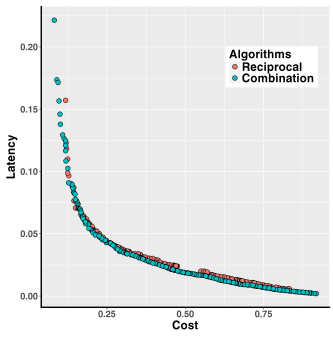
Fig. 6: Stepped Rounding Function Experiments: The non-dominated solutions among the sets obtained by 40 independent runs of Stepped function and Reciprocal function



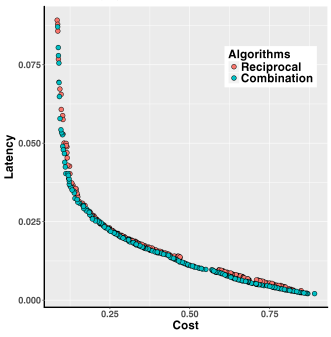
(a) instance 2



(b) instance 3



(c) instance 4



(d) instance 5

Fig. 7: 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

dominates all the other results in all the test instances. The solutions are not only diverse but also uniformly distributed. However, the method takes five times longer than using the Reciprocal function.

5.6 BMOPSOCD versus BNSPSO, NSGA-II, and BPSO

To evaluate the performance of our proposed BMOPSOCD with dynamic function, we conducted experiments to compare its performance with BNSPSO, BPSO from our previous research [11], and NSGA-II from [12]. Based on the analysis in Section 5.5, we use the Quadratic function as the rounding function. The results are shown in Table 5 and in Figure 8. The figures show that our propose BMOPSOCD dominate other algorithms in IGD (the small the better) and dominate other algorithms in HV (the bigger the better) except the instance 1. The margin between BMOPSOCD and other algorithms are huge. We can also observe a decline trend in the performance of other algorithms in HV. But the performance of BMOPSOCD maintains well in large test instances. The only exception is Instance 1, BPSO has the best HV value. On all instances, the p value between BMOPSOCD is less than 10^{-10} which means BMOPSOCD is significantly better. In addition, only BMOPSOCD shows an excellent performance in terms of HV, when the number of variables increases. On all instances, BMOPSOCD achieved a considerably better performance than other three algorithms in IGD, due to the better coverage.

In comparison with BNSPSO and NSGA-II, BMOPSOCD with dynamic rounding function achieved significantly better convergence and diversity. One reason is that with the dynamic rounding function, BMOPSOCD could move out of local optima. Other reason is the BMOPSOCD keeps an external archive. Although the three algorithms maintain the same population, they produce different sizes of solutions. BMOPSOCD outputs an archive with a size of 250 while other two algorithms output a population of size 50.

In Instance 1, the convergence of BMOPSOCD is worse than that of BPSO. One reason is related to the problem size. BPSO has better performance in small problems. When the problem size increases, its performance drops rapidly. In contrast, BMOPSOCD with the dynamic rounding function is not much affected by the number of variables.

Regarding execution time (Table 6), BMOPSOCD has similar execution time with other approaches. It achieves the best or the second best performance for 11 out of 14 instances. The reason is the velocity and position calculation (Equation 3 and 4) are linear so that these calculations can be vectorized. Conversely, for binary PSO, new positions are calculated by a sigmoid function which is non-linear. Therefore, an implementation of binary PSO must apply loop in the calculation.

In summary, from the experimental evaluation comparing the proposed algorithm with previous approaches, we observe that on most test instances, BMOPSOCD can achieve much better results than BNSPSO, NSGA-II, and BPSO in both convergence and diversity. The performance of MOPSOCD with dynamic rounding function is not much affected by the number of variables. This is a significant advantage of BMOPSOCD over the other three approaches.

6 Related Work

Location-allocation problems with immobile servers such as hospitals, stores and disposal centers have arisen for a long time [40]. The main characteristic of this problem is that

TABLE 5: Comparison between BMOPSOCD, BNSPSO, NSGA-II and BPSO: The non-dominated solutions among the sets obtained by 40 independent runs of different algorithms

instances	Method	HV (avg ± sd)	IGD (avg ± sd)
instance 1	BMOPSOCD	0.83 ± 0.04	**3.73E-02 ± 1.03E-02**
	BNSPSO	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
instance 2	BMOPSOCD	**0.73 ± 0.011**	**3.15E-02 ± 7.92E-03**
	BNSPSO	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
instance 3	BMOPSOCD	**0.81 ± 0.012**	**7.03E-03 ± 1.92E-03**
	BNSPSO	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
instance 4	BMOPSOCD	**0.83 ± 0.016**	**5.80E-03 ± 1.37E-03**
	BNSPSO	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
instance 5	BMOPSOCD	**0.84 ± 0.014**	**3.74E-03 ± 1.02E-03**
	BNSPSO	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
instance 6	BMOPSOCD	**0.81 ± 0.014**	**5.81E-03 ± 1.95E-03**
	BNSPSO	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
instance 7	BMOPSOCD	**0.79 ± 0.015**	**7.13E-03 ± 1.70E-03**
	BNSPSO	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
instance 8	BMOPSOCD	**0.81 ± 0.015**	**8.77E-03 ± 1.90E-03**
	BNSPSO	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
instance 9	BMOPSOCD	**0.83 ± 0.016**	**3.58E-03 ± 1.53E-03**
	BNSPSO	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
instance 10	BMOPSOCD	**0.80 ± 0.012**	**4.30E-03 ± 1.75E-03**
	BNSPSO	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
instance 11	BMOPSOCD	**0.79 ± 0.012**	**5.19E-03 ± 1.81E-03**
	BNSPSO	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
instance 12	BMOPSOCD	**0.80 ± 0.01**	**3.12E-03 ± 6.71E-04**
	BNSPSO	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
instance 13	BMOPSOCD	**0.83 ± 0.012**	**2.63E-03 ± 6.73E-04**
	BNSPSO	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
instance 14	BMOPSOCD	**0.84 ± 0.015**	**3.66E-03 ± 1.75E-03**
	BNSPSO	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

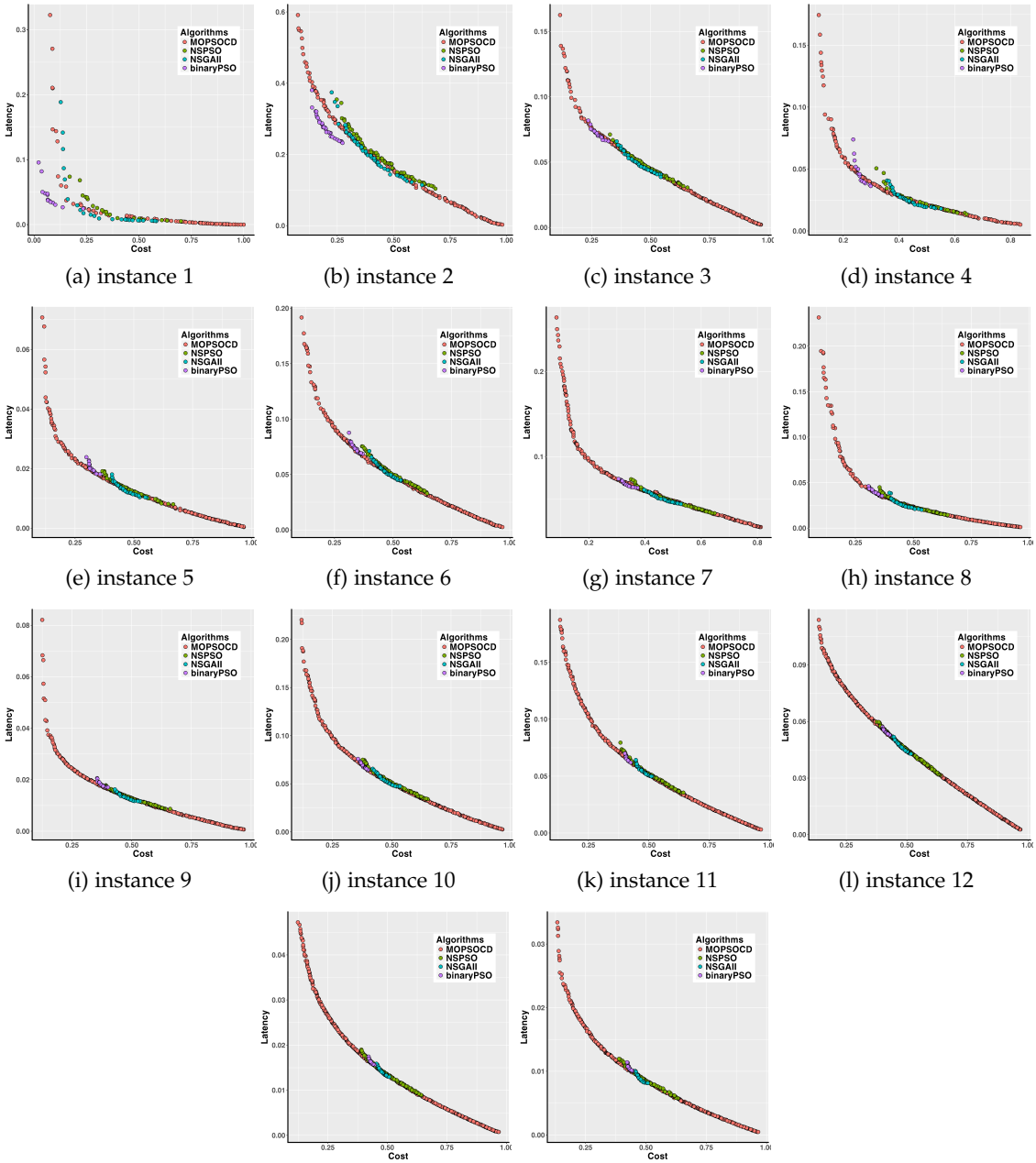
TABLE 6: Execution time (seconds)

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

customers need to travel to servers for services in contrast with mobile servers traveling to customers in response to requests. Similar to WSLAP, response time represents customer traveling time and network latency has a strong impact on response time between customers' locations and web services' locations.

Most previous researchers treated WSLAP as a single objective problem and proposed many simple greedy-based heuristic algorithms, such as ADD, DROP and Alternate Location Allocation (ALA) [9]. As an improvement, Sun et al. [9] proposed a DAL method which combines DROP, ALA and Linear Programming relaxations of Integer Transportation problems. Aboolian et al [8] introduced a location-allocation model for a web service provider in duopoly competitive market. They solved the problem with Integer Linear Programming (ILP) technique. Goiri et al. [41] proposed a novel combination of linear programming and simulated annealing approach for allocation of data centers for Internet services. Yuk and Lin [42] considered not only facility location and demand allocation, but also resource capacity allocation (number of service replicas). They formulated the problem as a stochastic mixed integer program and proposed a simulation-based hybrid heuristic to solve the dynamic problem under different response time service level. In contrast, in our previous studies [11], [12], we proposed a multi-objective algorithm with linear aggrega-

Fig. 8: MOPSOCD, BNSPSO, NSGA-II, and BPSO Experiments: The non-dominated solutions among the sets obtained by 40 independent runs of different algorithms



(a) instance 1 (b) instance 2 (c) instance 3 (d) instance 4

(e) instance 5 (f) instance 6 (g) instance 7 (h) instance 8

(i) instance 9 (j) instance 10 (k) instance 11 (l) instance 12

(m) instance 13 (n) instance 14

tion using PSO and a multi-objective algorithm with Pareto front using NSGA-II. Both results show that multi-objective model suits the problem well. The solutions contain a set of equally good allocations trade off between cost and service response time. Hence, the service providers can select the most suitable allocation according to their preferences.

For example, a multi-objective algorithm gives a solution set contains many solutions, among them, solution A is {cost: 1000, response time: 3.5s}, solution B is {cost: 500, response time: 15s}, and solution C is {cost:800, response time: 10s}. And a single-objective algorithm gives a solution D: {cost: 900, 12s}. In this case, solution C better than solution D in both objectives (Cost: 800 $<$ 900, Response: 10s $<$ 12s).

Meanwhile, If a service provider would like to provide the best service quality to customers without caring too much of the budget, then solution A is the most suitable choice. Conversely, a service provider with tight budget may choose solution C. Service providers can balance their profit with our multi-objective model.

We can see from previous example, a service provider can select a suitable solution if the solution set is diverse enough - a provider may have precise requirement (e.g. cost $<$ 525, response $<$ 14.5s). The new approach can provide a much more diverse solution set compared with our previous approaches including BPSO, BNSPSO, and NSGA-II (see Figure. 8). This is a major contribution of this work.

In addition, the major limitation of Integer Linear Programming is that the computational time increases rapidly when the size of the problem increases. Torrent-Fontbona et al. [43] proposed a method that first uses clustering technique to reduce the complexity and then applies heuristic methods to solve the problem. It reduces the computation time as well as the quality of the solutions. However, our proposed algorithms can deal with large test instance with hundreds of services and more. Our BMOPSOCD shows none-degradation with increasing number of variables.

With the emergence of cloud computing, virtual machine allocation problem is becoming important for service providers and cloud providers. Instead of allocating web services to physical servers, web services are first allocated to virtual machines(VMs) and then VMs are allocated to physical machines. The problem of VM allocation is similar to WSLAP with multiple objectives and binary VM allocation. Kessaci et al. [44] proposed an MOGA-CB for minimizing the cost of VMs and response time when considering web service composition as a workflow. Phan et al. [45] proposed a framework called Green Monster, to dynamically move web services across Internet data centers for reducing their carbon footprint while maintaining their performance. Greenmonster applies a modified version of NSGA-II algorithm [19] with an additional local search process. Our BMOPSOCD can solve VM allocation problem with minor modification of objective and constraint functions. Therefore, BMOPSOCD is useful for Cloud providers.

7 Conclusion and Future Work

This paper proposed a BMOPSOCD to solve the WSLAP with the aim of producing a set of high-quality solutions that covers most of the Pareto front when dealing with large test instances. For that, we proposed a binary version of multi-objective PSO with crowding distance to solve the WSLAP. We employed a rounding function mechanism which not only makes a continuous algorithm compatible with binary problems but also significantly improves the quality of solutions. Three types of rounding functions were developed. From the experiments, we observed that the solutions obtained by BMOPSOCD with dynamic rounding functions have a great diversity that almost covers the whole Pareto front. Meanwhile, BMOPSOCD could produce good solutions regardless of the increasing problem size. For service providers who use this algorithm to design the allocation of their services, there are several advantages. First of all, BMOPSOCD can provide near optimum solutions in comparison with solutions from greedy-based heuristic algorithms. Secondly, it provides a wider range of non-dominated solutions compares with single-objective algorithms. In addition, near-optimal solutions are given within a feasible amount of time.

There are a few directions that we can work on in the future. Firstly, our model can be further improved by considering service composition. For now, the problem model considers each service as an atomic service. With the increasing usages of composite services composed with atomic services distributed over the Internet, we need to consider service composition workflow while solving WSLAP. Service composition workflows have a significant impact on the allocation of atomic services 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. To avoid single point failure, web service providers often deploy many copies of the service in different candidate locations to keep the availability.

In addition, service deployment in Cloud becomes a hot topic between service computing and cloud computing. Cloud computing provides an elastic resource management for services so that it removes the burden of managing hardwares for service providers. However, new issues such as dynamic virtual machine placement for service composition, elastic resource management for services have emerged. Difficult (often NP-hard) and dynamic nature make it extremely challenging.

References

[1] A. Dan, R. D. Johnson, and T. Carrato, "SOA service reuse by design," in *SDSOA 2008 - 2nd international workshop on Systems development in SOA environments (@ICSE)*, ser. SDSOA '08, no. August. ACM, 2008, p. 25.

[2] S. Ran, "A model for web services discovery with QoS," *ACM SIGecom Exchanges*, vol. 4, no. 1, pp. 1–10, 2003.

[3] D. A. Menascé, "QoS issues in web services," *IEEE Internet Computing*, vol. 6, no. 6, pp. 72–75, nov 2002.

[4] E. Cronin, S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," in *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, 2002, pp. 1369–1382.

[5] K. Deb and K. Deb, "Multi-objective Optimization," in *Search Methodologies*. Springer London, 2014, pp. 403–449.

[6] J. Liu, M. Tang, Z. Zheng, X. F. Liu, and S. Lyu, "Location-Aware and Personalized Collaborative Filtering for Web Service Recommendation," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 686–699, sep 2016.

[7] N. Mladenović, J. Brimberg, P. Hansen, and J. A. Moreno-Pérez, "The p-median problem: A survey of metaheuristic approaches," *European Journal of Operational Research*, vol. 179, no. 3, pp. 927–939, 2007.

[8] R. Aboolian, Y. Sun, and G. J. Koehler, "A location-allocation problem for a web services provider in a competitive market," *European Journal of Operational Research*, vol. 194, no. 1, pp. 64–77, 2009.

[9] Y. Sun and G. J. Koehler, "A location model for a web service intermediary," *Decision Support Systems*, vol. 42, no. 1, pp. 221–236, 2006.

[10] E. Klotz, A. M. Newman, and M. Tyson, "Practical Guidelines for Solving Difficult Mixed Integer Linear Programs," *Surveys in Operations Research and Management Science*, vol. 18, no. 1, pp. 18–32, 2012.

[11] B. Tan, Y. Mei, H. Ma, and M. Zhang, *Particle swarm optimization for multi-objective web service location allocation*, 2016, vol. 9595.

[12] B. Tan, H. Ma, and M. Zhang, *Optimization of Location Allocation of Web Services Using a Modified Non-dominated Sorting Genetic Algorithm*, 2016, vol. 9592.

[13] S. Desai, "Multi-Objective Constrained Optimization using Discrete Mechanics and NSGA-II Approach," *International Journal of Computer Applications*, vol. 57, no. 20, pp. 14–20, 2012.

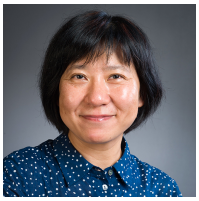
[14] W. Ongsakul, J. G. Singh, and S. R. Tuladhar, "Multi-objective approach for distribution network reconfiguration with optimal DG power factor using NSPSO," *IET Generation, Transmission & Distribution*, vol. 10, no. 12, pp. 2842–2851, 2016.

[15] M. R. Senouci, D. Bouguettouche, F. Souilah, and A. Mellouk, "Static wireless sensor networks deployment using an improved binary PSO," *International Journal of Communication Systems*, vol. 29, no. 5, pp. 1026–1041, 2016.

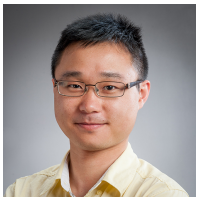
[16] C. R. Raquel and P. C. Naval, "An effective use of crowding distance in multiobjective particle swarm optimization," in *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, ser. GECCO '05. ACM, 2005, p. 257.

[17] *A Critical Survey of Performance Indices for Multi-Objective Optimisation*, 2003.
[18] N. Riquelme, C. Von Lucken, and B. Baran, "Performance metrics in multi-objective optimization," in *2015 Latin American Computing Conference (CLEI)*, oct 2015, pp. 1–11. [Online]. Available: http://ieeexplore.ieee.org/document/7360024/
[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, apr 2002.
[20] M. Kim, T. Hiroyasu, M. Miki, and S. Watanabe, "SPEA2+: Improving the Performance of the Strength Pareto Evolutionary Algorithm 2," in *Sort*, vol. 3242/2004. Springer, 2004, pp. 742–751.
[21] Q. Zhang and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.
[22] M. Clerc, "Particle Swarm Optimization," in *Encyclopedia of machine learning*. Springer, 2006, pp. 760–766.
[23] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 5. IEEE, 1997, pp. 4–8.
[24] C. A. Coello Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, 2004.
[25] X. Li, "A Non-dominated Sorting Particle Swarm Optimizer for Multiobjective Optimization," in *Genetic and Evolutionary Computation GECCO 2003*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, vol. 2723, pp. 37–48.
[26] C. Coello Coello Coello and G. Toscano Pulido, "A Micro-Genetic Algorithm for Multiobjective Optimization," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, vol. 1993, pp. 126–140.
[27] D. Borgetto, H. Casanova, G. Da Costa, and J. M. Pierson, "Energy-aware service allocation," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 769–779, 2012.
[28] M. A. S. Mojtaba Ahmadieh Khanesar, Mohammad Teshnehlab, M. A. Khanesar, M. Teshnehlab, and M. A. Shoorehdeli, "A novel binary particle swarm optimization," in *Particle Swarm Optimization*, vol. 1, no. 1. IEEE, 2007, pp. 1–6.
[29] E. Laskari, K. Parsopoulos, and M. Vrahatis, "Particle swarm optimization for integer programming," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, vol. 2, 2002, pp. 1582–1587.
[30] B. Xue, M. Zhang, S. Member, and W. N. Browne, "Particle swarm optimization for feature selection in classification: a multi-objective approach," *IEEE transactions on cybernetics*, vol. 43, no. 6, pp. 1656–1671, 2013.
[31] R. Haupt, "Antenna design with a mixed integer genetic algorithm," *Antennas and Propagation, IEEE Transactions on*, vol. 55, no. 3, pp. 577–582, 2007.
[32] E. S. Olivas, *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques*. IGI Global, 2009.
[33] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Comput. Methods Appl. Mech. Engrg*, vol. 191, no. 11, pp. 1245–1287, 2002.
[34] Y. Zhang, Z. Zheng, and M. R. Lyu, "Exploring latent features for memory-based QoS prediction in cloud computing," in *Proceedings of the IEEE Symposium on Reliable Distributed Systems*, 2011, pp. 1–10.
[35] E. Zitzler and L. Thiele, *Multiobjective optimization using evolutionary algorithms A comparative case study*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 292–301.
[36] M. A. Villalobos-Arias, G. T. Pulido, and C. A. Coello Coello, "A proposal to use stripes to maintain diversity in a multi-objective particle swarm optimizer," in *Proceedings - 2005 IEEE Swarm Intelligence Symposium, SIS 2005*, vol. 2005, 2005, pp. 23–30.
[37] D. a. Van Veldhuizen, "Multiobjective Evolutionary Algorithms: Classifications, Analyses and New Innovations," Evolutionary Computation, Tech. Rep., 1999.
[38] J. Xin, G. Chen, and Y. Hai, "A particle swarm optimizer with multi-stage linearly-decreasing inertia weight," in *Proceedings of the 2009 International Joint Conference on Computational Sciences and Optimization, CSO 2009*, vol. 1. IEEE, 2009, pp. 505–508.
[39] X. Qi and F. Palmieri, "Theoretical Analysis of Evolutionary Algorithms with an Infinite Population Size in Continuous Space. Part I: Basic Properties of Selection and Mutation," *Trans. Neur. Netw.*, vol. 5, no. 1, pp. 102–119, 1994.
[40] N. Vidyarthi and S. Jayaswal, "Efficient solution of a class of location allocation problems with stochastic demand and congestion," *Computers and Operation Research*, vol. 48, pp. 20–30, 2014.
[41] Í. Goiri, K. Le, J. Guitart, J. Torres, and R. Bianchini, "Intelligent placement of datacenters for internet services," in *Proceedings - International Conference on Distributed Computing Systems*, 2011, pp. 131–142.
[42] C. K. Y. C. Lin, "Solving a Location, Allocation, and Capacity Planning Problem with Dynamic Demand and Response Time Service Level," *Mathematical Problems in Engineering*, vol. 2014, pp. 1–25, 2014.
[43] F. Torrent-Fontbona, V. Mu??oz, and B. L??pez, "Solving large immobile location-Allocation by affinity propagation and simulated annealing. Application to select which sporting event to watch," *Expert Systems with Applications*, vol. 40, no. 11, pp. 4593–4599, 2013.
[44] Y. Kessaci, N. Melab, and E. G. Talbi, "A pareto-based genetic algorithm for optimized assignment of VM requests on a cloud brokering environment," in *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, 2013, pp. 2496–2503.
[45] D. H. Phan, J. Suzuki, R. Carroll, S. Balasubramaniam, W. Donnelly, and D. Botvich, "Evolutionary multiobjective optimization for green clouds," in *Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference Companion - GECCO Companion '12*, ser. GECCO '12. ACM, 2012, p. 19.

Boxiong Tan is a Ph.D student at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He received B.E degree from Guangzhou University and M.E. degree from Victoria University of Wellington in 2011 and 2016 respectively.



Hui Ma is a Senior Lecturer at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. Her research interests include web information systems, distributed databases, and Cloud Computing.



Yi Mei is a Lecturer at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research interests include evolutionary computation in scheduling, combinatorial optimization, and genetic programming.

Mengjie Zhang He is currently Professor of Computer Science, Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) in the Faculty of Engineering, Victoria University of Wellington. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization.