Optimization of Location Allocation of Web Service using non-dominated sorting algorithm(NSGA-II)

Boxiong Tan, Hui Ma, Mengjie Zhang

School of Engineering and Computer Science,
Victoria University of Wellington, New Zealand
{Boxiong.Tan, Hui.Ma, Mengjie.Zhang}@ecs.vuw.ac.nz

Abstract.

1 Introduction

Web Services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web [20]. In recent years, web services technology is becoming increasingly popular because the convenience, low cost and capacity to be composed into high-level business processes [1]. Because of modularization and open interfaces, Web services facilitate the development of highly customizable and adaptable applications to meet business demands. Furthermore, Web services offer a convenient registration, search and discovery system. e.g. Universal Description, Discovery and Integration (UDDI).

With the ever increasing number of functional similar web services being available on the Internet, the web service providers (WSPs) are trying to improve the quality of service (QoS) to become competitive in the market. QoS also known as non-functional requirements to web services, is the degree to which a service meets specified requirements or user needs [26], 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 [15]. Study [14] has shown that network latency is a significant component of 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 [12]. The network latency could also vary with the network topology changes. The only way to reduce the network latency is move the service to a location where has lower network latency to the user center. Hence, the WSPs need to consider which physical locations to deploy their services so that it could minimize the cost as well as ensure the QoS.

The Web service location-allocation problem is essentially a multi-objective optimization problem [3], for which there are two conflict objectives, to provide optimal QoS to service users and to consume minimal deployment cost. Ideally,

WSP could deploy their services to each user center in order to provide the best quality. However, the more services deployed, the better the quality and the higher cost. This problem is considered as NP-hard due to the fact that the combinatorial explosion of the search space [22].

Very few researches study this problem, most of the researchers treat this problem as a single objective problem. [1] [21] try to solve the problem by using integer linear programming techniques. In particular, [21] solved this problem by employing a modified Alternate Location-Allocation (ALA) algorithm. However, essentially ALA build upon integer programming and greedy optimization. As a result, the solution is easily stuck at local optima. Huang [10] proposed an enhanced genetic algorithm (GA)-based approach which make use of the integer scalarization technique to solve this problem. GA [18] is an Evolutionary algorithm (EA) that uses genetic operators to obtain optimal solutions without any assumptions about the search space. This algorithm solves the problem with one objective and one constraint, however there are a few disadvantages existed in integer scalarization technique [3], which were discussed explicitly in Section 2.

Since the location-allocation should be considered as a multi-objective problem. Evolutionary multi-objective optimization (EMO) methodologies have to be considered. Evolutionary multi-objective optimization (EMO) methodologies are a subset of Evolutionary algorithms (EA), which have been used in solving multi-objective optimization problems in recent years. EMO are ideal for solving multi-objective optimization problems [7], since EMO works with a population of solutions, a simple EMO can be extended to maintain a diverse set of solutions. With an emphasis for moving toward the true Pareto-optimal region, an EMO can be used to find multiple Pareto-optimal solutions in one single simulation run [16]. Among numerous EMO algorithms, Non-dominated sorting GA (NSGA-II) [4], Strength Pareto Evolutionary Algorithm 2 (SPEA-2) [5] have become standard approaches. Some schemes based on particle swarm optimization approaches [8] [11] are also important. NSGA-II is one of the most widely used methods for generating the Pareto frontier. NSGA-II implements elitism and uses a phenotype crowd comparison operator that keeps diversity without specifying any additional parameters [6]. In this paper, we propose to use NSGA-II to solve the web service location-allocation problem.

To show the effective and efficiency of our proposed NSGA-II based approach, we conduct an experiment to compare our approach with traditional approach based on integer programming and greedy optimization (IPSO). IPSO models one objective as an assignment problem then uses integer programming techniques to obtain an optimal solution. Then, based on this solution, the algorithm further optimize the other objective using greedy algorithm. The major advantage of IPSO is that it is very efficient and gives an reasonably good result. On the other hand, the objectives are considered unevenly. That is, IPSO considers the objective which optimize by linear programming as its priority.

The aim of this research is to provide a framework that guides a WSP to locate services. We consider the problem faced by a WSP who has existing facilities but wishes to use the collected data to re-allocate their services in order

to maximum their profit. The WSP must decide on facility locations from a finite set of possible locations. In order to make the decision, the WSP must first analyze the data collected from current use of services. The collected data should includes the records of invocations from each unique IP address. Therefore, based on these data, the WSP could summarize several customer demands concentrated at n discrete nodes [1], namely user centers. We assume the WSP has already done this step and list of user centers and candidate service deployment locations are given. In addition to decide which location to re-allocate the services, a dataset which contains the network latency between demand a user center and a candidate location are critical. The WSP could collect the data or use existed dataset [24] [25]. Then, the service provider could use the algorithm which proposed by this paper, to select an optimal plan based on their funds. The algorithm will produce a near optimal solution which indicate the services deployment locations with a minimum cost and best service quality. The main objectives are:

- To model the web service location-allocation problem so that it can be tackled with NSGA-II
- To develop a modified NSGA-II approach for the web service location-allocation problem
- To evaluate our approach by comparing it to a linear assignment with greedy optimization (IPGO) algorithm.

In Section 2 we introduce the background of NSGA-II and IPGO as well as discuss the work from previous researchers. In Section 3 we provide a formulation for our model. Section 4 develops a modified NSGA-II. We use these to study a number of hypotheses on the placement of WSPs. These solutions are compared to solutions from Integer programming with greedy optimization(IPGO). The experimental design was discussed in Section 6. The experimental results are shown in Section 7.

2 Related Works and Background

A few researchers attempt to solve the service location-allocation problem by using single objective approach. Most of them solve this problem with linear programming approaches. [9] consider more than 3 objectives: response time, availability and cost. It solves the placement of Datacenters for Internet Services with simulated annealing plus linear programming. Sun et.al proposed two approaches [1] [21] that employ LP and ALA respectively. The first approach [1] assumes the model is based on a duopoly competitive market. The second approach [21] assumes multiple competition services have been established in the market. Some researchers use evolutionary algorithms. Huang [10] applied an enhanced GA algorithm which employ the integer scalarization technique [3]. However, the integer scalarization technique has many disadvantages:

1. The decision maker needs to choose an appropriate weights for the objectives to retrieve a satisfactorily solution.

2. The algorithm does not produce an uniform spread of points on the Pareto curve. That is, all points are grouped in certain parts of the Pareto front.
3. Non-convex parts of the Pareto set cannot be reached by minimizing convex combinations of the object functions.

So far, there is no research consider the service location-allocation problem as a multi-objective problem. Therefore, this paper is the first attempt to solve this problem using multi-objective algorithm.

2.1 NSGA-II

NSGA-II [4] belongs to the larger class of genetic algorithm (GA). GA [18] is a powerful tool to solve combinatorial optimization problems. It is an iterative procedure based on a constant-size population. In a GA, a population of strings (called chromosomes or the genotype of the genome), which are encoded as candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, evolves towards better solutions. Each genome is associated with a fitness value based on a fitness function that indicates how close it comes to meeting the overall specification, when compared to other genomes in the population. The fitness value of an individual is also an indication of its chances of survival and reproduction in the next generation. A typical genetic algorithm requires a genetic representation of the solution domain and a fitness function to evaluate the solution domain. Since a chromosome from the population represents a solution, when the algorithm starts, the whole population moves like one group towards an optimal area, so the GA searches from a population of solutions rather than a single solution.

NSGA-II is a multi-objective algorithm based on GA. When used for problems with only two objectives, NSGA-II performs relatively well in both convergence and computing speed. It permits a remarkable level of flexibility with regard to performance assessment and design specification. NSGA-II assumes that every chromosome in the population has two attributes: 1) a non-domination rank in the population, 2) a local crowding distance in the population. The goal of NSGA-II is to converge to the Pareto front as possible and with even spread of the solutions on the front by controlling the two attributes.

The algorithm starts with initialization population. Once the population is sorted based on non-domination sorting, rank is assigning to each chromosome. Then, a parameter called crowding distance is calculated for each individual. The crowding distance is a measure of how close an individual is to its neighbors. Large average crowding distance will result in better diversity in the population.

Parents are selected from the population by using tournament selection based on the rank and crowding distance. An individual is selected in the rank is lesser than the other or if crowding distance is greater than the other. The selected population generates offsprings from crossover and mutation operators.

The population with the current population and current offsprings is sorted again based on non-domination and only the best N individuals are selected, where N is the population size. The selection is based on rank and the on crowding distance on the last front.

2.2 Integer programming with Greedy Optimaization (IPGO)

Linear assignment [17] is a special case of the transpotation problem, which is a special case of the minimum cost flow problem, which in turn is a special case of a linear program.

Much research has been devoted to develop efficient heuristic algorithm to solve location-allocation problem. Two of the predominant greedy construction heuristics are the ADD and DROP [21] methods, which build solutions from scratch. The ADD procedure starts with all facilities closed and locates a facility that gives the greatest savings at each step and iterates through all potential locations until no further savings can be made. In contrast, DROP opens all facilities and removes a location that gives the greatest savings at each step and tries to find optimal or near optimum solutions at the end of the iterative procedure. Integer programming with greedy optimization(IPGO) builds upon ADD and DROP methods. In order to solve multi-objective problem, IPGO first employ Integer programming to derive a minimum cost solution. Then, it uses a greedy procedure to optimize the other objective.

3 Problem Description and Modeling

In this section, we first describe the service location-allocation problem in details, then we will present models for the services location allocation problem.

3.1 Problem Description

Web service location-allocation problem is to determine reasonable locations for web services so that deployment cost of WSP can be minimized while service performance can be optimized. In this paper, to optimize service performance we consider to minimize network latency.

The task of service location allocation has two objectives:

- To minimize the total cost of the services.
- To minimize the total network latency of the services.

3.2 Model Formulation

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

For service location-allocation problem we needs information of service usage, network latency, and service deployment cost to decide service location-allocation so that the overall network latency can be minimized with minimal deployment cost and with constraint. Assume a set of $S = \{s_1, s_2, ..., s_x\}$ services are requested from a set of location $I = \{i_1, i_2, ..., i_i\}$. The service providers allocate services to $J = \{j_1, j_2, ..., j_j\}$ be the set of candidate facility locations. To model

the service location-allocation problem we use five matrices: service network latency matrix L, service location matrix A, service invocation frequency matrix F, cost matrix C and user response time matrix R.

In the following we list a set of notations that will be used for modelling.

Sets	
S	set of service $S = \{s_1, s_2, ..., s_x\}$
I	set of user center $I = \{i_1, i_2, ..., i_y\}$
J	set of candidate location $J = \{j_1, j_2, ..., j_z\}$
Matrices	
L	server network latency matrix $L = \{l_{ij}\}$
A	service location matrix $A = \{a_{sj}\}$
F	service invocation frequency matrix $F = \{f_{is}\}$
C	cost matrix $C = \{c_{sj}\}$

The service invocation frequency matrix $F = [f_{is}]$, is used to record services invocation frequency from user centers, which f_{is} is an integer that indicate the number of invocation in a period of time from user center to service. For example, $f_{13} = 85$ denotes service s_1 is invocated 85 times in a predefined period of time.

$$F = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \end{array} \begin{array}{c} \begin{array}{ccc} s_1 & s_2 & s_3 \end{array} \\ \begin{bmatrix} 120 & 35 & 56 \\ 14 & 67 & 24 \\ 85 & 25 & 74 \end{bmatrix} \end{array}$$

The network latency matrix $L = [l_{ij}]$, is used to record network latency from user centers to candidate locations. For example, The network latency between user center i_2 with candidate location j_1 is 5.776s. These data could be collected by monitoring network latency [24] [25].

$$L = \begin{array}{c} \\ i_1 \\ i_2 \\ i_3 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 0 & 5.776 & 6.984 \\ 5.776 & 0 & 2.035 \\ 0.984 & 1.135 & 2.3 \end{bmatrix} \end{array}$$

The cost matrix $C = [c_{sj}]$, is used to record the cost of deployment of services from candidate locations, which c_{sj} is an integer that indicate the cost of the deploying services from a candidate location. For example, $c_{12} = 80$ denotes the cost of deploying service s_1 to location j_2 is 80 units.

$$C = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \end{array} \begin{array}{c} \begin{array}{ccc} j_1 & j_2 & j_3 \end{array} \\ \begin{bmatrix} 130 & 80 & 60 \\ 96 & 52 & 86 \\ 37 & 25 & 54 \end{bmatrix} \end{array}$$

To model the service location-allocation problem we Consider the following assumptions:

1. The new WSP decides where to locate his facilities regardless if there is existed functional similar services from other WSPs.

2. The decision of service location-allocation is made only considering two factors: total network latency and total cost.
3. A static allocation policy is used by WSPs. In practice, Web Services typically offer clients persistent and interactive services, which often span over multiple sessions. Therefore, a dynamic reallocation scheme is not practical as it may disrupt the continuity of the services.

The service location-allocation matrix $A = [a_{sj}]$ represents the actual service location-allocation, where a_{sj} is a binary value i.e. 1 or 0.

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

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

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

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

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

4 NSGA-II For Web Services Location Allocation

To apply NSGA-II to the service location-allocation problem, the first step is to define the variable in NSGA-II, i.e. to identify chromosome, genetic operators and the fitness functions.

4.1 Chromosome Representation and Constraints

We use the service location matrix A = $[a_{sj}]$ as the representation of chromosome that we mentioned in Section 3.

The constraint setting is based on service providers' needs. One can set multiply constraints to the problem to narrow the potential searching space. In our case, we set two basic constraints: The first constraint guarantees that each service is deployed in at least one location.

$$\sum_{x \in S} a_{xj} \geq 1 \tag{2}$$

The second constraint is the cost constraint which predefined the upper boundary of the total cost. An integer number $CostLimitation$ is defined.

$$\sum_{s \in S} \sum_{j \in J} C_{sj} \times A_{sj} \leq CostLimitation \tag{3}$$

4.2 Genetic Operators

The original NSGA-II uses a simulated binary crossover (SBX) [2] and polynomial mutation [19] to cope with continuous problem. However, our problem is discretized, therefore we use the regular GA mutation and crossover operations.

Mutation The mutation operator works as follows: Initially choose one random location from the chromosome. Then, reverse the bit, e.g. 0 to 1 or 1 to 0.

In this study, a chromosome in a population will be selected based on mutation possibility P_m. As shown in below, a random position will be selected and replaced by a reversed number.

Fig. 1

Crossover The crossover operator in this paper is the single point crossover. The crossover is controlled by crossover probability P_c. The crossover point is created randomly within the length of the chromosome. As in the example below, two parents crossover at a point, then two offspring were generated.

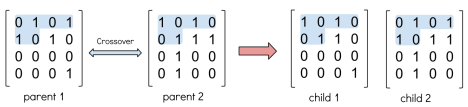


Fig. 2

Fitness Function

In order to accomplish these two objectives. We design two fitness functions to evaluate how good each chromosome meets the objectives.

Cost fitness function

$$CostFitness = \sum_{s \in S} \sum_{j \in J} c_{sj} \times a_{sj} \tag{4}$$

C_{sj} is the Cost matrix stores the cost information of service s at location j. A_{sj} represents the deployment plan. The sum of the multiplication of C_{sj} and A_{sj} derives the total cost of the deployment plan.

Network latency fitness function There are some assumptions we made of network latency. First, we assume the latency = 0 if the service is deployed locally. That is, if the service is deployed in the user center i. Then, the invocation from i to the service has no latency. The second assumption is the latency is symmetrical between user center and candidate location. e.g. The latency from location i to location j is same with the latency from location j to location i. Thirdly, the total invocation are equally contributed by user centers. In other words, we assume each user center has same invocation to a service.

There are five steps to calculate the total network latency.

1. Firstly, for each chromosome, calculate the number of invocation for each service. $Invocation_s$ is a vector which contains the number of invocation for each service.
$$Invocation_s = \sum_{i \in I} F_{is} \ s \in S,$$
2. Calculate total number of deployment for each service. $ServiceNo_s$ is a vector which contains the number of deployment for each service.
$$ServiceNo_s = \sum_{s \in S} A_{sj}$$
3. Calculate the average number of invocation for each user center contribute to the services.
$$AverageInvocation_i = Invocation_s \div \mathrm{s} \div ServiceNo_s$$
4. Calculate the latency of each service.
$$LocationLatency_s = \sum_{i \in I} L_{ij}$$
5. Calculate the total latency of the multiplication of $AverageInvocation_i$ and $LocationLatency_i$. If the service is deployed locally, then the latency equals zero.
$$\text{Check if deployed locally: } LocationLatency_s = 0$$

$$LatencyFitness = \sum_{s \in S} \sum_{i \in I} AverageInvocation_i \times LocationLatency_s \tag{5}$$

4.3 NSGA-II based algorithm for service location-allocation

Algorithm 1 shows the precedure of NSGA-II applied on service location-allocation problem.

Algorithm 1 NSGA-II for service location-allocation

```
Initialize a population of chromosome with random binary value
Evaluate population with fitness functions
Non-dominated sort and assign ranking to each chromosome
Evaluate the Crowding distance of each chromosome
Initialize the Pareto Front Pool
while predefined generation do
    Apply Tournament Selection
    Apply Crossover
    Apply Mutation
    for ( do each chromosome)
        while  violate service number constraint do
            Deploy the service to a random candidate location
        end while
        while  violate cost constraint do
            Close down a service from a random location as long as it does not violate service number constraint
            if  no service can be closed then
                Break
            end if
        end while
        if  chromosome does not exist in the Pareto front Pool then
            Evaluate with fitness functions
        end if
        Non-dominated sort and assign ranking
        Evaluate the Crowding distance
    end for
    Recombination and Selection
    Update Pareto Front Pool with current Pareto Front
end while
```

There are two major differences between standard NSGA-II and modified version. Firstly, in order to avoid repeat evaluation of the fitness of chromosome which contribute to the majority of computation. After the first generation is initialized, the Pareto front will be stored in the memory. In each generation, when evaluate the chromosome, the chromosome will be checked whether it is existed in the memory pool. If so, then the calculation of fitness will be skipped. At the end of each iteration, the Pareto front pool will be updated to the current Pareto front. The reason for setting a memory pool is that, after a number of iteration, the population start converging. Most of the population are redundant. Therefore, a memory pool could considerably reduce the repeat computation.

The other difference is that, we use general mutation and crossover operation instead of polynomial mutation and simulated binary crossover. It is important to note that mutation and crossover operators can produce solutions that might violate the constraints. Therefore, repair operators are needed to try to maintain feasible solutions. The repair operators are essentially while loop which repeatedly check if the chromosome violates constraints.

Worth noting that the cost constraint repair operator may not provide a strictly correct solution that satisfied the cost constraint.

5 IPGO for Web Service Location Allocation

In order to solve the location allocation problem with IPGO. We model the model the cost objective as the priority, optimize with Linear programming. The cost matrix C and solution matrix A are identical with the two matrices defined in Section 3. We use the same fitness functions as in NSGA-II to evaluate the Cost objective and Latency objective.

$$\begin{aligned} &\text{Minimize } \sum_{s\in S}\sum_{j\in J} C_{sj} \times A_{sj} \\ &\text{Subject to} \\ &\qquad \sum_{s} A_{sj} = 1(s = 1, 2, ...), \\ &\qquad \sum_{s\in S}\sum_{j\in J} C_{sj} \times A_{sj} \leq CostLimitation \\ &\text{and } A_{sj} = 0 \quad \text{or} \quad 1(s, j \quad = \quad 1, 2, ...) \end{aligned} \tag{6}$$

Once a minimum cost solution is found by linear assignment, the solution set will be passed to the ADD procedure to serve as an initial solution for further improvement on Network latency.

The ADD procedure as follows:

- Evaluate network latency fitness of the initial solution.
- Starts a facility then evaluate with network latency fitness function. If the network latency fitness decreased as well as the cost fitness does not exceed the cost constraint. The solution is kept.
- If there is no location could be choose, terminate the procedure.

6 Experiment Design

The purpose of the experiment is comparison between Integer programming and greedy optimization (IPGO) with modified NSGA-II in terms of efficiency and quality of solutions.

The exact algorithm was coded in R using packages: NSGA2R and LpSolve. The program was run on a 3.40GHz desktop computer with 8 GB RAM.

To compare with IPGO, four different datasets were used, the problem instance was generated as follows:

1. The potential user center and candidate locations were randomly selected from existed network latency dataset.
2. The number of potential user center equals the candidate locations. We defined 4 different size of allocation matrices: $3 \times 3, 5 \times 5, 10 \times 10$ and 15×15.
3. Network latency L_{ij} were selected from public WS-DREAM dataset [24] [25].The number of service are also set as: 3, 5, 10, 15.
4. The cost matrix was randomly generate from a normal distribution which mean = 100 with standard deviation = 20
5. The frequency matrix was an integer matrix which randomly generate from a uniform distribution on [1, 120]

In each dataset, algorithms run under four different levels of cost constraint: Sufficient condition is 100% the expected total cost, good condition (80%), pool condition (40%) and minimum budget condition (0%). In the minimum budget condition, both algorithm exhaustively reduce the cost until it reaches the service number constraint. The NSGA-II runs 40 times with different random seed ranging from 1 to 40. In efficiency test, We evaluate the average run time for each algorithm.

Parameter settings for application of NSGA-II are shown in Table 1. The terminal condition is that the population has evolved 50 generation.

Table 1: NSGA-II Parameters

Population	Generation	Tour Size	Crossover Rate	Mutation Rate
50	50	10	0.8	0.2

6.1 Case Study

To compare with the result of NSGA-II and IPGO, we derive the Pareto front by using Xue's approach [23]. Then take the approach from [13] to compare the results. In Xue's approach, NSGA-II generate 40 results (from 40 runs) under each cost constraint are presented in the Section 7.2. 40 sets of solutions achieved by each multi-objective algorithm are firstly combined into one union set. In the union set, the non-dominated solutions are presented to compare with the solutions achieved by IPGO.

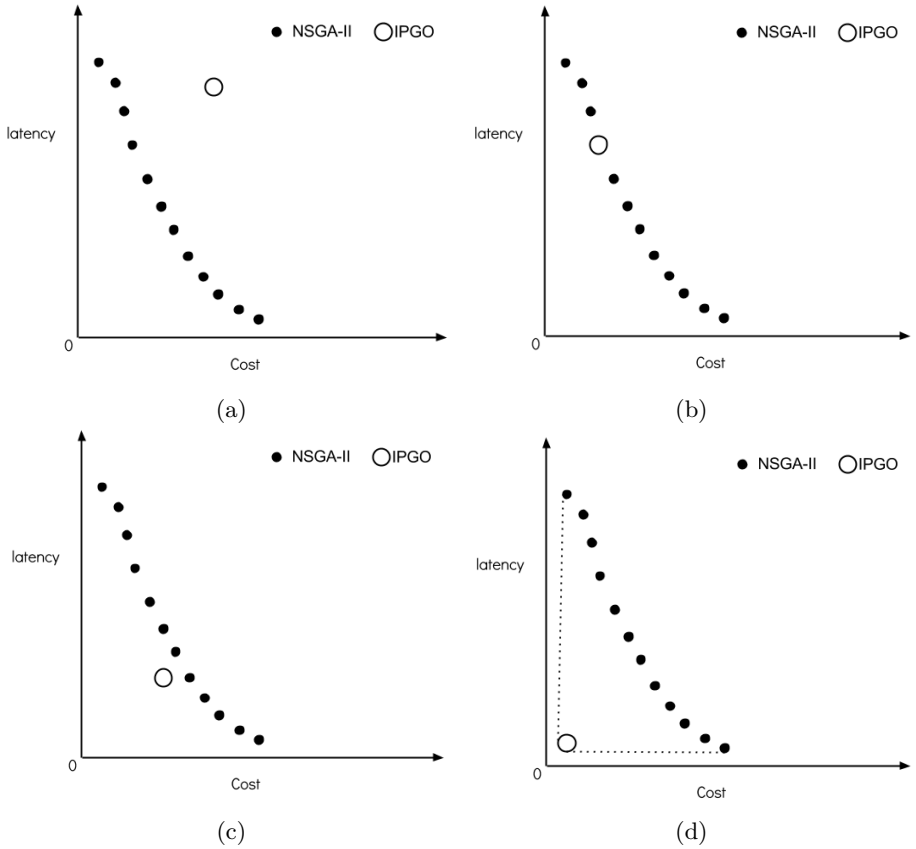


Fig. 3: Four cases of the relation between a IPGO solution and a NSGA-II solution set of non-dominated solutions

The relation between a single solution and a set of non-dominated solutions from NSGA-II can be classified into four cases as shown in Figure 3. (a), the IPGO solution is dominated by some solutions in NSGA-II solution set. In this case, we can say that NSGA-II outperforms IPGO since the inclusion of the IPGO solution does not improve the quality of the NSGA-II solution set. On the other hand, we can say that IPGO outperforms NSGA-II in Figure 3 (d) Since the IPGO solution dominates all solutions in the NSGA-II solution set. In Figure 3 (b), the IPGO solution is non-dominated solution in the NSGA-II solution set. While the inclusion of the IPGO solution somewhat improves the quality of the NSGA-II solution set, we cannot say that IPGO outperforms NSGA-II since the IPGO solution dominates no NSGA-II solutions. We may intuitively say that NSGA-II outperforms IPGO in the case of Figure 3 (b).

It is difficult to say which is better between IPGO and NSGA-II in 3(c) where IPGO solution dominates some NSGA-II solutions. If we use performance

measures that are strongly related to the convergence of solutions, IPGO maybe evaluated as being better than NSGA-II. On the contrary, if we related strongly to the diversity of solutions, NSGA-II may be evaluated as being better than IPGO.

7 Experimental results

7.1 Efficiency comparison

Table 2: Efficiency Test

Table 3: Sufficient condition

Matrix Size	GA time(s)	IPGO time(s)
3×3	4.21 ± 0.18	0.013
5×5	6.04 ± 0.24	0.22
10×10	14.55 ± 0.26	0.78
15×15	28.18 ± 0.54	4.02

Table 4: Good condition

Matrix Size	GA time(s)	IPGO time(s)
3×3	4.21 ± 0.28	0.007
5×5	6.02 ± 0.26	0.057
10×10	14.4 ± 0.26	0.76
15×15	28.1 ± 0.52	3.83

Table 5: Poor condition

Matrix Size	GA time(s)	IPGO time(s)
3×3	4.82 ± 0.27	0.01
5×5	6.755 ±0.21	0.06
10×10	16.73 ± 0.67	0.773
15×15	36.0 ± 0.7	3.85

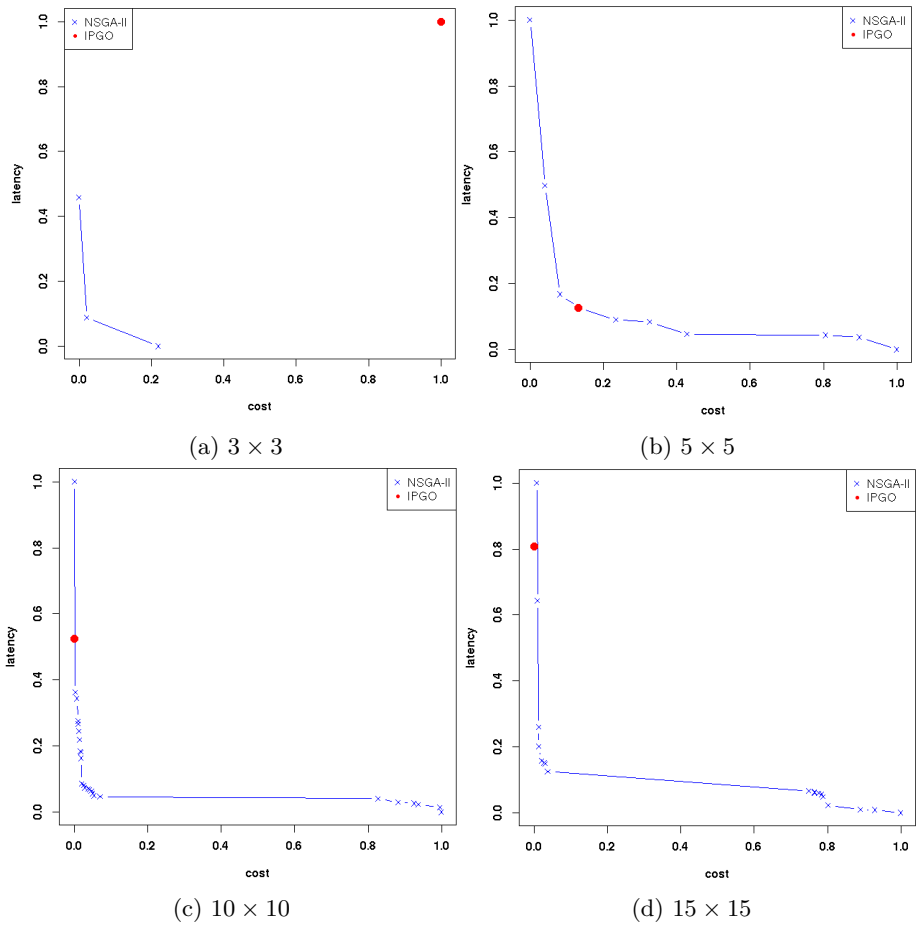
Table 6: Minimum condition

Matrix Size	GA time(s)	IPGO time(s)
3×3	4.69 ± 0.31	0.017
5×5	8.0 ± 0.26	0.06
10×10	23.5 ± 1.36	0.92
15×15	58.45 ± 1.9	3.83

As the experimental results show, in terms of the average run time, IPGO is clearly better than NSGA-II. There are two main reasons: firstly, it is fast to solve linear assignment for integer programming, because there is only one constraint. Second, repeat evaluation of chromosome is the decisive factor of time consuming. Although, the modified NSGA-II tries to avoid unnecessary evaluation by employing the idea of memory pool, the improvement is limited.

It is worth nothing that in sufficient condition and good condition, both algorithm's run time remain stable. The run time starts increasing under poor condition and minimum condition with NSGA-II. In particular, the run time for 15×15 matrix under minimum condition is more than twice as under other conditions. The reason is that, if the children exceed the cost constraint, the repair operator will exhaustively close redundant facilities until it satisfied the cost limitation or minimum facility number is achieved. If the cost constraint is set too low, the number of iteration in the repair processes will reach a maximum number. Therefore, the time consuming increases largely.

7.2 Effective comparison



(a) 3×3 (b) 5×5

(c) 10×10 (d) 15×15

Fig. 4

We conducted 16 experiments on 4 datasets under 4 cost constraints: sufficient, good, poor and minimum. The results show similar patterns for same dataset. Therefore, we only show one figure for each dataset.

Figure 4a and 4b show NSGA-II is outperforms IPGO. As we discussed in the previous section, the result of IPGO is dominated by NSGA-II's solution set. Although, IPGO solution for 3×3 matrix is quite close to the Pareto front. 4d shows that solution of IPGO is on the Pareto front. It does not improve the quality of the solution set. Therefore, we considered NSGA-II outperforms IPGO in this case. 4c is a special case, NSGA-II solution set does not cover the

possible solution range. That is, we predefined 50 generations for NSGA-II is not big enough for NSGA-II to find a solution set that cover the Pareto front. The solution from IPGO is non-dominated solution in the NSGA-II solution set. While the inclusion of the IPGO solution somewhat improves the quality of the NSGA-II solution set, we cannot say that IPGO outperforms NSGA-II since the IPGO solution dominates no NSGA-II solutions.

As a whole, we can conclude that NSGA-II outperforms IPGO in terms of the quality of solutions.

8 Conclusion

In this paper, we proposed a NSGA-II based approach to web service location-allocation problem. Our approach utilizes a memory pool so that greatly reduces the evaluation time while the diversity of the population gradually decreases. We have conducted a full experimental evaluation using the public WS-DREAM dataset to compare our approach to Alternate Location Allocation algorithm. The experimental results shows the modified NSGA-II is effective to provide a near-optima solutions for the web service location-allocation problem.

References

1. Aboolian, R., Sun, Y., Koehler, G.J.: A locationallocation problem for a web services provider in a competitive market. European Journal of Operational Research 194(1), 64 – 77 (2009)
2. Beyer, H.G., Deb, K.: On self-adaptive features in real-parameter evolutionary algorithms. Evolutionary Computation, IEEE Transactions on 5(3), 250–270 (2001)
3. Caramia, M.: Multi-objective optimization. In: Multi-objective Management in Freight Logistics, pp. 11–36. Springer London (2008)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. Evolutionary Computation, IEEE Transactions on 6(2), 182–197 (2002)
5. Deb, K., Mohan, M., Mishra, S.: Evaluating the epsilon-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions. Evol. Comput. 13(4), 501–525 (2005)
6. Deb, K., Sundar, J., N, U.B.R., Chaudhuri, S.: Reference point based multi-objective optimization using evolutionary algorithms. In: International Journal of Computational Intelligence Research. pp. 635–642. Springer-Verlag (2006)
7. Desai, S., Bahadure, S., Kazi, F., Singh, N.: Article: Multi-objective constrained optimization using discrete mechanics and nsga-ii approach. International Journal of Computer Applications 57(20), 14–20 (2012), full text available
8. Elhossini, A., Areibi, S., Dony, R.: Strength pareto particle swarm optimization and hybrid ea-pso for multi-objective optimization. Evol. Comput. 18(1), 127–156 (2010)
9. Goiri, I., Le, K., Guitart, J., Torres, J., Bianchini, R.: Intelligent placement of datacenters for internet services. In: Distributed Computing Systems (ICDCS), 2011 31st International Conference on. pp. 131–142 (2011)

10. Huang, H., Ma, H., Zhang, M.: An enhanced genetic algorithm for web service location-allocation. In: Decker, H., Lhotsk, L., Link, S., Spies, M., Wagner, R. (eds.) Database and Expert Systems Applications, Lecture Notes in Computer Science, vol. 8645, pp. 223–230. Springer International Publishing (2014)
11. Huang, V.L., Suganthan, P.N., Liang, J.J.: Comprehensive learning particle swarm optimizer for solving multiobjective optimization problems: Research articles. Int. J. Intell. Syst. 21(2), 209–226 (2006)
12. Huffaker, B., Fomenkov, M., Plummer, D., Moore, D., claffy, k.: Distance Metrics in the Internet. In: IEEE International Telecommunications Symposium (ITS). pp. 200–202. IEEE, Brazil (Sep 2002)
13. Ishibuchi, H., Nojima, Y., Doi, T.: Comparison between single-objective and multi-objective genetic algorithms: Performance comparison and performance measures. In: Evolutionary Computation, 2006. CEC 2006. IEEE Congress on. pp. 1143–1150 (2006)
14. Jamin, S., Jin, C., Kurc, A., Raz, D., Shavitt, Y.: Constrained mirror placement on the internet. In: INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. vol. 1, pp. 31–40 vol.1 (2001)
15. Johansson, J.M.: On the impact of network latency on distributed systems design. Inf. Technol. and Management 1(3), 183–194 (2000)
16. Kanagarajan, D., Karthikeyan, R., Palanikumar, K., Davim, J.: Optimization of electrical discharge machining characteristics of wc/co composites using non-dominated sorting genetic algorithm (nsga-ii). The International Journal of Advanced Manufacturing Technology 36(11-12), 1124–1132 (2008)
17. Lawler, E.L.: The quadratic assignment problem. Management science 9(4), 586–599 (1963)
18. Man, K.F., Tang, K.S., Kwong, S.: Genetic algorithms: concepts and applications. IEEE Transactions on Industrial Electronics 43(5), 519–534 (1996)
19. Raghuwanshi, M.M., Kakde, O.G.: Survey on multiobjective evolutionary and real coded genetic algorithms. In: Proceedings of the 8th Asia Pacific symposium on intelligent and evolutionary systems. pp. 150–161 (2004)
20. Ran, S.: A model for web services discovery with qos. SIGecom Exch. 4(1), 1–10 (2003)
21. Sun, Y., Koehler, G.J.: A location model for a web service intermediary. Decis. Support Syst. 42(1), 221–236 (2006)
22. Vanrompay, Y., Rigole, P., Berbers, Y.: Genetic algorithm-based optimization of service composition and deployment. In: Proceedings of the 3rd International Workshop on Services Integration in Pervasive Environments. pp. 13–18. SIPE '08, ACM (2008)
23. Xue, B., Zhang, M., Browne, W.N.: Multi-objective particle swarm optimisation (pso) for feature selection. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation. pp. 81–88. GECCO '12, ACM (2012)
24. Zhang, Y., Zheng, Z., Lyu, M.: Exploring latent features for memory-based qos prediction in cloud computing. In: Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on. pp. 1–10 (2011)
25. Zheng, Z., Zhang, Y., Lyu, M.: Distributed qos evaluation for real-world web services. In: Web Services (ICWS), 2010 IEEE International Conference on. pp. 83–90 (2010)
26. Zhou, J., Niemela, E.: Toward semantic qos aware web services: Issues, related studies and experience. In: Web Intelligence, 2006. WI 2006. IEEE/WIC/ACM International Conference on. pp. 553–557 (2006)