Seeding-Based Multi-Objective Evolutionary Algorithms for Multi-Cloud Composite Applications Deployment

Tao Shi, Hui Ma and Gang Chen
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email: {tao.shi, hui.ma, aaron.chen}@ecs.vuw.ac.nz

***Abstract*—There are an increasing number of enterprises deploying their application services to multi-cloud to benefit the advantages brought by cloud computing. The multi-cloud composite applications deployment problem (MCADP) aims to select proper cloud resources from multiple cloud providers at different locations to deploy applications with shared constituent services so as to optimize application performance and deployment cost. Multi-objective evolutionary algorithms (MOEAs) can be utilized to find a set of trade-off solutions for MCADP. During population initialization of MOEAs, seeding strategies can considerably improve the algorithms' performance. For example, the seeding-based MOEAs, AO-Seed and SO-Seed, introduce a pre-optimization phase to search for solutions to be embedded into the initial population of MOEAs. With the extra optimization overhead, however, the two seeding-based MOEAs can only identify one or a limited few solutions to MCADP utilized by MOEAs. To solve MCADP effectively and efficiently, we propose new seeding-based MOEAs in this paper. The approach can construct application-specific seeds according to problem domain knowledge and build a group of diverse and high-quality solutions for the initial population of MOEAs. Extensive experiments have been conducted on a real-world dataset. The results demonstrate that the proposed seeding-based MOEAs outperform SO-Seed and AO-Seed with less computation cost for MCADP.**

***Keywords*-Location, composite application, multi-objective, evolutionary algorithm, seeding strategy, multi-cloud**

I. Introduction

In recent years, cloud computing has become a popular paradigm for flexible and scalable deployment and delivery of enterprise-scale business softwares/applications [1]. Multi-cloud makes it possible to coordinate shared use of heterogeneous cloud resources all over the world and avoid vendor lock-in [2]. Supported by the multi-cloud, enterprise application providers are able to transfer application workload among worldwide cloud data centres freely to increase their global competitiveness and business flexibility [3].

Many application providers such as Salesforce[1] often offer several business applications simultaneously. The relevant business processes are usually abstracted as workflows [4]. These *composite applications* consist of a set of specific functionalities, namely *constituent services*, and different applications often share some common constituent services [5]. For example, ordering, selling, and payroll applications can share the same tax-setting service [6].

The deployment of constituent services is crucial for application providers since it significantly affects the average response time (ART) of composite applications [6]. Minimizing ART of composite applications can lead to high monetary cost of service deployment in multi-cloud. Considering the conflicting nature of ART and cost, we formulate the multi-cloud composite applications deployment problem (MCADP) as a multi-objective problem (MOP) with the aim to jointly optimize ART and cost as two separate objectives. Finding optimal solutions for MCADP is computationally intractable due to the large search spaces of available cloud services available at different data centres, dependencies among services across different applications, and diverse user distributions [6].

Evolutionary algorithms (EAs) have been successfully applied to tackling different application deployment in clouds [7] [8]. In the context of MOPs, multi-objective evolutionary algorithms (MOEAs) can be utilized to search for the *Pareto front*, in which each solution represents a unique trade-off in consideration of all conflicting objectives.

The performance of MOEAs depends heavily on population initialization [9] [10]. It is common for MOEAs to inject some seeding solutions obtained by leveraging the information and knowledge from the problem domain. This is known as the seeding strategy [11]. Various seeding strategies have been widely demonstrated to effectively improve the performance and convergence rate of MOEAs [12] [13]. However, most of the existing seeding-based MOEAs are not suitable for tackling MCADP. To find complex solutions of jointly deploying multiple applications, for example, AO-Seed optimizes the aggregated objectives of MCADP and SO-Seed optimizes each objective of MCADP as seeds. The vital information regarding individual applications, such as user distribution of an application, will be simply ignored. Moreover, the pre-optimization process, i.e. single-objective genetic algorithm (GA) in AO-Seed and SO-Seed, is not efficient as it imposes extra overhead in running time.

To solve MCADP effectively and efficiently, we propose new seeding-based MOEAs inspired by blocks building (BB-Seed) in this paper. Concretely, we have the following

[1] https://www.salesforce.com/

2474-2473/20/$31.00 ©2020 IEEE
DOI 10.1109/SCC49832.2020.00039

research objectives:

(1) To utilize the domain knowledge to reduce computation cost, MCADP can be divided into several related sub-problems, each corresponding to a separate application. Hence, well-designed heuristics can be utilized to generate application-specific seeds efficiently.

(2) To effectively construct Pareto-optimal solutions for MCADP, we can seed MOEAs with a group of high-quality and diverse solutions to cover varied compromises across all conflicting objectives. Because each application only relies on a proportion of constituent services, the application-specific seeds are partial solutions to MCADP. They must be integrated together to form complete solutions as seeds of the initial population of MOEAs.

(3) To evaluate the proposed seeding-based MOEA, we collect the information about virtual machines (VMs) capacity and pricing offered by the global top three cloud providers, i.e. Amazon, Microsoft, and Alibaba. Based on the collected information, extensive experiments have been conducted to deploy 10 sets of online business applications. Our approach is compared to the state-of-the-art seeding-based MOEAs, i.e. AO-Seed and SO-Seed [11], for multi-objective MCADP. The experiment results indicate that BB-Seed significantly outperforms all baseline methods and noticeably reduces the computation time.

II. RELATED WORK

In recent years, cloud computing is becoming a booming paradigm for delivering *business* software/applications. It is critical for application providers to deploy business applications with optimal Quality of Service (QoS) [6] [14]. In our previous work [6], MCADP was introduced with the goal to minimize ART of composite applications subject to a budget constraint.

Selecting the proper cloud resources for deploying *scientific* applications is another commonly studied problem, which is also known as the workflow scheduling problem [13]. Recently, workflow scheduling problem has been formulated as a MOP with multiple conflicting objectives. For example, to optimize both makespan and cost, Zhu et al. proposed a new MOEA with problem-specific encoding scheme, genetic operators, and population initialization to produce schedules with state-of-the-art performance and high stability [13]. Particularly, two heuristics are used for seeding the initial population of MOEA to accelerate the search procedure.

Many seeding strategies have also been widely adopted to enhance the performance of EAs in the literature [15]. For example, Schmidt et al. investigated a seeding strategy, i.e. building block mutation, for genetic programming (GP) to reuse individual pieces of an expert model on randomly generated symbolic regression problems [16]. In [17], the permutation gene-pool optimal mixing evolutionary algorithm (GOMEA) is proposed by incorporating a problem-specific heuristic to seed the initial population for the flowshop scheduling problem. In Search-Based Software Engineering (SBSE) community, seeding strategies were proposed for MOEAs on software testing [12] and service composition [11]. However, these approaches usually generate seeds in one single step. For example, AO-Seed and SO-Seed in [11] apply a single-objective GA to search for seeds in the form of complete solutions to the service composition problem.

In summary, to build high-quality seeds that can substantially improve the performance of MOEAs, the original problems often need to be substantially simplified to make direct seeding possible. However, no efforts have ever been made to generate and integrate the sub-problem-based seeds for cloud-based application deployment problems.

III. PROBLEM DESCRIPTION

An application provider provides a set of composite applications $\mathcal{W} = \{W_1, ..., W_a, ..., W_{|\mathcal{W}|}\}$ that jointly use a collection of constituent services $\mathcal{S} = \{s_1, ..., s_l, ..., s_{|\mathcal{S}|}\}$. That is, one constituent service s_l might be shared by one or more composite applications. We represent each application W_a as a Directed Acyclic Graph (DAG), denoted by $G =< S_a, E_a >$. $S_a \subseteq \mathcal{S}$ is a subset of constituent services, each represented by a vertex in G. E_a is a set of directed edges where $e_{mn} \in E_a$ connects s_m with s_n. Based on e_{mn}, s_m is called the parent service of s_n and s_n is the child service of s_m. In application W_a, service $s_l \in S_a$ may have more than one parent services and child services, we denote the set of the parent services of s_l as $Pre(s_l)$, and the set of the child services of s_l as $Succ(s_l)$. A service that has no parent services is called the starting service of W_a (denoted as $start_a$), and a service without child services is called the ending service of W_a (denoted as end_a). If an application has multiple starting or ending services, a dummy starting or ending service will be added. An example with three composite applications is shown in Figure 1. In the example, there are totally 11 constituent services. Service s_1 is used by applications W_1 and W_3, while s_2 is shared by all three applications.

We consider a set of user centres $\mathcal{U} = \{U_1, ..., U_k, ..., U_{|\mathcal{U}|}\}$, which represent the centres of global user groups. The request rate γ_{ka} denotes the request frequency from a user centre U_k to W_a. In this regard, the total workload of W_a can be determined as: $\theta_a = \sum_{k=1}^{|\mathcal{U}|} \gamma_{ka}$.

Constituent services required by all composite applications can be deployed onto a set of VMs in the multi-cloud supported by a third-party, the broker [18]. We consider a set of VM types $\mathcal{V} = \{V_1, ..., V_i, ..., V_{|\mathcal{V}|}\}$ and a collection of data centre locations $\mathcal{A} = \{A_1, ..., A_j, ..., A_{|\mathcal{A}|}\}$. If the VM type $V_i \in \mathcal{V}$ is available in the data centre location $A_j \in \mathcal{A}$, we use v_{ij} to denote this VM type and c_{ij} to denote its cost at hourly rate.

Based on the related study in [6] and industry practice [19], we assume each constituent service must have exclusive use of one VM instance. Each service maintains

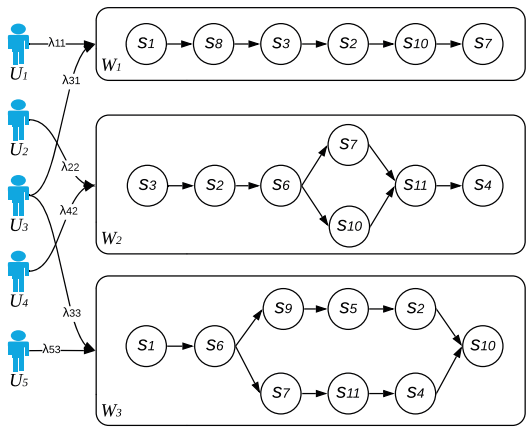


Figure 1. Three example composite applications for deployment.

an independent queue for pending requests arriving from all user centres or its parent services in all deployed applications. Refer to [6], the average resource consumption per request over a long sequence of requests for the same service is considered highly stable. We hence model model the operation of each individual service as an $M/M/1$ queue [20]. Because an application service can be shared by multiple applications, the aggregated workload of the service s_l can be calculated by:

$$\lambda_l = \sum_{a=1}^{|\mathcal{W}|} \theta_a \qquad \text{s.t.} \quad s_l \in S_a. \tag{1}$$

Once s_l is deployed, the capacity of s_l, i.e., μ_l, measures the request processing speed of the deployed VM instance. According to Little's Law [21], the average request processing time of service s_l is:

$$pt_l = \frac{1}{\mu_l - \lambda_l}, \tag{2}$$

where $\mu_l > \lambda_l$, that guarantees that the workload of the VM instance will not exceed its capacity. In the remaining of this paper, we use *capacity feasible* VM types to refer the VM types on which the services' capacities are greater than their workloads.

We define the average response time of an application ART_a as the total time on average required for W_a to process its user requests. Since a user centre $U_k \in \mathcal{U}$ and a data centre $A_j \in \mathcal{A}$ can span large geographic areas, the network latency must be taken into account when computing ART_a. The *average* network latency ut_a between user centres and the application W_a can be calculated by:

$$ut_a = \frac{\sum_{k=1}^{|\mathcal{U}|} \gamma_{ak}(It_{ak} + Ot_{ak})}{\theta_a}, \tag{3}$$

where It_{ak} and Ot_{ak} are the network latency between a user centre U_k and the data centre where $start_a$ is deployed and the network latency between the data centre where end_a is deployed and the user centre U_k, respectively. Besides, we denote the network latency between service s_m and its child service s_n as st_{mn}. The aforementioned two types of network latency, i.e. ut_t and st_{mn}, are usually determined by several non-changeable factors in the communication network, such as the physical distance among user centres and services [6]. Accordingly, the ART_a can be determined as follows:

$$ART_a = ut_a + MS(W_a), \tag{4}$$

where function $MS(W_a)$ calculates the response time within the workflow W_a. The two functions EST (Earliest Start Time) and FT (Finish Time) for each service can be defined to calculate the application response time. The computing process begins with the starting service, and the $MS(W_a)$ is the finish time of the ending service:

$$\begin{aligned} &EST(start_a) = 0, \\ &FT(s_l) = EST(s_l) + pt_l, \\ &EST(s_l) = \max_{s_m \in Pre(s_l)} \{FT(s_m) + st_{ml}\}, \\ &MS(W_a) = FT(end_a). \end{aligned} \tag{5}$$

We use a one-dimensional vector $X = \{v_{ij}^l\}_{l=1}^{|\mathcal{S}|}$ to represent the applications deployment solution, where the value of v_{ij}^l indicates that a constituent service s_l is deployed to one instance of VM type V_i at the data centre location A_j. Correspondingly, the deployment cost of s_l, i.e. DC_l, is c_{ij}.

The total deployment cost (TDC) and total response time (TRT) of all the applications can be calculated as follows:

$$TDC(X) = \sum_{l=1}^{|\mathcal{S}|} DC_l, \tag{6}$$

$$TRT(X) = \frac{\sum_{a=1}^{|\mathcal{W}|} \theta_a \cdot ART_a}{\sum_{a=1}^{|\mathcal{W}|} \theta_a}. \tag{7}$$

Therefore, MCADP has the goal to minimize two objectives:

$$\min \quad TDC(X), \tag{8}$$

$$\min \quad TRT(X). \tag{9}$$

That is, the aim of MCADP is to minimize TDC and TRT of all the given applications simultaneously.

IV. The Proposed Seeding-based MOEAs: BB-Seed

Seeding-based MOEAs operate similarly to the classic MOEAs, but include some solutions, obtained through explicitly using problem knowledge, into the initial population of MOEAs to make their evolutionary search effective [11]. In this section, we will introduce the representation,

the reproduction operators, and the seeding strategy. The discussion of mating and environmental selection is omitted in this paper, because they are usually algorithm dependent (e.g., dominance-based comparison in NSGA-II [22]).

A. Chromosome Representation

We need to encode MCADP solutions into chromosomes so that we can use MOEAs to evolve solutions. As we mentioned in Section III, we need to choose a data centre location and a VM type at the location to deploy each constituent service. Therefore, we design a chromosome with two strings, i.e. *VM location* string and *VM type* string, to encode deployment solutions. The length of each string is the total number of constituent services among all the given applications.

Constituent services:	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}
VM location index:	4	5	3	6	2	1	3	2	2	4	2
VM type index:	1	2	1	1	4	1	2	2	2	2	1

Figure 2. An example chromosome for encoding deployment solutions.

Figure 2 gives the encoding of a possible deployment for the 11 constituent services used by the three composite applications in Figure 1, according to which service s_1 is deployed to a VM instance of type V_1 at location A_4, and service s_2 is deployed to the VM instance of V_2 at A_5, and so on.

B. Reproduction Operators

The crossover and mutation operators in this work are designed for the two-string chromosomes. We apply the widely used uniform crossover. For each constituent service, as shown in Figure 3, we pick up both *VM location* and *VM type* from two parents with the equal probability to generate offsprings subject to a crossover rate.

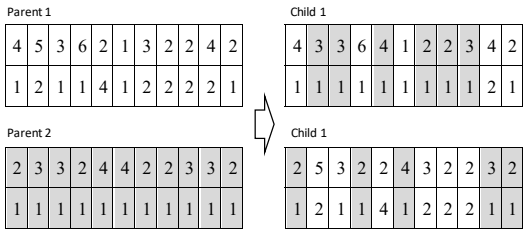


Figure 3. An example of crossover operator.

Our mutation operator mutates the *VM location* and *VM type* of constituent services separately. Subject to a mutation rate, some constituent services are randomly selected. For each selected service, the location is randomly changed to a different data centre. Then the VM type is randomly updated by using a different VM type that is available in the new data centre and can meet the capacity requirement, as shown in Figure 4.

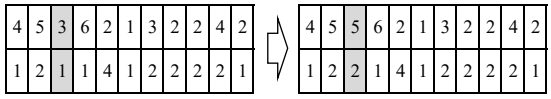


Figure 4. An example of mutation operator.

C. Block-building Seeding Strategy

For MCADP, we consider individual composite applications as sub-problems. Note that solving any sub-problem in isolation, i.e. deploying one specific composite application, may ignore the relationship with other sub-problems through shared constituent services, as shown in Figure 1. However, well-designed heuristics can be employed to harness the information from each sub-problem, e.g., the user distribution of an application, in order to generate high-quality seeds.

1) Method for seeds generation: We develop new methods to generate seeds for each composite application with the aim of solely optimizing each individual objective of MCADP. These seeds, with strong bias toward each single objective, may help to identify emergent and good solutions [11] [13].

There are two objectives in MCADP, therefore, two sub-problem-based seeds, i.e. the *cheapest solution* and the *fastest solution*, can be generated for *each* composite application as shown in Figure 5. On the one hand, the cheapest solution of an application with the minimal TDC can be easily obtained by deploying all its constituent services to the cheapest VMs that meet the capacity requirement in multi-cloud, e.g., the VMs in Amazon Web Service (AWS) US East datacentre. On the other hand, the fastest solution with the lowest ART can be easily obtained as follows.

According to the problem description in Section III, ART of an application includes the response time within the application and the average network latency between user centres and the application. To find the fastest solution of an application, we first select a data centre location where the average network latency is minimal, e.g., AWS London datacentre. Then we deploy all constituent services of the application to the VMs with the highest capacity available at this location to minimize the response time within the application.

The total number of sub-problem-based seeds to be obtained from the process described above is $2 \cdot |\mathcal{W}|$. In Figure 5, 6 seeds are generated for the 3 composite applications (i.e. sub-problems) in Figure 1. Note that all these seeds are *partial* solutions to MCADP (with blanks shown in Figure 5), because only a subset of constituent service set $\mathcal{S}$ have been deployed according to these seeds. That is, we must integrate these seeds to form a group of *complete* deployment solutions, which can then be included into the initial population of MOEAs.

2) Methods for seeds integration: We integrate complete deployment solutions in two ways, and the integration unit

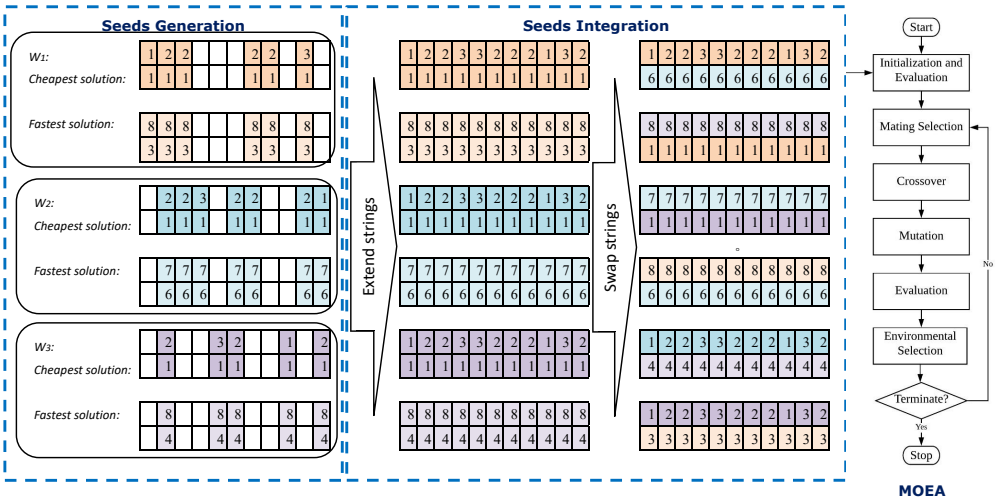


Figure 5. The proposed seeding-based MOEA.

is a string of chromosomes, i.e., both the VM location string and VM type string.

Firstly, we *extend* the sub-problem-based seeds to complete solutions. For the VM location string, specifically, the blank positions in a seed are filled directly according to the current location of deployed constituent services. That is, all constituent services are deployed at the same datacentre. Hence, the time required for inter-service communication is negligible. For the VM type string, we fill all missing places in the seed by using either the cheapest capacity-feasible VMs or the most powerful but expensive VMs, depending on whether the optimization focus is either on cost or time, respectively. As shown in Figure 5, all the 6 seeds were *extended* to form complete solutions.

Afterwards, the VM location strings and VM type strings are *swapped* among these complete solutions randomly to form new complete solutions illustrated in Figure 5. Both the *extended* and *swapped* complete solutions are added into the initial population of MOEAs. Theoretically, our seeding method can generate up to $(2 \cdot |\mathcal{W}|)^2$ distinctive complete solutions to seed the initial population of MOEAs.

Note that these complete solutions must undergo modification with respect to *location-availability* and *capacity-feasibility* to ensure feasible deployment for MCADP.

V. Experimental Studies

In this section, we investigate the performance of the proposed seeding-based MOEAs on MCADP.

A. Datasets

According to latest worldwide Infrastructure-as-a-Service (IaaS) public cloud services market share [23], we collect the real VM type description and pricing in December 2019 from three leading cloud providers, i.e. AWS[2], Microsoft Azure[3] and Alibaba Elastic Compute Service (ECS)[4]. 24 different VM types (8 from AWS, 8 from Azure, and 8 from Alibaba) have been included in the experiments. We also consider a total of 15 locations for major AWS, Azure and Alibaba data centres. Furthermore, we adopt 82 user centres from 35 countries on 6 continents in Sprint IP Network[5] to simulate the global user community.

In our experiments, we consider 10 business applications corresponding to diverse online activities such as online shopping and travel planning services studied in [5]. Totally 20 different constituent services are included in these applications. We generate 10 sets of composite applications by randomly selecting applications among them. The number of applications (NoA in Table I) in the 10 sets ranges from 3 to 7 with the number of services (NoS in Table I) from 14 to 20. Basically, MCADP is more complex due to the increment on the amounts of applications and services.

To evaluate the network latency between user centers and constituent services, and the network latency between two connected services, we use real-world observation of network latency from Sprint[6] IP backbone network databases.

B. Simulation Settings

Refer to [6], we apply Facebook subscribers statistics to simulate the distribution of application requests from user centres, i.e. γ_{ka}. By June 2017, there are approximately 1.33 billion Facebook subscribers from all of 35 countries

[2]https://aws.amazon.com/ec2/instance-types/
[3]https://azure.microsoft.com/en-us/services/virtual-machines/
[4]https://www.alibabacloud.com/product/ecs.
[5]https://www.sprint.net/network_maps.php
[6]https://www.sprint.net/performance/

Table I
THE MEAN AND STANDARD DEVIATION OF HV AND IGD FOR THE FINAL SOLUTION SET OVER 30 RUNS (THE BEST IS HIGHLIGHTED)

Metrics	NoA	NoS	**BB-Seed**	**AO-Seed** [11]	**SO-Seed** [11]	**NONE**
HV	3	14	**0.9971±0.0001**	0.9853±0.0063	0.9942±0.0001	0.989±0.0069
	3	16	0.9974±0.0000	0.9951±0.0052	**0.9975±0.0001**	0.9973±0.0003
	4	16	**0.998±0.0000**	0.9726±0.0001	0.9846±0.0001	0.9839±0.0013
	4	17	**0.9958±0.0003**	0.9814±0.0000	0.9811±0.0001	0.9777±0.0059
	5	17	**0.9977±0.0001**	0.9894±0.0089	0.9948±0.0000	0.9897±0.0092
	5	19	**0.9966±0.0001**	0.9539±0.0002	0.9578±0.0002	0.9561±0.0037
	6	17	**0.9976±0.0001**	0.9890±0.0044	0.9950±0.0000	0.9942±0.0013
	6	20	**0.9960±0.0004**	0.9728±0.0102	0.9874±0.0000	0.9832±0.0069
	7	18	**0.9969±0.0003**	0.9786±0.0101	0.9858±0.0000	0.9851±0.0016
	7	19	0.9975±0.0001	0.9977±0.0001	**0.9979±0.0001**	0.9952±0.0051
IGD	3	14	**0.0033±0.0002**	0.1024±0.0773	0.0231±0.0040	0.0719±0.0623
	3	16	**0.0031±0.0002**	0.0637±0.0406	0.0037±0.0007	0.0193±0.0235
	4	16	**0.0033±0.0002**	0.0426±0.0102	0.0151±0.0011	0.0406±0.0106
	4	17	**0.0038±0.0003**	0.0384±0.0141	0.0155±0.0006	0.0461±0.0170
	5	17	**0.0034±0.0003**	0.1212±0.0627	0.0168±0.0068	0.0724±0.0658
	5	19	**0.0041±0.0004**	0.0544±0.0252	0.0896±0.0076	0.1288±0.0221
	6	17	**0.0033±0.0002**	0.1322±0.0730	0.0139±0.0018	0.0866±0.0338
	6	20	**0.0035±0.0004**	0.1164±0.0565	0.0513±0.0120	0.1349±0.0565
	7	18	**0.0038±0.0004**	0.1579±0.0655	0.0524±0.0128	0.1100±0.0742
	7	19	**0.0044±0.0003**	0.0398±0.0190	0.0044±0.0010	0.0684±0.0272

considered in our simulation[7]. We consider the user scale for composite applications as 1/1000th of Facebook subscribers to simulate the application providers with million users, because we have more applications at this scale in the multi-cloud market, e.g., Xero[8]. For each composite application, we select 10 out of all 82 user centres randomly, and each user from the selected user centres makes 25 requests on average daily following [24]. Accordingly, the application request rate spans from 12 to 163 requests per second (that is, approximately 1.0-14.1 millions of requests daily).

Following [12], we apply two commonly used quality indicators, i.e. hypervolume (HV) [25] and inverted generational distance (IGD) [26] to evaluate different seeding-based MOEAs. The two metrics can meet different practical requirements and present us with a comprehensive understanding of the effectiveness of different algorithms. To compare the results, we consider the mean and standard deviation of each metric after running each experiment 30 times. We implemented MOEAs based on the jMetalPy framework [27], and all the experiments[9] are performed on computers with identical hardware configuration: Intel Core i7-8700 CPU (3.2 GHz and 16 GB of RAM).

C. The Performance of BB-Seed

We evaluate our newly proposed seeding-based MOEAs, BB-Seed, in comparison to the recently proposed seeding-based MOEAs in [11]:

AO-Seed: The approach assumes equal-weighted sum aggregation of all the objectives to obtain one seed.

SO-Seed: The approach focuses on single objective optimization to find the best solutions for each objective as seeds.

[7]https://www.internetworldstats.com/facebook.htm

[8]https://www.xero.com/nz/

[9]The source code can be accessed at: https://github.com/qingdaost/seeding-MCADP

The MOEAs without seeding (NONE for convenience) is also used as a baseline approach.

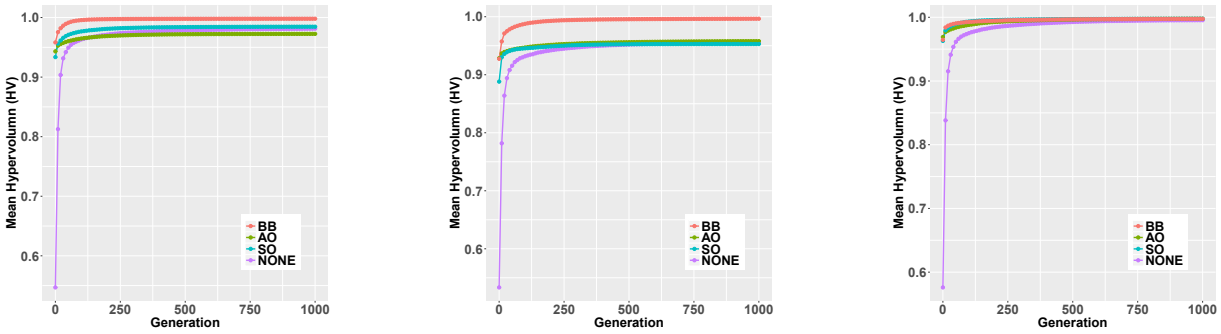
We have applied the competing seeding-based MOEAs above to several popular MOEAs, including NSGA-II [22], SPEA2 [28], and NSGA-III [29]. Regardless of the algorithms used, the performance improvement of BB-Seed over AO-Seed and SO-Seed is similar. Due to the space limitation, we only present the experimental results associated with NSGA-II.

Refer to [11], we set the parameters of NSGA-II as follows: the population size is 100, the crossover rate and mutation rate are 0.9 and 0.1, respectively, and the number of seeds is 50% of the initial population size, i.e. 50. Besides, we increase the maximum number of generations to 1000 to ensure all competing approaches can converge on our problem.

1) Overall Performance: As we can see from Table I, when comparing the mean HV values, which capture the convergence and diversity of the final solution set, BB-Seed outperforms AO-Seed and SO-Seed for 8 out of 10 cases. For the other 2 cases, BB-Seed can achieve compatible performance as the existing approaches.

The improvements achieved by BB-Seed are more obvious when focusing on the mean IGD values, which is another frequently used performance measure for MOPs with key focus on the diversity of solutions in the evolved Pareto front [26]. We note that BB-Seed achieves the best performance across all the cases. Generally, the performance advantage is more noticeable on complex problems with a high number of applications and services.

We also find that BB-Seed has the smallest standard deviation of both HV and IGD among all baseline approaches in most cases. In the cases where AO-Seed or SO-Seed has the smallest standard deviation of HV values, e.g., $NoA = 4, NoS = 17$ and $NoA = 5, NoS = 17$, the mean HV values of BB-Seed are significantly higher, confirming that BB-Seed can consistently find good trade-



(a) $NoA = 4, NoS = 17$ (b) $NoA = 6, NoS = 17$ (c) $NoA = 7, NoS = 19$

Figure 6. The evolution of mean HV with respect to the number of generations on selected 3 sets of composite applications

off deployment solutions.

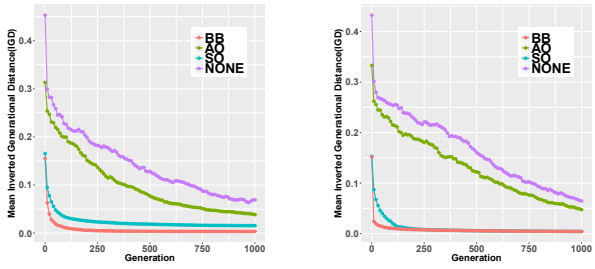
For both quality indicators, the performance differences between BB-Seed and all baseline approaches are all verified through statistical test (Wilcoxon Rank-Sum test) with significance level of 0.05.

2) Evolution Process: We analyse the experimental results by observing how the HV and IGD values change during the evolution process. For each 10 generations, the mean HV and IGD of the populations over 30 runs are illustrated in Figure 6 and Figure 7 respectively. Due to space constraint, we only present the representative scenarios.

In most cases, BB-Seed can achieve better HV values than AO-Seed, SO-Seed, and NONE in the initial population of NSGA-II, e.g., when deploying 4 composite applications with 17 constituent services as shown in Figure 6(a). With the relatively high initial HV value, BB-Seed is able to improve its mean HV very quickly. This demonstrates that BB-Seed can generate more diverse trade-off solutions between the two conflicting objectives. The relatively poor initial population from SO-Seed impacts the algorithm performance, as shown in Figure 6(b). In this case, we also find that the randomly initialized NONE outperforms AO-Seed by rendering enough generations (1000 generations in our experiments). Occasionally, BB-Seed has similar performance with AO-Seed and SO-Seed for the initial and final population of NSGA-II, e.g., when we deploy 7 composite applications with 19 constituent services shown in Figure 6(c).

In term of mean IGD, BB-Seed also has the best initial population, and retain that advantage during the evolution as shown in Figure 7(a). From Figure 7(b), we can see the special case where the IGD value of the initial population based on BB-Seed are similar to that obtained by SO-Seed. As the HV evolution in this case, the situation continues till the maximal number generation.

3) Further Analysis: The computation cost of AO-Seed and SO-Seed depends on the number of fitness evaluations required by the GA-based seeding phase. Instead, by directly utilising problem-specific knowledge, including both VMs regional pricing differences and application requests



(a) $NoA = 4, NoS = 17$ (b) $NoA = 7, NoS = 19$

Figure 7. The evolution of mean IGD with respect to the number of generations on selected 2 sets of composite applications

distribution, BB-Seed can achieve much lower computation cost than AO-seed and SO-seed. The greedy-based seeding heuristics require only a limited number of solution modifications. In our experiments, the computation time of BB-Seed is less than 50ms, which is approximately 1% of AO-Seed and SO-Seed for population initialization.

Overall, BB-Seed is more effective and efficient on MCADP in terms of both HV and IGD by generating well-diversified seeds, in comparison to both AO-Seed and SO-Seed.

VI. Conclusion

In this paper, we propose a novel seeding-based MOEA to tackle MCADP with two conflicting objectives, i.e. minimizing ART and cost. The extensive experimental results confirm that the proposed seeding-based MOEA, BB-Seed, outperforms AO-Seed and SO-Seed in the majority of cases. In particular, BB-Seed significantly improves the HV and IGD on some popular MOEAs by seeding their initial population with a group of diverse and high-quality deployment solutions with less running time overhead.

In the future, we will consider many potentially conflicting objectives of MCADP, e.g., the reputation, scalability and usability of the resource in multi-cloud. The seeding-

based MOEA is expected to demonstrate more importance when addressing these many-objective problems.

REFERENCES

[1] R. Xu, Y. Wang, W. Huang, D. Yuan, Y. Xie, and Y. Yang, "Near-optimal dynamic priority scheduling strategy for instance-intensive business workflows in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 18, p. e4167, 2017.

[2] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2010, pp. 13–31.

[3] N. Grozev and R. Buyya, "Inter-cloud architectures and application brokering: taxonomy and survey," *Software: Practice and Experience*, vol. 44, no. 3, pp. 369–390, 2014.

[4] X. Zhu, S. vanden Broucke, G. Zhu, J. Vanthienen, and B. Baesens, "Enabling flexible location-aware business process modeling and execution," *Decision Support Systems*, vol. 83, pp. 1–9, 2016.

[5] K.-C. Huang and B.-J. Shen, "Service deployment strategies for efficient execution of composite saas applications on cloud platform," *Journal of Systems and Software*, vol. 107, pp. 127–141, 2015.

[6] T. Shi, H. Ma, G. Chen, and S. Hartmann, "Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1954–1969, 2020.

[7] T. Shi, H. Ma, and G. Chen, "A genetic-based approach to location-aware cloud service brokering in multi-cloud environment," in *2019 IEEE International Conference on Services Computing (SCC)*. IEEE, 2019, pp. 146–153.

[8] T. Shi, G. Chen, and H. Ma, "Multi-objective container consolidation in cloud data centers," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 783–795.

[9] Y. Jin, *Knowledge incorporation in evolutionary computation*. Springer, 2013, vol. 167.

[10] S. Srinivasan and S. Ramakrishnan, "Evolutionary multi objective optimization for rule mining: a review," *Artificial Intelligence Review*, vol. 36, no. 3, p. 205, 2011.

[11] T. Chen, M. Li, and X. Yao, "On the effects of seeding strategies: a case for search-based multi-objective service composition," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 1419–1426.

[12] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, A. Egyed, and E. Alba, "Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines," in *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2014, pp. 387–396.

[13] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Transactions on parallel and distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2015.

[14] T. Shi, H. Ma, and G. Chen, "A seeding-based GA for location-aware workflow deployment in multi-cloud environment," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 3364–3371.

[15] D. R. White, A. Arcuri, and J. A. Clark, "Evolutionary improvement of programs," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 4, pp. 515–538, 2011.

[16] M. D. Schmidt and H. Lipson, "Incorporating expert knowledge in evolutionary search: a study of seeding methods," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 2009, pp. 1091–1098.

[17] G. Aalvanger, N. H. Luong, P. A. Bosman, and D. Thierens, "Heuristics in permutation gomea for solving the permutation flowshop scheduling problem," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 146–157.

[18] A. N. Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, pp. 7–53, 2014.

[19] Skytap. (2019) Help and documentation. [Online]. Available: https://help.skytap.com/Accessing_VMs_with_Published_Services.html

[20] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, 2014.

[21] J. D. Little and S. C. Graves, "Little's law," in *Building intuition*. Springer, 2008, pp. 81–100.

[22] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[23] C. Stamford. (2019) Gartner forecasts worldwide public cloud revenue to grow 17% in 2020. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020

[24] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. IEEE, 2010, pp. 446–452.

[25] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.

[26] C. A. C. Coello and M. R. Sierra, "A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm," in *Mexican International Conference on Artificial Intelligence*. Springer, 2004, pp. 688–697.

[27] A. Benitez-Hidalgo, A. J. Nebro, J. Garcia-Nieto, I. Oregi, and J. Del Ser, "jmetalpy: a python framework for multi-objective optimization with metaheuristics," *Swarm and Evolutionary Computation*, vol. 51, p. 100598, 2019.

[28] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," *TIK-report*, vol. 103, 2001.

[29] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints," *IEEE transactions on evolutionary computation*, vol. 18, no. 4, pp. 577–601, 2013.