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
{tanboxi, 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 [17]. 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 demand. 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 [23], 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 [14]. Study [13] 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 [11]. 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 smaller network latency to the user center. Hence, the WSPs need to consider which physical location 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]. Because of the confliction between service quality and deployment cost. Ideally, WSP could deploy their services to each user center in

order to provide the best quality. That is, the more services deployed, the better the quality and the higher cost. This problem is considered as an NP-hard due to the fact that the combinatorial explosion of the search space [19].

Very few researches [1] [18] study this problem. Both studies try to solve the problem by integer linear programming techniques. In particular, [18] has solved this problem by employ a modified Alternate Location-Allocation (ALA) algorithm. However, essentially ALA build upon integer programming and use greedy algorithm to optimize the result. As a result, the solution is easily stuck at local optima.

Evolutionary algorithms (EAs) have been used in solving multi objective optimization problems in recent years. EAs are ideal for solving multi objective optimization problems [7], since EA works with a population of solutions, a simple EA can be extended to maintain a diverse set of solutions. With an emphasis for moving toward the true Pareto-optimal region, an EA can be used to find multiple Pareto-optimal solutions in one single simulation run [15].

Huang [9] proposed an enhanced genetic algorithm-based approach which make use of the integer scalarization technique to solve this problem. The genetic algorithm (GA) is an 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 the integer scalarization technique [3] has some 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.

Evolutionary multi objective optimization (EMO) methodologies on the other hand, successfully avoid the above mentioned problems and demonstrated their usefulness in finding a well-distributed set of near Pareto optimal solutions [1]. 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] [10] are also important. Among numerous EA approaches, 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 our approach, we apply a modified version of NSGA-II since the web service location-allocation is a discrete problem.

The main purpose of this research is to provide a framework that guides a WSP to locate its servers. 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 which were collected from current 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 demand 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 user center and candidate location are critical. The WSP could collect the data or use existed dataset [22] [21]. 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 GA approach which uses integer scalarization technique.

In Section 2 we provide a formulation for our model. Section 3 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 ALA. The experimental design was discussed in Section 4. The experimental results are shown in Section 5.

2 Problem Description

2.1 Model formulation

The problem is to determine which facility locations that could maximus WSPs' profit as well as ensure low network latency. Let $S = \{1, 2, ..., s\}$ be the set of services. We assume that the demand for service is concentrated at i demand nodes $I = \{1, 2, ..., i\}$. Let $J = \{1, 2, ..., j\}$ be the set of j candidate facility locations. To model the service location-allocation problem we use four matrices: service network latency matrix L, service location matrix A, service invocation frequency matrix F and cost matrix C.

The server network latency matrix $L = [l_{ij}]$, is used to record network latency from user centers to candidate locations, where l_{ij} is a real number denotes the network latency from user center i to candidate location j. These data could be retrieved from implementing a network latency experiment or using existed datasets [22] [21].

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

The service location matrix $A = [y_{sj}]$ represents the actual service location-allocation, where y_{sj} is a binary value (i.e., 1 or 0) shows whether a service s is deployed in candidate location j or not. We use the service location matrix A as the representation of chromosome that evolve itself during the progress.

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

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 i to service s. e.g. 120 invocations per day from user center i_1 to s_1.

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

The 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 deployment cost from a candidate location. e.g $c_{11} = 130$ units to deploy s_1 from j_1.

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

Consider the following key modeling assumptions:

1. The new WSP decides where to locate his facilities regardless if there is existed functional similar services from other WSPs.
2. This choice is made only considering two factors: total network latency and total cost.
3. We assume a fixed customer allocation policy is for 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.

3 Modified NSGA-II algorithm

NSGA-II belong to the larger class of evolutionary algorithms (EAs), which generate approximate solutions to optimization and search problems by using techniques inspired by the principles of natural evolution: selection, crossover and mutation.

Algorithm 1 modified NSGA-2 algorithm

```
Initialize a population of chromosome with random binary value
Evaluate 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
    Check Constraint
    for ( doeach chromosome)
        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
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 check 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 will be replaced by the latest 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.

3.1 Chromosome Representation

In our problem, the chromosome is the service location matrix A that we mentioned in the previous section.

3.2 Objectives and Fitness Function

The objective functions of this entire problem are following:

– Minimize the total cost of services.

$$CostFitness = \sum_{s \in S} \sum_{j \in J} C_{sj} \times A_{sj} \tag{1}$$

– Minimize the network total latency of the services. 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 A. Then, the invocation from A to the service has 0 latency. The second assumption is the latency is symmetrical between cities. e.g. The latency from city A to city B is same with the latency from city B to city A. Thirdly, the total invocation are equally contributed by user centers.

- For each chromosome, firstly, calculate the number of invocation for each service.

$$Invocation_s = \sum_{i \in I} F_{is}$$

- Calculate the total number of each services.

$$ServiceNo_s = \sum_{s \in S} \sum_{j \in J} A_{sj}$$

- Calculate the number of each service deployed in each location.

$$ServiceNo_{sj} = \sum_{j \in J} A_{sj}$$

- Calculate the average number of each user center contribute to the total number of invocation of each services.

$$AverageInvocation_i = Invocation_s \div ServiceNo_s \div ServiceNo_{sj}$$

- Calculate the latency of each service by multiply latency matrix L by service allocation matrix A(chromosome).

$$LocationLatency_i = \sum_{i \in I} L_{ij} \times A_{sj}$$

- Calculate the total latency of the multiplication of $AverageInvocation_i$ and $LocationLatency_i$. If the service is deployed locally, then the latency equals zero.

Check if deployed locally: $LocationLatency_i = 0$

$$LatencyFitness = \sum_{s \in S} \sum_{i \in I} AverageInvocation_i \times LocationLatency_i \tag{2}$$

The population is initialized based on the problem range and constraints. In our problem, we have two constraints, the cost constraint and minimum number of service constraint. If the initialized chromosome does not satisfy the constraints, then repair operators will attempt to recover from possible constraint violations. The detail of repair operators will be discuss in next section.

3.3 Constraints

Each chromosome needs to satisfy two constraints to be a feasible solution. The first constraint controls the service number which is make sure that each service is deployed in at least one location.

$$\sum_{s \in S} A_{sj} \geq 1 \tag{3}$$

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

$$\sum_{s \in S} \sum_{j \in J} C_{sj} \times A_{sj} \leq CostLimitation \tag{4}$$

3.4 Genetic operators

The original NSGA-II use a simulated binary crossover (SBX) [2] and polynomial mutation [16] 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.

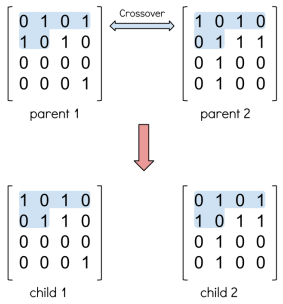


Fig. 2

Repair operators Since mutation and crossover are very likely to generate offspring that violate the constraints, therefore, repair operators are necessary. Normally, each constraint would has an unique repair operator in order to recover different types of violation.

After the process of mutation and crossover, the repair operators are examine each chromosome. If there violation found in the chromosome, then it will try to repair it. In our problem, we have two repair operators: service number and cost.

Service number repair operator

$$\sum_{s \in S} A_{sj} \geq 1$$

service number constraint

The operator will go through each row of the Location Allocation matrix A, if the number of each service is less than one, then randomly choose one location and reverse the selected bit. Although the randomness does not necessarily provide an optimal solution, the computation time is better than exhaustively compare all solutions.

Cost repair operator

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

cost constraint

If the cost exceed the predefined limitation. The operator would iteratively check if any service has been deployed in more than one location. After found one service has been deployed multiple locations, the operator will randomly select one of them and reverse the bit. That means, cancel the deployment of that service in this location. After doing that, re-examine the chromosome, if it is still exceed the limitation then repeat this process until there is no redundant services or it satisfies the constraint.

This algorithm will try its best to reduce the cost regardless of other factors. Even though the modified chromosome may end up with higher network latency. Worth noting that this algorithm may not provide a strictly correct solution that satisfied the cost constraint, partially because the randomly chosen of canceling deployment. On the other hand, it may also indicate that the predefine upper boundary of cost limitation is too low.

4 Experiment Design

The purpose of the experiment is comparison between Alternate Location Allocation (ALA) 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 ALA, 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 network latency dataset based on the user centers and candidate locations that we chose previously. 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 level 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.

In order to compare the solution set from NSGA-II and a single solution from ALA, we employ non-dominated Pareto front [20] and Dominance Relation [12]. 40 results (from 40 runs) under each cost constraint are presented in the Section 5.2. 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 ALA.

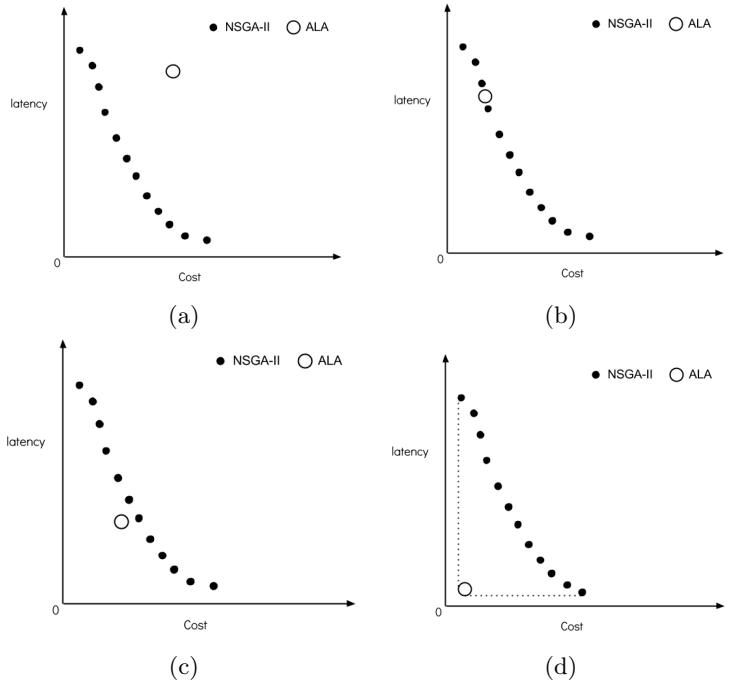


Fig. 3: Four cases of the relation between a ALA solution and a NSGA-II solution set of non-dominated solutions

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 ALA solution is dominated by some solutions in NSGA-II solution set. In this case, we can say that NSGA-II outperforms ALA since the inclusion of the ALA solution does not improve the quality of the NSGA-II solution set. On the other hand, we can say that ALA outperforms NSGA-II in Figure 3 (d) Since the ALA solution dominates all solutions in the NSGA-II solution set. In Figure 3 (b), the ALA solution is non-dominated solution in the NSGA-II solution set. While the inclusion of the ALA solution somewhat improves the quality of the NSGA-II solution set, we cannot say that ALA outperforms NSGA-II since the ALA solution dominates no NSGA-II solutions. We may intuitively say that NSGA-II outperforms ALA in the case of Figure 3 (b).

It is difficult to say which is better between ALA and NSGA-II in 3(c) where ALA solution dominates some NSGA-II solutions. If we use performance measures that are strongly related to the convergence of solutions, ALA 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 ALA.

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

Table 1: NSGA-II Parameters

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

4.1 ALA

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 [18] methods, which build solutions from scratch. An important greedy improvement heuristic is the Alternate Location-Allocation method which builds on an existing solution. 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. ALA builds upon ADD and DROP methods. In order to solve multi-objective problem, ALA first uses Integer programming to solve the problem in terms of cost.

$$\begin{aligned} &\text{Minimize } \sum_{s\in S}\sum_{j\in J} C_{sj} \times A_{sj} \\ &\text{Subject to} \\ &\qquad \sum_{s\in S} A_{sj} \geq 1 \\ &\qquad \sum_{s\in S}\sum_{j\in J} C_{sj} \times A_{sj} \leq CostLimitation \end{aligned} \tag{5}$$

Once a minimum cost solution is found, 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.

5 Experimental results

5.1 Efficiency comparison

Table 2: Efficiency Test

Table 3: Sufficient condition

Matrix Size	GA time(s)	ALA time(s)
3×3	1.7	0.002
5×5	2.1	0.007
10×10	3.775	0.11
15×15	6.355	0.64

Table 4: Good condition

Matrix Size	GA time(s)	ALA time(s)
3×3	1.71	0.003
5×5	2.07	0.01
10×10	3.6	0.11
15×15	6.26	0.6

Table 5: Poor condition

Matrix Size	GA time(s)	ALA time(s)
3×3	1.89	0.004
5×5	2.3	0.01
10×10	4.08	0.11
15×15	8.67	0.57

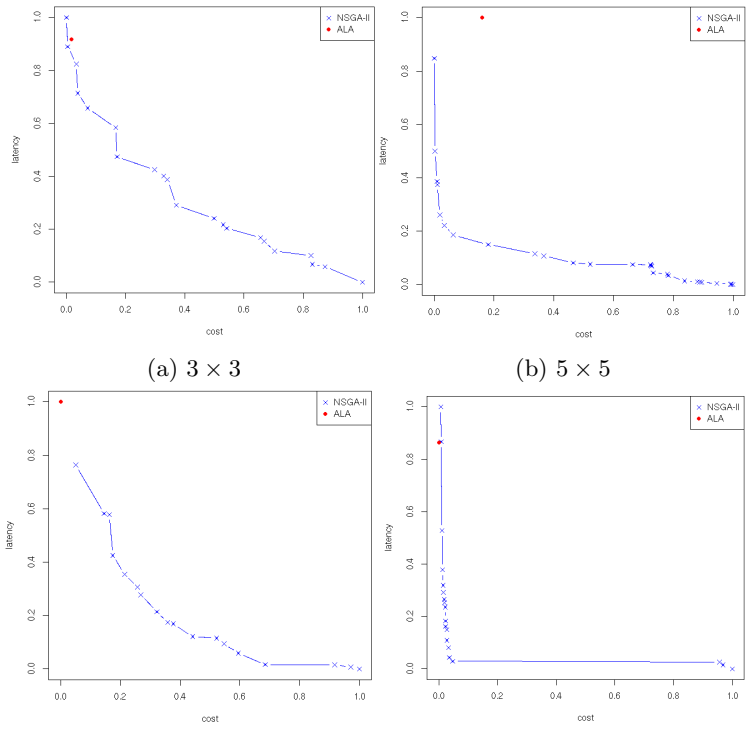
Table 6: Minimum condition

Matrix Size	GA time(s)	ALA time(s)
3×3	1.88	0.003
5×5	2.67	0.01
10×10	6.06	0.11
15×15	15.4	0.6

As the experimental results show, in terms of the average run time, ALA is clearly better than NSGA-II. There are two main reasons: firstly, NSGA-II's run time is largely depend on the generation setting. 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.

5.2 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 ALA. As we discussed in the previous section, the result of ALA is dominated by NSGA-II's solution set. Although, ALA solution for 3×3 matrix is quite close to the Pareto front. 4d shows that solution of ALA is on the Pareto front. It does not improve the quality of the solution set. Therefore, we considered NSGA-II outperforms ALA in this case. 4c is a special case, NSGA-II solution set does not cover the possible solution range. That is, we predefined 50 generation for NSGA-II is not enough for NSGA-II to find a solution set that cover the Pareto front. The solution from ALA is non-dominated solution in the NSGA-II solution set. While the inclusion of the ALA solution somewhat improves the quality of the NSGA-II solution set,

we cannot say that ALA outperforms NSGA-II since the ALA solution dominates no NSGA-II solutions.

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

6 Conclusion

In this paper, we presented a modified NSGA-II for 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 multiobjective 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 multiobjective 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. 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)
10. 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)

11. 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)
12. 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)
13. 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)
14. Johansson, J.M.: On the impact of network latency on distributed systems design. Inf. Technol. and Management 1(3), 183–194 (2000)
15. 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)
16. 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)
17. Ran, S.: A model for web services discovery with qos. SIGecom Exch. 4(1), 1–10 (2003)
18. Sun, Y., Koehler, G.J.: A location model for a web service intermediary. Decis. Support Syst. 42(1), 221–236 (2006)
19. 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)
20. 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)
21. 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)
22. 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)
23. 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)