Evolutionary Multi-Objective Optimization for Web Service Location Allocation

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

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 paper, we model the Web service location allocation problem as a multi-objective optimization problem - minimizing overall network latency and total cost. We develop a new PSO-based algorithm with rounding function approach to provide a set of 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.

Index Terms—Web service location allocation, Quality of Service, Evolutionary Computation, Particle Swarm Optimization.

1 Introduction

In recent years, service-oriented computing (SOC) enables software applications to be developed in an agile and cost efficient way [**?**]. Web services are well defined, self-contained modules that provide standard business functionality and can be accessed via the Internet [**?**]. 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 [**?**], 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 [**?**]. Study [**?**] 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 [**?**], since network latency is related to network topology as well as physical distance [**?**]. 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 [**?**] 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. This gives rise of the Web service location allocation (WSLA) problem.

The WSLA is a very challenging problem. It is NP-hard due to its huge searching space. Therefore, it is impractical to find the optimal solution when the number of services and the locations are huge. The WSLA problem is essential a multi-objective optimization problem [**?**] for which there are two conflicting objectives, to minimize latency and total cost. Multi-objective Evolutionary Optimization Algorithm (MOEA) methodologies are ideal for solving multi-objective optimization problems [**?**], 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 [**?**].

Previous researches [**?**], [**?**] 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. Various multi-objective optimization algorithms have been proposed in the past. [**?**], [**?**] 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 considered to solve the problem.

In our previous work [**?**] on Web service location allocation, we study the performance of applying three algorithms, Binary PSO (BPSO), Non-Dominated Sorting PSO (NSPSO), and NSGA-II (a Genetic Algorithm (GA)-based multi-objective optimization algorithm) to the problem of WSLA. 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. Multi-objective particle swarm optimization with crowding distance (MOPSOCD) is developed in [**?**] to produce a well-

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

distributed set of non-dominated solutions. MOPSOCD 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. Based on MOPSOCD, in this paper we develop a new binary multi-objective PSO with crowding distance approach (BMOPSOCD) to solve the WSLA problem. We introduce a new mechanism of rounding to improve the original MOPSOCD. The rounding function method is a mechanism that makes a continuous algorithm compatible with discrete problems.

The overall goal is to develop a new binary multi-objective PSO-based approach to the WSLA problem by considering two potentially conflicting objectives - minimizing cost and minimizing network latency. To accomplish this goal, we will achieve the following objectives:

1) To design rounding strategies 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 using some experiments.

The paper is organized as follows. Section **??** reviews existing works and various PSO approaches and provides background knowledge of solving the problem. Section **??** describe the WSLA problem with formal models. Section **??** presents our approach of BMOPSOCD. Section **??** provides a conclusion and discusses future work.

2 Background

2.1 Related Work

Most of the researchers treat WSLA problem as a single objective problem. [**?**], [**?**] try to solve the problem by using integer linear programming techniques. In particular, [**?**] solves this problem by employing greedy and linear relaxation. Researches on network virtualization [**?**], [**?**] employs greedy algorithms to allocate virtual machines (VMs) in the data center so that the requirements of network bandwidth are met. [**?**] 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.

Huang [**?**] proposes an enhanced genetic algorithm (GA)-based approach on WSLA. 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 [**?**] proposes MOGA-CB for minimizing cost of VMs instance and response time. [**?**] 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 [**?**] with an additional local search process.

As shown from above previous researchers have studied the WSLA problem with single-objective algorithm, linear programming technique and greedy algorithm. These approaches have many obvious disadvantages. WSLA problem in nature is a multi-objective problem which should be addressed by multi-objective algorithms. In our previous work [**?**] we develop two PSO-based approaches, one with weighted-sum fitness function (named WSPSO), and the other using the fast Non-dominate sorting scheme (named NSPSO), for solving the WSLA problem. We study the performance of WSPSO, NSPSO and that of NSGA-II, one of the most commonly used multi-objective genetic algorithms with experimental evaluations. Our evaluation results show that both WSPSO and NSPSO outperform NSGA-II while NSPSO achieved a more diverse set of solutions that WSPSO. However, the performance of all the three approaches decrease while working on large datasets.

2.2 Particle Swarm Optimization (PSO)

PSO was proposed by Kennedy and Eberhart in 1995 [**?**]. It 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:

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

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

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.

PSO was originally developed to address continuous optimization problems with a single objective. 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) [**?**]. In BPSO, the position of each particle is a vector of binary numbers, which are restricted to 1 or 0.

Several multi-objective optimization algorithms are based on PSO such as Multi-objective PSO (MOPSO) [**?**], and Non Dominated Sorting PSO (NSPSO) [**?**]. [**?**] studies the performance of four multi-objective algorithms, NSGA-II [**?**], PAES [**?**], Micro-GA [**?**] and MOPSO, and shows that

MOPSO is most capable to generate the best set of non-dominated solutions close to the true Pareto front but with low computational cost. To improve the diversity of non-dominated solutions, Raquel et al. [?] 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. Due to its competitive of generating a well-distributed set of non-dominated solutions, in this paper we apply MOPSOCD to solve the WSLA problem.

3 PRELIMINARY

In this work we considers the WSLA as a multi-objective problem with two potentially conflicting objectives, minimizing cost and network latency. In this section, we first describe the WSLA problem in detail. Then we introduce matrices for modelling the input and output information of the problem.

To solve the WSLA 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 can be a centre location of a user-centered area. Candidate locations are the geographic location that are suitable to deploy Web services, e.g., the locations of servers hosting Web services. A service providers need to deploy a set of Web services $\mathcal{W} = \{W_1, \ldots, W_s\}$, each of which to be deployed to at lease one location. Note that a Web service can be deployed to multiple locations for the benefit of reducing 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 location A_j. For each user centre $U_k \in \mathcal{U}$ and each candidate location $A_j \in \mathcal{A}$, there is a latency L_{jk}, which affects the response time from the location A_j to the user center U_k. It normally depends on the distance between two geographical locations. Each Web service is invoked from a user center $U_k \in \mathcal{U}$ with an invocation frequency F_{ik}. Given the information above, WSLA is to design an allocation plan that allocate a set of services $\mathcal{W} = \{W_1, W_2, \ldots, W_s\}$ to set of candidate locations $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ so that the total deployment cost f_1 and network latency f_2 is minimized. Total deployment cost f_1 and total network latency f_2 can be calculated as follows:

$$f_1 = \sum_{1=1}^{s} \sum_{j=1}^{n} C_{ij} x_{ij}, \tag{3}$$

$$f_2 = \sum_{i=1}^{s} \sum_{k=1}^{m} F_{jk} r_{ik} \tag{4}$$

where x_{ij} takes 1 if service W_i is allocated to location A_j, and 0 otherwise. r_{ik} stands for the response time of service W_i to the center U_k, which is calculated as

$$r_{ik} = \min\{L_{jk} \mid j \in \{1, 2, ..., k\} \text{ and } x_{ij} = 1\} \tag{5}$$

WSLA has the following two objective functions and one constraint.

$$\text{minimize} \quad f_1 = \sum_{1=1}^{s} \sum_{j=1}^{n} C_{ij} x_{ij}, \tag{6}$$

$$\text{minimize} \quad f_2 = \sum_{i=1}^{s} \sum_{k=1}^{m} F_{ik} r_{ik}, \tag{7}$$

$$\begin{aligned} \text{subject to} \quad & \sum_{j=1}^{n} x_{ij} \geqslant 1, \forall i \in 1, \cdots, s \\ & x_{ij} \in 0, 1, \forall i \in 1, \cdots, s, \forall j \in 1, \cdots, n \end{aligned} \tag{8}$$

In this paper, we will use the following matrices to model the above mentioned information.

Matrices	
L	server network latency matrix $L = \{L_{jk}\}$
A	service location matrix $X = \{x_{ij}\}$
F	service invocation frequency matrix $F = \{F_{ik}\}$
C	cost matrix $C = \{C_{ij}\}$
R	user response time matrix $R = \{r_{ik}\}$

A *service invocation frequency matrix*, $F = [F_{ik}]$, is used to record services invocation frequencies from user centers to services. A *cost matrix*, $C = [C_{ij}]$, is used to record the fixed deployment fees at candidate locations, where C_{ij} is an integer that indicates the fixed deployment fee at a candidate location. A *service location matrix* $X = [x_{ij}]$ represents location allocation plan, with x_{ij} representing wether a service W_i is deployed at a candidate location A_j or not. That is,

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

For each given service location allocation matrix, A *response time matrix* $R = [r_{ik}]$ can be computed to get the shortest response time of accessing service W_i from user center U_k. The aim of WSLA is find a location allocation matrix $X = [x_{ij}]$ such that it results minimal overall network latency and overall deployment cost.

For example assume we are given the following matrices, F, L, C we can calculate total cost and latency for a given allocation plan represented by the allocation matrix X below.

$$F = \begin{pmatrix} 120 & 35 & 56 \\ 14 & 67 & 24 \\ 85 & 25 & 74 \end{pmatrix} L = \begin{pmatrix} 0 & 5.776 & 6.984 \\ 5.776 & 0 & 2.035 \\ 0.984 & 1.135 & 2.3 \end{pmatrix}$$

$$C = \begin{pmatrix} 130 & 80 & 60 \\ 70 & 50 & 30 \\ 40 & 78 & 54 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

We can use the two example matrices L and X presented above to construct the response time matrix R. For each service W_i, by checking matrix X, 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 U_k. If there is more than one location, then the smallest latency is selected.

f_1 calculates the overall cost of deployed services, where C_{ij} is the cost of deployed service W_i at candidate location

A_j, x_{ij} represents whether service W_i is allocated to candidate location A_j. The sum of the multiplication of C_{ij} and x_{ij} is the total deployment cost.

We can calculate total cost using the above matrices C and A.

$$\begin{aligned} f_1 &= C_{11} * x_{11} + C_{12} * x_{12} + C_{13} * x_{13} + ... + C_{33} * x_{33} \\ &= 130 * 0 + 80 * 1 + 60 * 0 + ... + 54 * 0 \\ &= 228 \end{aligned}$$

We can compute the response time matrix $R = [r_{ik}]$ as the following, with r_{ik} denotes the shortest response time from service W_i to a user center U_k and F_{ik} is the invocation frequency of service W_i from user center U_k

$$R = \begin{pmatrix} 5.776 & 0 & 2.035 \\ 0.984 & 1.135 & 2.3 \\ 0 & 0 & 2.035 \end{pmatrix}$$

Finally, we can compute the overall network latency f_2 using matrix F and R.

$$\begin{aligned} f_2 &= F_{11} * r_{11} + F_{12} * r_{12} + F_{13} * r_{13} + ... + F_{33} * r_{33} \\ &= 120 * 5.776 + 35 * 0 + 56 * 2.035 + ... + 74 * 2.035 \\ &= 1102.691 \end{aligned}$$

The constraint requires that each web service is deployed in at least one location. The example matrix X above satisfies the constraint.

4 BMOPSOCD for Web Service Location Allocation

In this section we present our approach of BMOPPSOCD to solve the WSLA problem. We first define the representation of the problem followed by fitness functions to be used and our method of handling the constraint. We will then present the BMOPPSOCD algorithm for WSLA followed by different rounding methods that are considered in this study.

4.1 Particle Representation

As we see from above that WSLA is to design a location allocation matrix $X = [x_{ij}]$, where $i = 1, ..., s$ and $j = 1, ..., n$. The major difference between BMOPSOCD and three previous approaches, WSPSO, NSPSO and BMOPSOCD, is the particle representation. As mentioned in Section **??**, the solution of WSLA is a $x \times n$ matrix. As we known that PSO can be used to generate vector-based solutions, we need to transform the $x \times n$ matrix into a $(x \times n)$ vector y. The element x_{ij} in X corresponds to the $(n \cdot (i-1)+j)^{th}$ element y_u 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

$$X = \begin{pmatrix} 0.12 & 0.87 & 0.42 \\ 0.07 & 0.32 & 0.95 \\ 0.76 & 0.64 & 0.27 \end{pmatrix}$$

can transformed into a vector:
$Y = [0.12, 0.87, 0.42, 0.07, 0.32, 0.95, 0.76, 0.64, 0.27]$.

As we know that the final output of WSLA is an allocation matrix with x_{ij} as a binary value, the particle with the continuous representation needs to be transformed into the binary representation, using a rounding methods. The way of rounding is significant to the quality of the final results. In the following sections we will discuss different rounding methods. Note that, Vector Y is used in the update phase of our PSO-based algorithm. During the fitness evaluation phase, Y is first decoded into matrix X with a selected rounding algorithm.

4.2 Fitness Function and Constraint Handling

After particles represented as vectors being transformed to allocation matrices, two fitness functions Equation **??** and **??** are used to evaluate fitness of particles that represents location allocation plans. Based on the fitness of solutions a set of non-dominant solutions are returned.

As we see in Section **??**, WSLA needs to satisfy the constraint defined as Equation. **??**, which means each service needs to be allocated to at least one location. However, during the searching process of PSO, the constraint can be violated. That is, infeasible particles may be generated 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, 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 the death penalty method.

4.3 The BMOPSOCD algorithm for Web Service Location Allocation

Algorithm **??** presents our BMOPSOCD algorithm for solving the WSLA problem. As seen in the algorithm, 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.

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

$$v_{id} = w * v_{id} + c_1 * r_{1i} * (p_{id} - x_{id}) + c_2 * r_{2i} * (p_{pg} - x_{id}) \quad (9)$$

Algorithm 1 BMOPSOCD for WSLA

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 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
13: Update its position x_i
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 $pbest$
18: **end for**
19: Insert new non-dominated solution into $Archive$, remove dominated solutions from $Archive$
20: **until** maximum iterations is reached
21: return $Archive$

w is the inertia weight, c_1 and c_2 are the acceleration factors, r_{1i} and r_{2i} are the randomm variables sampled from uniform distribution between 0 and 1, v_{id}, x_{id}, p_{id} and g_d denote the value in dimension d of v_i, x_i, p_i and g, respectively. For WSLA, the decision variable is binary, 0 or 1. Therefore, in our algorithm (Line 3 and 16) we apply a rounding function to transform continuous values to binary values. In the next section we discuss different rounding methods and studies their performance.

4.4 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 process of fitness evaluation. That is, in the initial stage, particles are initialized in real values. The updates of velocity and position are performed as usual. To evaluation the fitness of particles, particles in continuous representation need to transformed to the particles in binary representation, which can then be evaluate 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 [**?**] to solve integer programming problem. [**?**] uses a real value representation of chromosome for GA. Then, a eal value chromosome is rounded to an integer and binary representations in order to achieve a mixed integer optimization of array antenna pattern and micro-strip antenna. [**?**] adopts rounding and interval mapping strategy to solve 0-1 discrete, integer optimization and mixed optimization problem. [**?**] uses a random-round function which randomly returns round-up value or round-down value.

4.4.1 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.

$$x_{ij} = \begin{cases} 1 & \text{if } x'_{ij} > t \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

4.4.2 Dynamic Rounding Functions

The threshold plays an important role in searching for solutions for a given problem. 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. In the second step, same as static rounding function, it either rounds up or rounds down the value of x_{ij} according to t. Three dynamic rounding functions are considered. Equation **??** is a *linear function*. Equation **??** is a *quadratic function*. Equation **??** is a *reciprocal function*.

$$t = \frac{l-u}{\text{max\_gen}} g + u \tag{11}$$

$$t = \frac{l-u}{(\text{max\_gen})^2} g^2 + u \tag{12}$$

$$t = u - \frac{u-l}{\text{max\_gen} - g} (g \neq \text{max\_gen}) \tag{13}$$

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 **??**.

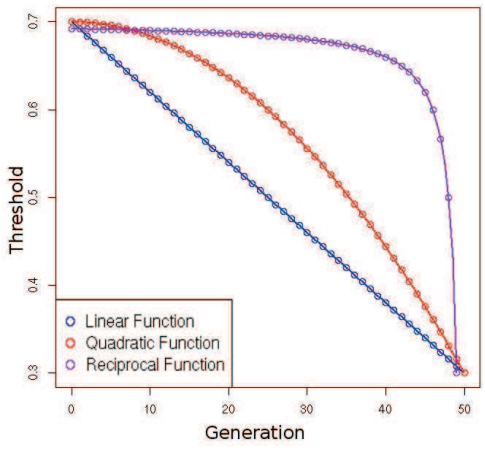


Fig. 1: Curve of three dynamic thresholds

4.5 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 [**?**]. In this section, we propose an adaptive threshold that embodied in the transfer learning process.

Figure **??** 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 runs 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 Equation **??** and remain steadily until next interval. This process is repeated until the lower boundary l is 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,s 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 **??** where t' denotes the current threshold value.

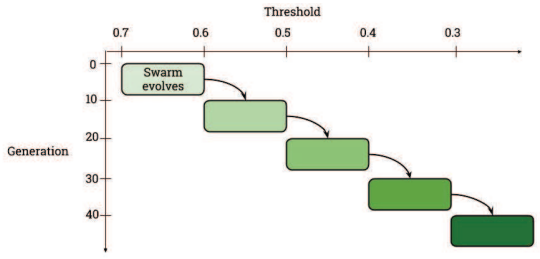


Fig. 2: Evolutionary process with an adaptive threshold

$$t = \begin{cases} = t' - \frac{u-l}{(\frac{\max\_gen}{i} - 1)} & \text{if (cur\_gen} \mod i) = 0 \\ = t' & \text{otherwise} \end{cases} \quad (14)$$

5 Experiment Design

The aim of this study is to propose a multi-objective WSLA approach which can produce a well-distributed solutions with good scalability. In the above section we present our proposed BMOPSOCD approach to the problem of WSLA. For this approach, static and dynamic rounding methods with different threshold settings are considered. In this section, a set of experiments have been conducted over three major features of the proposed 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 is the dynamic rounding functions used in the algorithm. 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 three other algorithms: PSO, NSPSO and NSGA-II (please see [**?**] for details).

5.1 Datasets

This project is based on both real world datasets [**?**], [**?**] and stimulated datasets [**?**]. The dataset includes a network latency matrix between 339 user centers and 5825 candidate locations. 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. As mentioned in Section **??**, two other matrices, deployment cost matrix C and invocation frequency matrix F, are needed as input information. In principle, development cost can be either fixed fees (monthly rent) or variable fees (e.g. depending on storage and other resource usage). For the sake of simplicity we consider fixed deployment fee. For each service, the deployment cost was randomly generated from a normal distribution with the mean of 100 and standard deviation of 20. For each user centre and each service, the invocation frequency was randomly generated from a uniform distribution between 1 and 120. To test scalability of our proposed approach we design a set of problems with different complexities.

Table **??** shows fourteen problems, listed with increasing size and difficulty, which are used as representative samples of the WSLA problem.

TABLE 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

5.2 Performance Metrics

We use hypervolume and IGD as the evaluation metrics. The IGD [**?**] is a modified version of generational distance [**?**], [**?**] 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 applying a non-dominated sorting over it. The approximated true Pareto front dominates all the other solutions.

5.3 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 Table **??**.

5.3.1 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 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 a fast non-dominated sorting.

5.3.2 Dynamic Rounding Function

In Section **??**, three dynamic rounding functions are proposed. In this section we evaluate the performances of different rounding functions to find out 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 **??** 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.

5.3.3 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.

5.3.4 Dynamic Rounding Functions

Table **??** 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 (Figure **??**). Figure **??** shows the reciprocal function slightly dominate the other two dynamic functions. However, the problem of the 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 **??**).

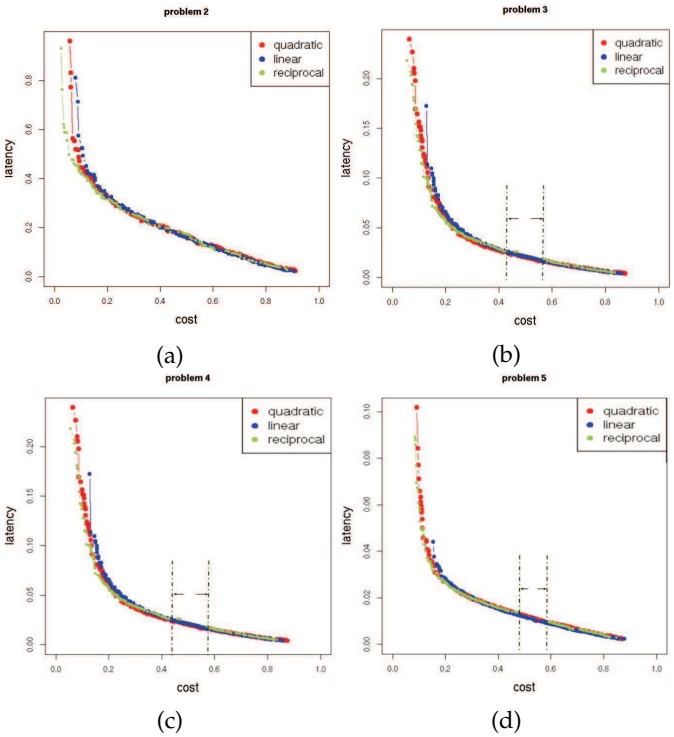


Fig. 3: Dynamic Rounding Function Experiments : The non-dominated solutions among the sets obtained by 40 independent runs of BMOPSOCD with different dynamic functions

TABLE 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**

According to Table **??**, the experimental results show that in terms of convergence, the reciprocal function produces the best result. However, it also shows the major disadvantage of the reciprocal function, which can not produce an entire Pareto front. In contrast, the quadratic function performs a little worse in convergence but obtains an uniformly distributed 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 minor change in threshold in the most generations, the swarm has longer time to search along the same direction. On the other hand, with the linear and quadratic function, the constant changing in direction could leads to premature convergence.

In comparison between the quadratic function and the linear function, the only difference is that changing in the quadratic function is smoother than linear. The searching

TABLE 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** $\pm$ **0.013**	0.72 $\pm$ 0.011
problem 3	0.815 $\pm$ 0.013	**0.83** $\pm$ **0.014**
problem 4	**0.87** $\pm$ **0.014**	0.85 $\pm$ 0.014
problem 5	**0.86** $\pm$ **0.020**	0.85 $\pm$ 0.022

process is in a continuous space, therefore, sudden changes are considered to be harmful.

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 the reciprocal function might be considered as trivial, but the gap in the non-dominated set could not be neglected. Therefore, the quadratic function is a better choice. On the other hand, from the perspective of WSLA, 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 WSLA.

5.3.5 An Adaptive Threshold Approach

We compared the performance of the adaptive threshold approach with reciprocal rounding function. Table **??** clearly shows reciprocal function dominates the adaptive threshold approach in the most cases. However, Figure **??** 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.

5.3.6 A Combination of Static Function

In this experiment, we combined all solutions from 5 static rounding functions mentioned in Section **??** and applied a fast non-dominated sorting over it. The performance is compared with reciprocal rounding function in Figure **??**. 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 BMOPSOCD versus NSPSO, NSGA-II and BPSO

To evaluate the performance of our proposed BMOPSOCD we conduct experiments to compare its performance with three previous approaches, NSPSO, BPSO and NSGA-II. The results are shown in Table **??**. 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 **??**, 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 while the performance of the other three approaches is obviously decreasing with the number of variable increasing. On all

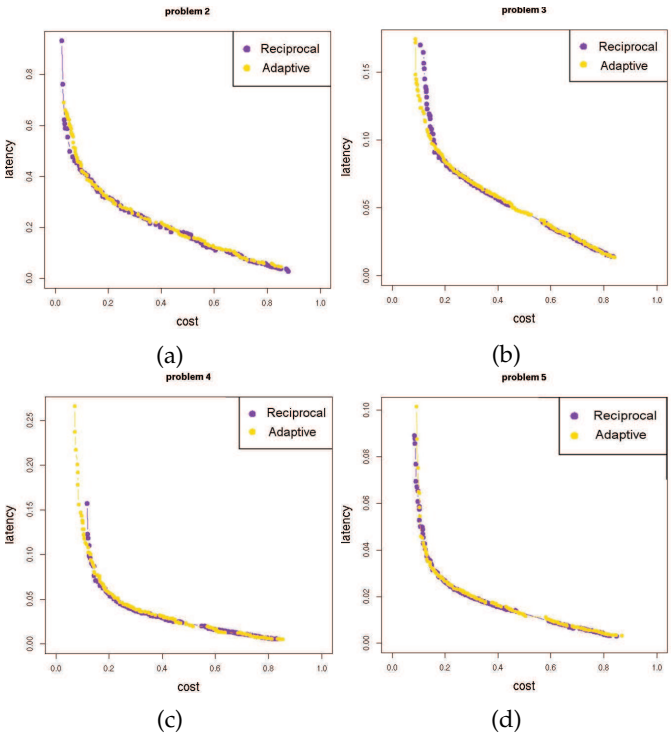


Fig. 4: Adaptive Rounding Function Experiments: The non-dominated solutions among the sets obtained by 40 independent runs of adaptive function and reciprocal function

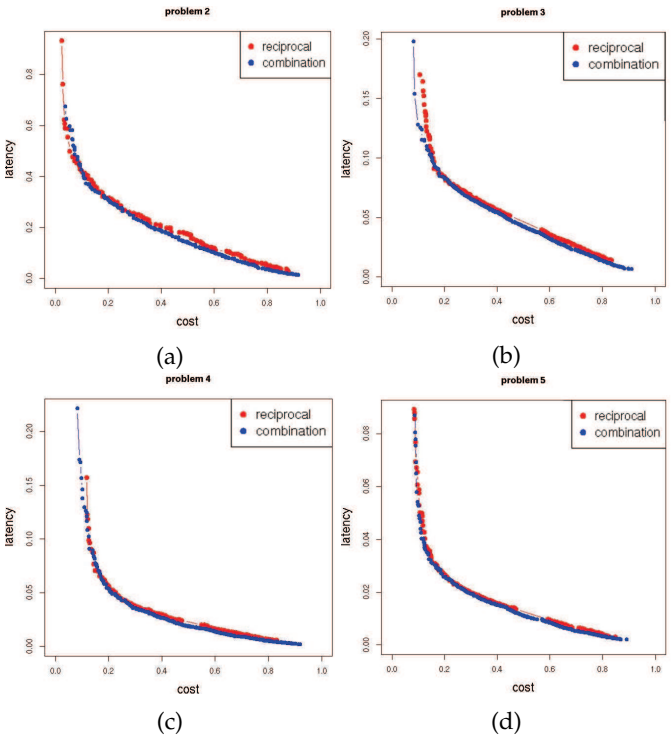


Fig. 5: Combination of Static Function Experiments: The non-dominated solutions among the sets obtained by 40 independent runs of combination of static thresholds and reciprocal function

datasets, BMOPSOCD achieved a considerably better performance than other three algorithms do 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 the three algorithms maintain the same size of 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 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 the best or the second best performance for 11 out of 14 problems (Table **??**).

In summary, from the experimental evaluation that comparing the proposed algorithm with previous approaches, we observe 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 MOPSOCD with dynamic rounding function is not affected by the size of variable. This is a significant advantage of BMOPSOCD over the other three approaches.

6 Conclusion and Future Work

This paper proposed a BMOPSOCD to solve the WSLA problem with the aim of producing a set of high quality solutions with good diversity that covers most of the Pareto front when dealing with large datasets. For that, we proposed a binary version of multi-objective PSO with crowding distance to solve the WSLA problem. 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.

There are a few directions that future work can work on. 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 that composed with atomic services distributed over the internet, we need to consider service composition workflow while doing WSLA. 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 in the literature [**?**]. In addition, future work can consider multiple constraints such as the overall cost constraints and bandwidth constraints.

TABLE 4: Comparison between BMOPSOCD, NSPSO, NSGA-II and BPSO: The non-dominated solutions among the sets obtained by 40 independent runs of different algorithms

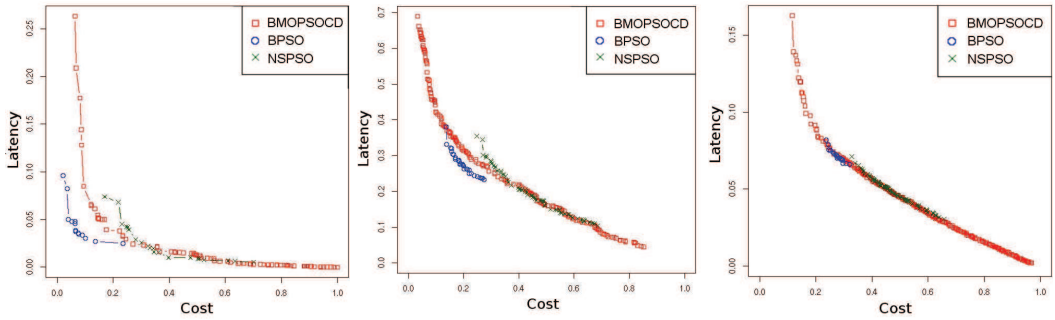
Dataset	Method	Hypervolume (avg ± sd)	IGD (avg ± sd)
problem 1	BMOPSOCD	0.83 ± 0.04	**3.73E-02 ± 1.03E-02**
	NSPSO	0.76 ± 0.018	0.16 ± 3.45E-02
	NSGA-II	0.83 ± 0.013	0.19 ± 3.21E-02
	BPSO	**0.89 ± 0.015**	0.46 ± 2.45E-02
problem 2	BMOPSOCD	**0.73 ± 0.011**	**3.15E-02 ± 7.92E-03**
	NSPSO	0.61 ± 0.001	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: Execution time

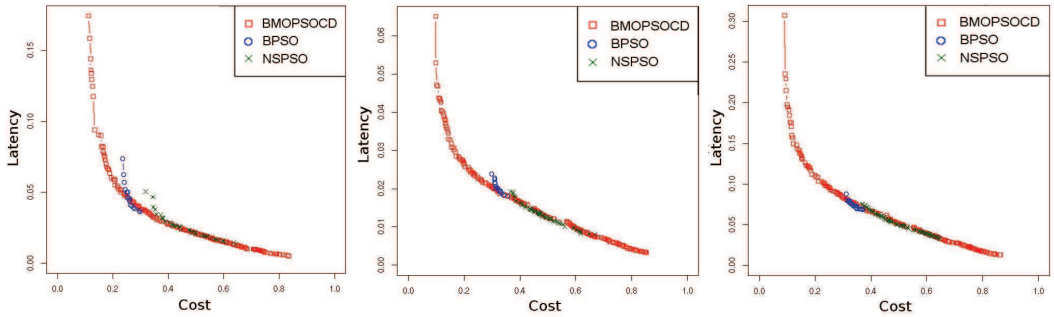
	method	time (avg ± sd)
problem 1	BPSO	17.99 ± 0.26
	BMOPSOCD	**12.98 ± 0.18**
	NSPSO	19.00 ± 0.17
	NSGA-II	15.35 ± 0.15
problem 2	BPSO	23.55 ± 0.27
	BMOPSOCD	**16.18 ± 0.26**
	NSPSO	25.52 ± 0.27
	NSGA-II	15.38 ± 0.31
problem 3	BPSO	103.65 ± 1.87
	BMOPSOCD	94.98 ± 7.28
	NSPSO	111.86 ± 1.11
	NSGA-II	**74.34 ± 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**

	method	time (avg ± sd)
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

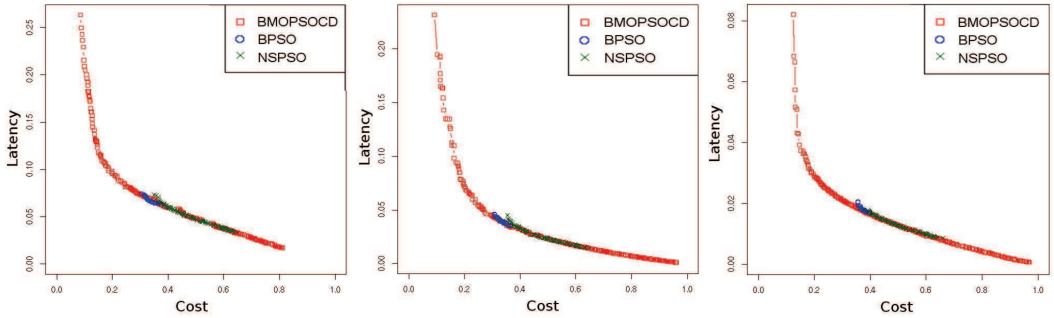
Fig. 6: MOPSOCD, NSPSO and BPSO Experiments: The non-dominated solutions among the sets obtained by 40 independent runs of different algorithms



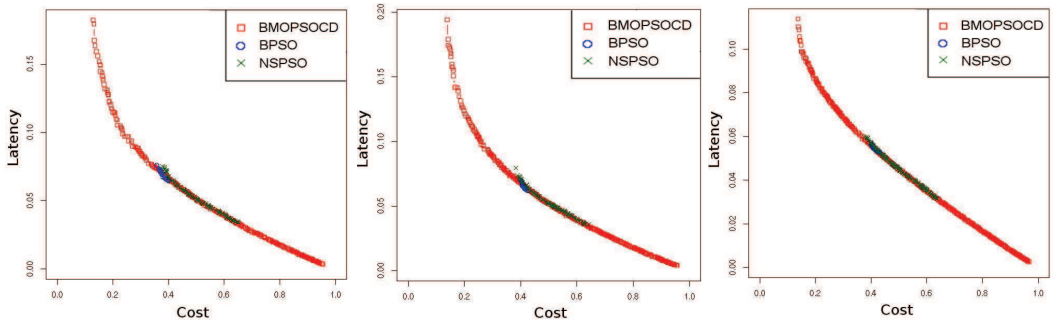
(a) Problem 1 (b) Problem 2 (c) Problem 3



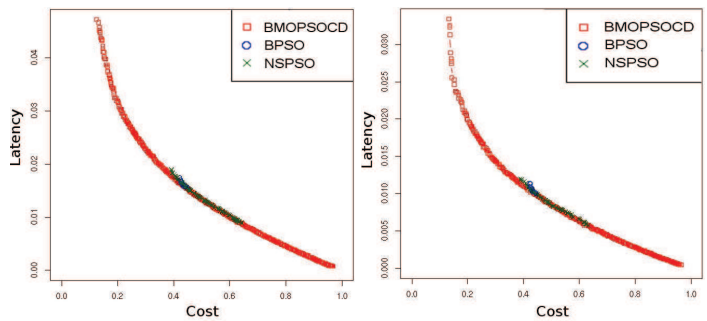
(d) Problem 4 (e) Problem 5 (f) Problem 6



(g) Problem 7 (h) Problem 8 (i) Problem 9



(j) Problem 10 (k) Problem 11 (l) Problem 12



(m) Problem 13 (n) Problem 14