



# Multi-objective microservice deployment optimization via a knowledge-driven evolutionary algorithm

Wubin Ma<sup>1</sup> · Rui Wang<sup>1</sup> · Yuanlin Gu<sup>2</sup> · Qinggang Meng<sup>2</sup> · Hongbin Huang<sup>1</sup> · Su Deng<sup>1</sup> · Yahui Wu<sup>1</sup>

Received: 8 March 2020 / Accepted: 13 July 2020 / Published online: 30 August 2020  
© The Author(s) 2020

## Abstract

For the deployment and startup of microservice instances in different resource centres, we propose an optimization problem model based on the evolutionary multi-objective theory. The objective functions of the model consider the computation and storage resource utilization rate, load balancing rate, and actual microservice usage rate in resource service centres. The constraints of the model are the completeness of service, total amount of storage resources, and total number of microservices. In this study, a knowledge-driven evolutionary algorithm (named MGR-NSGA-III) is proposed to solve the problem model and seek the optimal deployment and startup strategy of microservice instances in different resource centres. The proposed model and solution have been evaluated via real data experiments. The results show that our approach is better than the traditional microservice instance deployment and startup strategy. The average computation rate, storage idle rate, and actual microservice idle rate were 13.21%, 5.2%, and 16.67% lower than those in NSGA-III, respectively. After 50, 100, and 150 evolutionary generations in several operations, the population members in NGR-NSGA-III dominated the population members in NSGA-III 6,270, 3,581, and 7,978 times in average, respectively, which means that NGR-NSGA-III can converge to the optimal solution much quicker than NSGA-III.

**Keywords** Multi-objective optimization · NSGA-III · MGR-NSGA-III · Microservice

## Introduction

Benefiting from the revolution of SOA (service-oriented architecture) [13] and cloud technology, microservices techniques have been developed and applied in many applications [26]. Compared with microservices, traditional single applications have many limitations, especially when there are a large number of users in different regions and with different business needs. A single application means that a war package contains all the functions of the project. The scalability, fault tolerance, stability, and invulnerability of single applications are clearly insufficient to adapt to the rapid development of mobile internet, which is widely used for large-scale distributed concurrent users. Therefore, microservice-based applications research has attracted wide attention [14, 26]. Microservices decompose traditional single-application tasks into independent microservices horizontally or vertically to meet different business and functional requirements. Each decomposed service can run multiple instances individually.

Microservices are flexible, scalable, and highly available. (a) Flexibility means that each service instance can

---

✉ Rui Wang  
ruiwangnudt@gmail.com  
Wubin Ma  
mawubin417@gmail.com  
Yuanlin Gu  
y.gu@lboro.ac.uk  
Qinggang Meng  
q.meng@lboro.ac.uk  
Hongbin Huang  
hb\_huang@nudt.edu.cn  
Su Deng  
su\_deng@nudt.edu.cn  
Yahui Wu  
yahui\_wu@nudt.edu.cn

<sup>1</sup> Science and Technology on Information System Engineering Laboratory, National University of Defense Technology, Changsha 410073, China

<sup>2</sup> Computer Science Department, Loughborough University, Loughborough 4LF 3TU, UK

be flexibly deployed in multiple servers or cloud resource centres on the condition of sufficient network communication bandwidth and computing and storage resources [10]; (b) scalability is the ability to start and shut down instances according to the user's concurrent needs. These operations should be adaptable to a dedicated service module. For example, in e-commerce microservices, the concurrent user payment services suddenly increase at a certain time while other services (such as user registration service and ERP service) are unchanged. In that case, the system should only expand the payment service ability rather than the entire service. Therefore, the flexible extension mechanism of microservices can provide users with high concurrent services while saving resources. (c) High availability is another typical feature of the microservices. The resources required for the microservice instances are relatively small, and each microservices can start multiple instances at the same time. It can perform multi-service hot standby operations on certain key services to improve the stability of the entire application. In summary, due to the above characteristics, microservices are an effective method when there is a large number of users and access devices and massive concurrent requests [1, 11] for applications of mobile internet and internet of things.

Many large-scale network application companies or enterprises (e.g., Amazon [27], IBM [6], eBay [24], Twitter [19], Alibaba [21], Tencent [35]) are gradually adopting microservice-based architecture. Microservices can provide powerful and flexible expansion in many applications. However, there are also some problems which need to be solved urgently. One of the most critical issues is the microservice deployment and startup optimization problem. The main problem is that some microservices in multiple locations need to be started to provide services for a large number of users distributed over a wide area. How and where should the instance of these microservices be started in the constraint of hardware resources, service delays, and service reliability? This is the core problem of service centre configuration and schedules. The difficulty of this problem lies in the following: (a) many factors (computing or storage resource usage rate, service efficiency, and load balancing indicators) in various servers and service centres are mutually constrained, and it is difficult to achieve an optimal solution on all targets in all centres. (b) The computational complexity is high, and it is difficult to find the optimal solution in polynomial time using traditional optimization algorithms. (c) The startup and service requests for microservice instances have certain dynamics, and the solution strategy needs to be quickly solved to meet the actual needs.

The contributions of this paper are as follows: (a) a multi-objective optimization model is constructed for microservices deployment and startup strategy. (b) A knowledge-driven evolutionary algorithm (Multi-Generations Reference

NSGA-III, MGR-NSGA-III) is proposed to choose the elite solutions in each evolutionary generation by considering the lineages of the population members. (c) A number of multi-objective algorithms are compared to solve the microservices deployment and startup problem in real datasets.

According to the authors' knowledge, this is the first study to model the microservices deployment and startup problem based on multi-objective optimization on the server side. The model is evaluated by applying a microservice system to typical datasets in New York and Tokyo. The simulation experiment shows that the solution is feasible and supports the deployment and startup of microservices.

The remainder of this paper is organized as follows: Section "Related works" provides a brief review of the multi-objective optimization model of microservices deployment and startup strategy. Section "Problem definition and multi-objective model" proposes a multi-objective optimization model of problem. MGR-NSGA-III algorithm is proposed in Section "Multi-generations reference NSGA-III". Section "Experiment analysis" presents a comparative analysis and summary of the experiments.

## Related works

The research of service computing mainly focuses on the traditional web service composition problem [3, 15]. The traditional solution is to provide users with the most appropriate service combination strategy from the perspective of optimizing service quality indicators, such as service performance indicators, energy indicators, and service security indicators [32]. This article provides a good review of service computing; however, it does not discuss the application of the algorithm in service computing, and it lacks experimental results.

In the cloud service environment, the service deployment strategy problem consists of the following aspects. The quality of service (QoS) of composite services plays an important role in the deployment value chain and partner collaboration in the manufacturing field [29]. QoS is an evaluation system which is used to evaluate performance of the services, including comprehensive evaluation of service availability, accessibility, reliability, legality, security, success rate, access volume, response time, etc. The cloud service was adopted to improve the service quality in the intelligent manufacturing field [16]. The synthetic genetic algorithm was used to solve the service deployment and composition problem. For the QoS-based service composition problem, a service composition solution based on the multi-constraint optimal path problem model was proposed [12] by optimizing the correlation between various QoS factors. The results of some experiments show that the solution achieves good performance of QoS in cloud computing. However, when the

number of available services in the service pool is increased, these solutions become very inefficient. As the services manufactured by the cloud are complementary and interact with each other, it is hard to select a service composition group. The swarm differential evolution algorithm [34] and the hybrid grey wolf group optimization algorithm [4] were introduced into the cloud manufacturing large-scale service deployment problem.

Optimal service deployment and combination strategies have been used to extend the lifespan of the internet of things (IoT). A game and evolutionary theory-based method was proposed [18] to solve the optimal service composition strategy of IoT. The life cycle of IoT services can be extended by seeking a load-balancing optimal service arrangement strategy for multiple devices. In addition, a heuristic strategy was used to schedule user requests and services in IoT applications [22] to achieve a supply and demand equilibrium between users and service providers in heterogeneous environments.

Microservices are applied to provide users with high-quality and high-performance services while minimizing the cost of hosting user services. The quality and performance of microservices are affected by some major factors [17]. These factors include the flexibility of the deployment infrastructure, load balancing, changing service providers, infrastructure reservations, memory size, etc. Low cost means that service providers have high resource usage rates. Under the condition of the FAAS (function as a service) technology architecture concept, this paper [2] proposes an event-driven and scalable service environment to determine which and how service instances need to be started. However, this strategy is based on function calls rather than resource requirements. In this case, the policy is implemented on the server. Therefore, the FAAS needs to be expanded to respond to the user requests. In this way, microservices can provide users with more scalable services. However, a limitation is that the expansion of FAAS is not efficient and transferable. The server needs to be restarted and re-deployed every time the physical resources are expanded. One solution is the composite service instance startup strategy. The server can dynamically start and end different service instances according to the global user requirements. Thus, a combination optimization strategy of the entire server can be obtained without restarting the physical server. In [9], the microservices allocation strategy in the heterogeneous microservices environment was studied. An efficient real-time dynamic microservices scheduling strategy that enables users to schedule their requests after the microservices have been deployed was proposed.

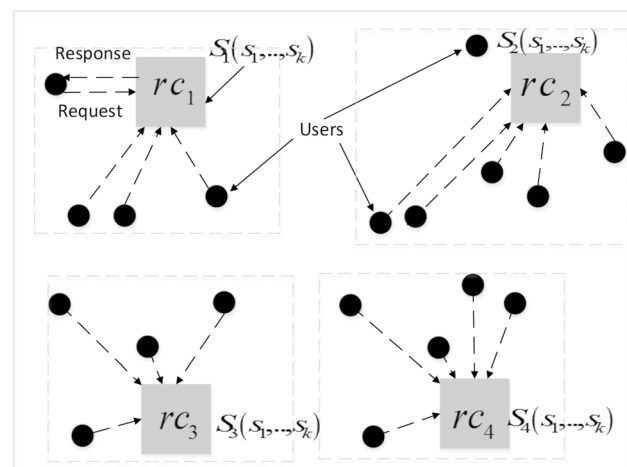
The services/microservices deployment and composition scheduling strategies mentioned above can adapt to different QoS indicators. The advantage of microservices is that it can produce adaptive deployment strategies for different

requirements. However, most of the microservice startup strategies are not efficient because they adopt a fixed total amount [1, 2, 14] and determine the number of fixed microservice instances startup based on the number of pre-estimated resources in the resource centre. The resource usage rate and load balance rate are low in traditional methods for microservices. Therefore, a new microservices deployment and startup method is needed.

The multi-objective evolutionary algorithms (MOEAs) have proved their effectiveness in solving multi-objective optimization problems [16]. MOEAs mainly include multi-objective genetic algorithms (NSGA, NSGA-II, NSGA-III, SPEA-II, etc.) [7, 23], multi-objective optimization based on decomposition [28, 31], multi-objective optimization based on artificial immune system [25, 34] and some other methods and applications [5, 33]. Their evolutionary nature based on population members makes it possible to obtain computational goals of multiple targets in a single run. Therefore, the MOEAs can solve the microservices deployment and startup problem.

## Problem definition and multi-objective model

The illustration of the microservices deployment and startup problem is shown in Fig. 1. Assume that several resource centres (RCs) are distributed in space, and each one has two kinds of resources: computation and storage resources. The startup of each microservice instances will consume a certain amount of resources. At the same time, a large number of service users are distributed in the same space. Each user requires one or more microservice instances to be serviced. The problem of microservice deployment and startup



**Fig. 1** The illustration of microservices deployment and startup problem

is to seek the optimal number of the running microservice instances in each resource centre. With this optimal strategy, the resources can be used efficiently, and the load balance in each RC can be increased.

According to the needs of microservices deployment, the optimal model should be constructed based on a typical multi-objective optimization model. Before the model is constructed, some definitions are provided as follows:

$$\text{Strategy}(\text{RC}, \text{MS}) = [\text{rc}_1(x_{11}, \dots, x_{1K}), \text{rc}_2(x_{21}, \dots, x_{2K}), \dots, \text{rc}_N(x_{N1}, \dots, x_{NK})]^T, \quad (1)$$

**Definition 1** Resource centres (RC). A resource centre is an entity that provides an environment for the deployment and startup of microservice instances. It contains computation and storage resources, and it can be distributed in different locations.

Assuming that there is a number  $N$  of RCs in the space,  $\text{RC} = \{\text{rc}_1, \text{rc}_2, \dots, \text{rc}_N\}$ , each resource centre contains two types of information (resource and location), which can be described as  $\text{rc}_i = (\text{RRes}(r_{\text{com}_i}, r_{\text{sto}_i}), \text{RLoc}(r_{\text{lat}_i}, r_{\text{lon}_i}))$ , where  $\text{RRes}(r_{\text{com}_i}, r_{\text{sto}_i})$  represents the existing computation resources  $\text{com}_i$  and storage resources  $\text{sto}_i$  in the  $i$ -th RC. To facilitate the model solution, the computation and storage resources are all normalized to a specific real value.  $\text{Loc}(\text{lat}_i, \text{lon}_i)$  are the latitude and longitude of the  $i$ -th RC location.

**Definition 2** Microservice collection (Microservices, MS). In this paper, the microservice collection refers to some independent functional services that are separate from a single big application. It can be described as follows:  $\text{MS} = \{s_1, s_2, \dots, s_K\}$  represents  $K$  independent microservices that are separate from a single application.  $s_i = (\text{SRes}(r_{\text{com}_i}, r_{\text{sto}_i}))$ ,  $i \in [1, K]$  represents the computation and storage resources of the  $i$ -th microservice when its instance is deployed and started up.

**Definition 3** User collection (Users, U). User collection refers to the consumers of microservices. Each user may consume a single microservice or a sequence of several microservices. Users are independent individuals.

It is assumed that there are  $M$  users in the space described as  $\text{Users} = \{u_1, u_2, \dots, u_M\}$ . Each user contains two types of information (service request information and location information), as  $u_i = (\text{UReq}(S_i(s_p, \dots, s_k)), \text{ULoc}(u_{\text{lat}_i}, u_{\text{lon}_i}))$ ,  $i \in [1, M]$ ,  $0 < p < k < K$ , where  $\text{UReq}(S_i(s_p, \dots, s_k))$  is the current requirement of the  $i$ -th user, and  $\text{ULoc}(u_{\text{lat}_i}, u_{\text{lon}_i})$  is the current location of the microservice.

**Definition 4** Microservice instances deployment and startup strategy (MDSS). For a number of  $N$  resource centres in the space  $\text{RC} = \{\text{rc}_1, \text{rc}_2, \dots, \text{rc}_N\}$ , the microservice instances  $\text{MS} = \{s_1, s_2, \dots, s_K\}$  deployment and startup strategy is expressed as  $\text{Strategy}(\text{RC}, \text{MS})$ . The mathematical expressions are as (1):

where  $\text{rc}_1(x_{11}, \dots, x_{1K})$  represents MDSS in resource centre 1.  $x_{11}$  indicates the number of startup instances of microservice 1 in resource centre 1.

The multi-objective optimization model for this problem is as Eq. (1).

It is assumed that there are a number of  $N$  resource centres  $\text{RC} = \{\text{rc}_1, \text{rc}_2, \dots, \text{rc}_N\}$  in space  $\mathbb{R}$ ,  $K$  functionally independent microservices  $\text{MS} = \{s_1, s_2, \dots, s_K\}$ , and  $M$  users  $\text{Users} = \{u_1, u_2, \dots, u_M\}$ , and the decision vector  $x = \text{Strategy}(\text{RC}, \text{MS})$  is a continuous independent variable.

$$\begin{cases} \min_{x \in \delta} F(x) = \min(f_1(x), f_2(x), \dots, f_L(x))^T \\ \text{s.t. } g_i(x) \leq 0 (i = 1, 2, \dots, m), \\ h_j(x) = 0 (j = 1, 2, \dots, n) \end{cases}, \quad (2)$$

where  $F(x)$  is the objective function of this model  $g_i(x)$  and  $h_j(x)$  represent two different constraints.  $L$  is the number of objective functions,  $m$  is the number of inequality constraints, and  $n$  is the number of equality constraints in the model. The model has five objective functions: computation and storage resource utilization efficiency, the real idle rate, and the computation and storage load balancing ratios. The five objective functions are defined as follows:

## Objective functions

The first objective function calculates the computation resource idle rate, which is defined as  $f_1(x)$ .

$$f_1(x) = 1 - \sum_{i=1}^N \frac{\text{rc}_i(x_{i1}, \dots, x_{iK}) \cdot \text{SRes}(r_{\text{com}_1}, \dots, r_{\text{com}_K})^T}{\text{RRes}(r_{\text{com}_i})} \Bigg/ N,$$

where  $\text{rc}_i(x_{i1}, \dots, x_{iK})$  represents the number of microservices that are running at the  $i$ -th resource centre.

The  $\text{SRes}(r_{\text{com}_1}, \dots, r_{\text{com}_K})$  represents the vector of computational resources consumed by the microservices.  $\text{RRes}(r_{\text{com}_i})$  represents the existing computational resource value of the  $i$ -th resource centre.

Similarly, the second objective function storage resource idle rate is defined as  $f_2(x)$ .

$$f_2(x) = 1 - \frac{\sum_{i=1}^N rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r_{sto_1}, \dots, r_{sto_K})^T}{RRes(r_{sto_i})} \Big/ N.$$

The third objective function is the real idle rate of the microservice. In general, a single microservice instance can be used for several service requests once it is started up. However, when the number of service requests exceeds the maximum load of a single microservice instance, extra instances need to be started. The maximum number of service users that a single microservice can handle can be represented as  $MSMaxUsers = [mu_1, mu_2, \dots, mu_K]$ . It is expected that the load of each microservice should be fully used.

The real idle rate for microservices is defined as  $f_3(x)$ .

$$f_3(x) = 1 - \sum_{i=1}^N \sum_{j=1}^K \frac{RealUserNum(s_j, rc_i)}{ServAbility(x_{ij}, mu_j)} \Big/ (KN),$$

$$ServAbility(x_{ij}, mu_j) = \begin{cases} x_{ij}mu_j & \text{if } x_{ij} \neq 0 \\ 1, & \text{if } x_{ij} = 0 \end{cases},$$

where  $RealUserNum(s_j, rc_i)$  is the actual number of users of microservice  $s_j$  at the resource centre  $rc_i$ . The calculation method will be described in detail later as the result of the service discovery strategy in Section III.

Under the premise of this user’s service discovery strategy, the feasible solution must satisfy the constraints (see the constraint section below), and the user can get at least one available microservice. At the same time, the microservice instance that is started up in each resource will identify the number of users as  $RealUserNum(s_j, rc_i)$ .

The fourth objective function is the computation load balancing ratio among the resource centres. It is not good for some resource centres to maintain high-load operation for a long time while other resource centres are idle.

The load balancing rate is calculated as  $f_4(x)$ .

$$f_4(x) = \sum_{i=1}^N \left( ComUsage(rc_i) - \overline{ComUsage(RC)} \right)^2,$$

$$ComUsage(rc_i) = \frac{rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r_{com_1}, \dots, r_{com_K})^T}{RRes(r_{com_i})},$$

$$ComUsage(RC) = \frac{\sum_{i=1}^K rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r_{com_1}, \dots, r_{com_K})^T}{\sum_{i=1}^K RRes(r_{com_i})}.$$

The fifth objective function is the storage load balancing ratio in the resource centres. The storage load balancing ratio is used to measure the storage balance degree of the resource centre. Generally, due to the existence of data synchronization tasks between resource centres, a resource centre with a large

amount of data storage may synchronize data to another one. It often occupies a large amount of communication resources and might cause congestion in services. Therefore, to reduce the data synchronization overhead, the storage load balancing of the resource centre should be considered.

The storage load balancing ratio is computed as  $f_5(x)$ .

$$f_5(x) = \sum_{i=1}^N \left( StoUsage(rc_i) - \overline{StoUsage(RC)} \right)^2,$$

$$StoUsage(rc_i) = \frac{rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r_{sto_1}, \dots, r_{sto_K})^T}{RRes(r_{sto_i})},$$

$$StoUsage(RC) = \frac{\sum_{i=1}^K rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r_{sto_1}, \dots, r_{sto_K})^T}{\sum_{i=1}^K RRes(r_{sto_i})}.$$

### Constraints

There are four constraints in the model. They are described as follows:

The first constraint is the completeness constraint of the service  $g_1(x)$ . All microservices must be guaranteed to start at least one instance in RC. The mathematical expression of the constraint is as follows:  $g_1(x) : x_{ij} \geq 1, 1 \leq i \leq N, 1 \leq j \leq KN$  is the number of resource centres, and  $K$  is the number of microservices in the single service.

The second constraint is the total amount of computation resource constraint  $g_2(x)$ . Because the computation resource in each RC is limited, the amount of computing resources occupied by started-up service instances cannot exceed its total amount.

$$g_2(x) : rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r_{sto_1}, \dots, r_{sto_K})^T < RRes(r_{sto_i}), 1 \leq i \leq N.$$

The third is the storage resource constraint  $g_3(x)$ . The amount of storage resources occupied by started-up service instance cannot exceed the total amount in each RC.

$$g_3(x) : rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r_{com_1}, \dots, r_{com_K})^T < RRes(r_{com_i}), 1 \leq i \leq N.$$

The fourth constraint is the total number of service sequence constraints  $g_4(x)$ . The total amount of all started-up microservice instances should be bigger than the total amount of user demand for microservices. The expression is as follows:

$$g_4(x) : \sum_{i=1}^M UserReq(u_i, s_j) \leq \sum_{i=1}^N x_{ij}mu_j, 1 \leq j \leq K.$$



$UserReq(u_i, s_j)$  indicates whether the  $i$ -th user has requested the service; it returns 1 if so and returns 0 if not.  $mu_j$  indicates the maximum number of service users that  $j$ -th microservice.

**Optimization problem model**

In summary, the microservice deployment and startup strategy model based on multi-objective optimization is described as follows:

Assuming that the MS set  $O = \{o_1, o_2, \dots, o_K\}$ , for the solution space decision vector  $x \in \Omega$

$$x = \begin{bmatrix} rc_1(x_{11}, x_{12}, \dots, x_{1K}), \\ rc_2(x_{21}, x_{22}, \dots, x_{2K}), \\ \dots, \\ rc_N(x_{N1}, x_{N2}, \dots, x_{NK}) \end{bmatrix}^T, \text{ and the model is}$$

$$\begin{cases} \min_{x \in \delta} F(x) = \min(f_1(x), f_2(x), \dots, f_5(x))^T, \\ \text{s.t. } g_1(x) : x_{ij} \geq 1, 1 \leq i \leq N, 1 \leq j \leq K, \\ g_2(x) : rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r\_sto_1, \dots, r\_sto_K)^T < RRes(r\_sto_i), 1 \leq i \leq N, \\ g_3(x) : rc_i(x_{i1}, \dots, x_{iK}) \cdot SRes(r\_com_1, \dots, r\_com_K)^T < RRes(r\_com_i), 1 \leq i \leq N, \\ g_4(x) : \sum_{i=1}^M UserReq(u_i, s_j) \leq \sum_{i=1}^N x_{ij}mu_j, 1 \leq j \leq K. \end{cases}$$

It can be known from the description of the model that the problem is a typical NP-hard problem. The problem is difficult to solve using the traditional polynomial method. Therefore, it is necessary to seek the evolutionary computation method.

**Multi-generations reference NSGA-III**

In this paper, multi-generations reference NSGA-III (MGR-NSGA-III) is proposed to solve the problem. Compared with other multi-objective optimization algorithms such as NSGA-III, SPEA-II, etc., MGR-NSGA-III can quickly converge to the optimal solution and effectively find the Pareto frontier.

Non-dominated sorted genetic algorithm (NSGA) and its variants are one of the most classic algorithms for solving multi-objective optimization problems [20]. NSGA is a multi-objective optimization algorithm based on the Pareto frontier to optimize multiple targets. The NSGA-II algorithm [8] is also known as the non-dominated sorting genetic algorithm with elite strategy. It can find a way to retain the elite solutions as the next generation’s parents according to the dominance relationship between the

offspring. NSGA-III [7], also known as the reference point non-dominated sorting genetic algorithm and based on the NSGA-II computational framework, further improves the elites retention strategy. It uses a reference point strategy based on maintaining population diversity. It improves the computational efficiency of seeking multiple elite solutions in the non-dominated layer.

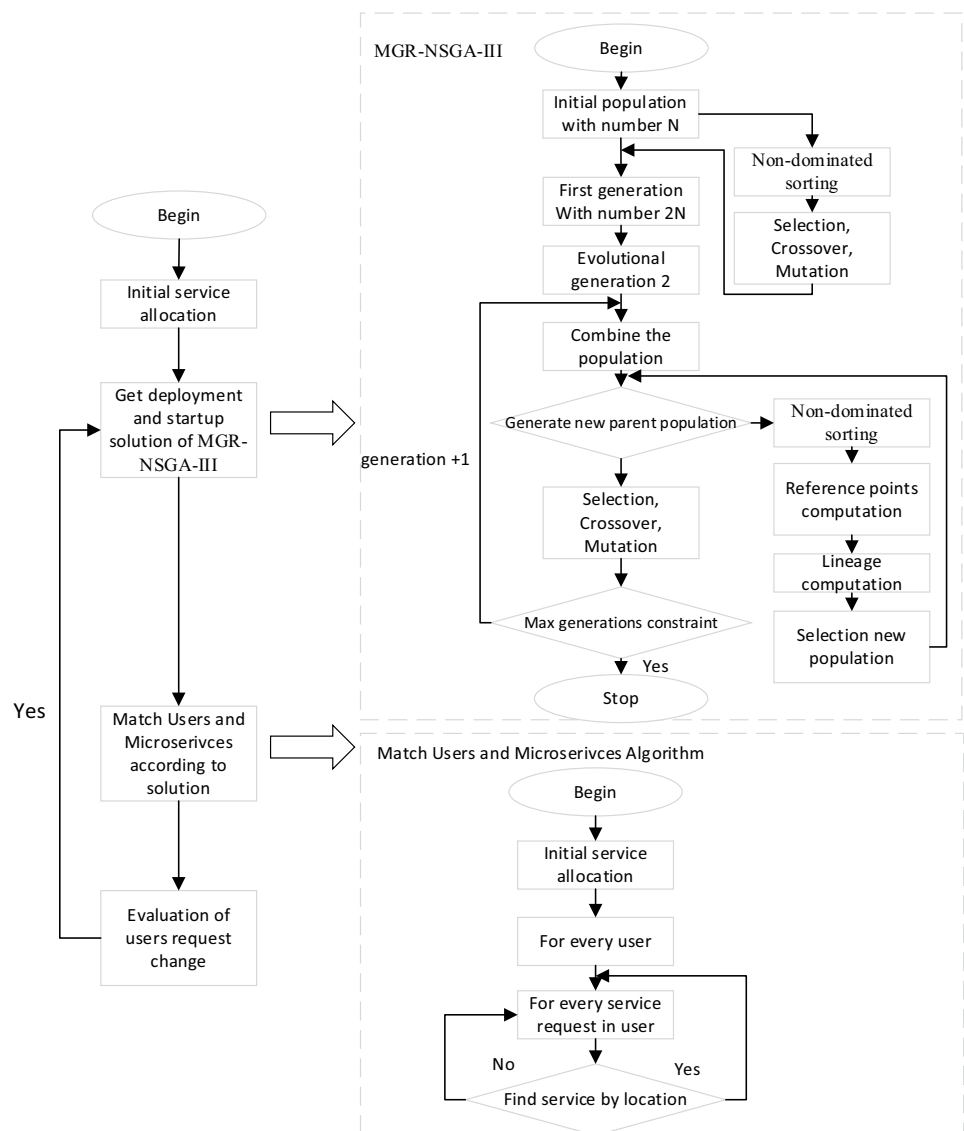
The MGR-NSGA-III uses a knowledge-driven population selection method (named the learning elite population selection method) which was designed based on the NSGA-III in the non-dominated level. The learning elite population selection method can record the dominance level of historical generations. It can select elite population members with high genetically pure lineage as the knowledge driven and use them for the mutation of the next generation. The descendants which are inherited

by high dominated-level ancestors are more likely to be selected. The selection of the population is based not only on this generation of inheritance but also the knowledge about lineage of the population. The mutation and crossover of the population, which refer the potential lineage, are more suitable for solving multi-objective optimization problems.

**Main process**

Once the microservices and users are allocated in the space, the system should decide how and where the microservice instances starts up and how users can access the microservices. The main process to handle this problem is shown in Fig. 2. First, the microservice allocation vector is initiated in RC, including the real starting up microservice vector, remaining computation and storage ability vector and user access vector. Second, the optimal deployment and starting up solution is calculated by the MGR-NSGA-III (the solving details will be explained in next section), and the microservices will be initiated according to this solution. Then, users and microservices will be matched. For every user in the users’ set, find the nearest service by location, if the service

**Fig. 2** Main process framework of microservices deployment and startup problem



is available, complete the matching process of this user, and if not, continue to find the nearest service location, until all user’s find the services, then stop the matching process. RC will evaluate the variation of the user requests to decide whether it is necessary to recompute the optimal policy by MGR-NSGA-III.

**MGR-NSGA-III**

1. Algorithm’s solving process

MGR-NSGA-III integrates the evolutionary mechanism of NSGA-III with the lineage. Rather than considering the dominance and crowding degree between population members, the elite strategy selection will compute the lineage of the population and compare their historic relationship of parents. In the same situation, population members with higher unsorted-dominated rank ancestry are selected.

**Algorithm 1:** *MGR-NSGA-III* ( )**Input:** *Parent population*  $P_t$ , *Referense Points* RF**Output:** *Parent population*  $P_{t+1}$ **Get:** *Next generation population*


---

```

1:  $Q_t \leftarrow \emptyset, \text{Lineage}_t \leftarrow \emptyset, i = 1;$ 
2:  $Z_t \leftarrow \text{Offspring\_Lineage}(P_t)$ 
3:  $\text{Joint}P_{t+1} \leftarrow Z_t \cup P_t;$ 
4:  $(R_1, R_2, \dots) \leftarrow \text{Rank}(\text{Joint}P_{t+1});$ 
5:  $\text{AddRankLineage}(R_1, R_2, \dots)$ 
6: While ( $|Q_t| < \text{MaxPopuSize}$ )
7:    $Q_t \leftarrow Q_t \cup R_i, i = i + 1;$ 
8: End While;
9:  $f_i \leftarrow R_i;$ 
10: If  $|Q_t| = \text{MaxPopuSize}$ 
11:   Break;
12:  $P_{t+1} \leftarrow Q_t + f_i$ 
13:  $\text{Idealpoints}_{t+1} \leftarrow \text{TranslateObjectives}(Q_t)$ 
14:  $\text{ExtremPoint} s_{t+1} \leftarrow \text{FindExtremPoints}(Q_t)$ 
15:  $\text{Intercepts}_{t+1} \leftarrow \text{ConsHyper}(Q_t, \text{ExtremPo int} s_{t+1})$ 

16:  $\text{RF}_{t+1} \leftarrow \text{Norm}(Q_t, \text{Intercepts}_{t+1}, \text{Idealpoints}_{t+1}, \text{RF})$ 

17: Associate( $Q_t$ )
18: While ( $|Q_t| < \text{MaxPopuSize}$ )
19:    $\text{min\_rp} \leftarrow \text{FindNicheReferencePoint} ( );$ 
20:    $\text{PointsChosen} \leftarrow \text{SelectChusMem}(\text{min\_rp});$ 
21:   If  $\text{PointsChosen} = \emptyset$ 
22:      $\text{PointsChosen} = \text{RF}_{t+1} \cdot \text{Remove}(\text{min\_rp});$ 
23:   Else
24:      $Q_t \leftarrow Q_t \cup \text{PointsChosen};$ 
25:   End If;
26: End While;

```

---

The input of the algorithm is the previous generation population  $P_t$  (if it is the first generation, the population is randomly initialized), the reference point set of the population (calculation method in [7]). First, a set of population  $Q_t$ , lineage collection  $\text{Lineage}_t$ , is initialized. Then, crossover and mutation of the population are operated, and the lineages are stored in the offspring;

see lines 1–2 of Algorithm 1. The detailed calculation process  $\text{Offspring\_Lineage}(P_t)$  will be displayed in Algorithm 2. The parent  $P_t$  and offspring  $Z_t$  are incorporated into  $\text{Joint}P_{t+1}$ ; then, using the non-dominated sorting of  $\text{Joint}P_{t+1}$ , a result set  $(R_1, R_2, \dots)$  is obtained, and the resulting rank code is stored as a lineage collection  $\text{Lineage}_{t+1}$  to the current population. See lines 4–5 in Algorithm 1. Second, the dominated solution of the previous  $i$  levels is selected until the number of solutions of the first  $i + 1$  levels is greater than the maximum population size. Then, besides the solution of the previous  $i$  levels, some solutions are selected from the solutions in the  $i + 1$ th level according to their reference point distance and lineage rank list until the population size is equal to the maximum population size. For its selection procession, see line 12–25 in Algorithm 1. Calculate the ideal point, extreme points, and reference points and then obtain the reference vectors [7].  $\text{ConsHyper}(Q_t, \text{ExtremPoints}_{t+1})$  function is a definition in paper [7], which indicates that combined population members  $Q_t$  and  $\text{ExtremPoints}_{t+1}$  are projected on a hyper-plane and a clustering operation is performed on the hyper-plane to select a desired number of clusters. Find the solutions that are near the reference vectors and have high lineage rank list of ancestors as candidates of the next generation population. The details on this selection process will be displayed in Algorithm 3.

## 2. Encoding

This paper employs a binary encoding method. The length of the encoding is determined by the maximum number of microservice instances that a single resource centre initiate. This paper assumes that each resource centre can initiate 16 microservice instances. Therefore, the number of initiated instances in a resource centre can be expressed by a four-digit binary code. Obviously, the total length of the binary code is  $4 \times N \times K$ .

## 3. Crossover and mutation operation

Based on the traditional crossover and mutation strategy, a lineage crossover is proposed. According to the non-dominated sorting results of the parents, it aggregates the ancestry lineage to the offspring in the crossover process. At the same time, to maintain the freshness of the lineage, the loss mechanism of the ancestral lineage of distance is adopted. The calculation of the lineage crossover and mutation process is as Algorithm 2.



**Algorithm 2:** *Offspring\_Lineage*( )

**Input:** *Parent population*  $P_t$   
**Output:** *Offspring population*  $Z_t$   
**Get:** Offspring with lineage by mutation and crossover

```

1:  $O_t \leftarrow \emptyset, Lineage_t \leftarrow \emptyset, i = 1;$ 
2: While ( $i < MaxpopuSize$ )
3:    $PT_t \leftarrow P_t(i) \cup P_t(i+1);$ 
4:    $New\ Offspring \leftarrow Corssover(PT_t);$ 
5:   If  $|P_t(i).rankList| > MaxLineageNum$ 
6:      $Remove\ first\ MaxLineageNum / 2\ ranks;$ 
7:   End If;
8:   If  $|P_t(i+1).rankList| > MaxLineageNum$ 
9:      $Remove\ first\ MaxLineageNum / 2\ ranks;$ 
10:  End If;
11:   $Offspring\_lineage \leftarrow$   

    $P_t(i).rankList \cup P_t(i+1).rankList;$ 
12:   $Mutation\ Operator\ on\ Offspring;$ 
End While;
    
```

Algorithm 2 first initializes the result list  $O_t$  and lineage list  $Lineage_t$ . Then, all of the population members are traversed, and the two adjacent bodies are crossed by the traditional strategy; see lines 2–4. To ensure that the number of lineage rank lists does not exponentially explode with the increase in evolutionary generations, a reduction strategy is introduced here. A max threshold number of lineage rank lists in each population is defined as  $MaxLineageNum$ ; if the number of lineage rank lists in the parent population exceeds this threshold, the  $MaxLineageNum/2$  lineage rank value of the ancestors will be removed from the far distance to nearby. The freshness of the lineage is also guaranteed in that way; see lines 5–8. Then, the contents of the parent lineage rank lists are merged with the offspring. Finally, the offspring are mutated by probability; see lines 9–11.

4. Selection operation

In this paper, the knowledge-driven population selection strategy uses the lineage calculation of the candidate set, which is based on the NSGA-III reference point distance calculation. The calculation process is as follows:

When selecting the optimal object based on the reference point in the 20th row of Algorithm 1, it is necessary to sort the selection according to the lineage. The specific process is as in Algorithm 3.

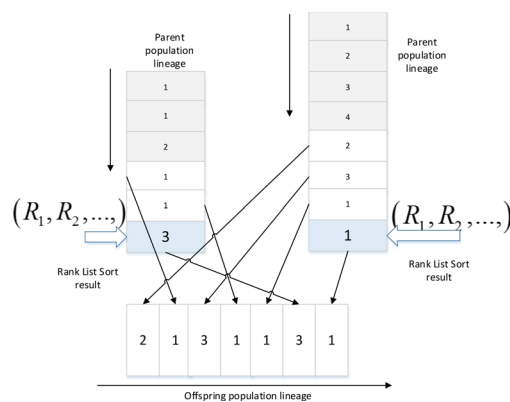


Fig. 3 How offspring lineage is inherited from parents

**Algorithm 3:** *SelectClusMem*( $min\_rp$ )

**Input:** *Reference Point*  $rp$ , ranked population  $Q_t, f_t$   
**Output:** Chosen population  $C_t$   
**Get:** finding the best population according to reference point and lineage

```

1:  $C_t \leftarrow \emptyset, i = 1;$ 
2: If  $rp.totalNearNumber > 0$ 
3:   If  $rp.nearNumber(Q_t) = 0$ 
4:      $C_t \leftarrow rp.findNearestNumber(f_t);$ 
5:   Else
6:      $flag \leftarrow rp.nearNumber(f_t).random();$ 
7:      $TN \leftarrow rp.nearNumber(f_t)$ 
8:     Foreach  $tn$  in  $TN$ 
9:        $ave_m = Average(tn.getLineage);$ 
10:    End Foreach;
11:     $flag = ArgMin(ave_m)$  in  $TN;$ 
12:     $C_t \leftarrow TN(flag);$ 
13:  End If;
14: Return  $C_t;$ 
    
```

Algorithm 3 aims to find the best population members in the  $f_t$  rank. On the selection of non-dominated solutions, the lineage factor is considered in the retention strategy of the elite solution. First, the nearest distance population set of the reference point  $rp$  is found. If the number of sets is larger than 0, then there are two cases. The first is that if the number of the nearest distance population set of  $rp$  in  $Q_t$  is 0, then the nearest distance population of  $rp$  will be selected; the second is that if the number of the nearest distance population set of  $rp$  in  $Q_t$  is bigger than 0, then a population with the best historical lineage is selected according to the average lineage rank sorting.

---

Algorithm 4: *findNearestServiceForUsers()*

Input: *solutionList* and model (including the *centerList* and *userList*)

Output: true or false

Get: *userServiced* (including used microservices list of each resource centre)

---

```

1: nearestServiceForUsers(sLs, cLs, uLs)
2: For every center in cLs
3:   center.realMicroServicesStart ← sLs;
4:   center.remMicSerAbility ← sLs;
5:   center.userServiced ← NULL;
6: End For
7: For every user in uLs
8:   For every mSer in user.SerListRequest()
9:     centerList.setCenterFoundFlag(0);
10:    If !findNeaSerbyLo(mSer, user, cLs)
11:      Return false;
12:    Else Continue;
13:    End If;
14:  End For;
15: End For;
Return true;

```

---

An example of the lineage rank list process is presented in picture 3. Assume that two population members with lineage rank list [1, 1, 1, 1, 2] and [1, 1, 2, 2, 3, 3, 4] need to crossover. The MaxLineageNumber is 6. First, the rank list of each population will be calculated and added into the lineage rank list, assumed to be 3 and 1. Then, the lineage rank lists of the two population members are updated to [1, 1, 1, 1–3] and [1, 1, 1, 2, 2, 3, 3, 4]. As the limitation of the max number of lineages, we use the latest half of the parents' lineage rank list to crossover, which are [1, 1, 3] and [1, 1–3]. Finally, after crossing each rank element, we obtain the offspring's lineage [1, 1, 1, 1–3, 3] as shown in Fig. 3.

##### 5. Match users and microservices provider algorithm

When the best deployment strategy is found, some matching algorithms between users and microservice providers should be applied to validate the strategy in the real service environment.

The user service policy mentioned in the previous section adopts a distance-based service selection algorithm as in Algorithm 4. First, for each user, the nearest available microservices is searched for according to the location principle; then, the microservices are checked for whether they have reached the maximum service load. If not, this user request is sent to the service; otherwise, search for

the next nearest one by distance. Repeat this until all users discover available microservices.

Details on the calculation process are in Algorithm 4. The first step is to initialize the actual microservice start quantity vector *realMicroServicesStart* of each resource centre, the remaining resource vector of each resource centre *remMicSerAbility*, and the user list served by each resource centre *userServiced*, as in lines 2–6. Then, traverse each user in *uLs* to find the most suitable resource centre service. For each service request in the user service request list, initialize the access identifier of each resource centre to 0; see lines 8–9. Recursively search for the nearest resource service centre that meets the user's location condition. If it is found, continue to find the next service. If not, return 'false'; see lines 10–13. Finally, return 'true' when all services for all users are found in all resource centres.

The implementation of the function *findNeaSerbyLo*(*mSer, user, cLs*) is shown in Algorithm 5. The basic process consists of several steps. First, find the nearest resource centre that has not been visited which is identified by *CenterFoundFlag* based on the user's location. Get the RC number *neLoCenter* with the function *findNeSerbyLo*(*user.location*) return value. If it returns -1, then all resource centres have already been visited, there is no solution, return 'false'; see lines 2–3. Second, determine if there is an available service in the resource centre that matches the user's needs. If the number of services

is greater than or equal to 1, the user will be added to the service user list of this centre, and the list of available services in this resource centre is updated by adding this service; see lines 3–6; if not, the access identifier CenterFoundFlag of the resource centre will be changed to 1. The most suitable service of the user is recursively searched for; see lines 9–11.

**Algorithm 5:** *findNeaSerbyLo(mSer, user, cLs)*

Input: *microServices*, *user* and *centerList*

Output: true or false

Get: Nearest centre for singer user

```

1: findNeaSerbyLo(mSer, user, cLs)
2: Int neLoCenter ← findNeSerbyLo(user.loc)
3: If ( neLoCenter == -1 ) return false;
4: If( remMicSerAbility [ mSer .Num () ] >= 1)
5:   cLs . neLocCenter.Users.Serviced().add( user );
6:   remMicSerAbility [ mSer.MirSerNum() ]--;
7:   return true;
8: Else
9:   cLs .get( neLocCenter ).
   setCenterFoundFlag(1);
10: findNeSerForOneUser(mSer, user, cLs);
11: End If;
    
```

6. Algorithm complexity analysis

The main process of the microservices deployment strategy comprises two parts: the MGR-NSGA-III and matching users with services. The complexity of MGR-NSGA-III is as follows. Algorithm 1 shows the whole process of MGR-NSGA-III. Algorithm 2 and Algorithm 3 need to be computed once for each generation of iterations in MGR-NSGA-III. It is assumed that *T* represents the number of objective functions, *Q* represents the population size, *G* is the maximum allowable number of lineage in each

population, and *H* is the reference number. The computational time complexity for the insertion of lineage with the 2*Q* population members in Algorithm 1 rows 2–8 is  $O(Q) \log^{T-2} Q$  in extreme cases (i.e., each lineage requires computational choices, and each lineage has the maximum allowable number). Each reference computation complexity is  $O(T \cdot Q \cdot H)$ , and each crossover and merging time of the lineage is  $O(Q \cdot G^2)$ . The computational complexity chosen for the lineage of each generation is  $O(Q \cdot G) \log^{T-2} Q$ , so the computational complexity of each generation in MGR-NSGA-III is due to  $O(Q \cdot G) \log^{T-2} Q + O(Q \cdot G^2) + O(T \cdot Q \cdot H)$ . The complexity of Algorithm 4 is  $O(N) + O(M \cdot K)$  (this assumes that each user contains a request for all *K* microservices, which is actually less than *K*). The essence of Algorithm 5 is a recursive calculation with a complexity of  $O(M^2)$  during which insertion and selection of pedigrees is required. Because in each generation of Algorithm 1, we should use Algorithm 4 and Algorithm 5 to evaluate the population, the whole process of the strategy is  $\text{MaxIterations} \cdot (O(Q \cdot G) \log^{T-2} Q + O(Q \cdot G^2) + O(T \cdot Q \cdot H) + O(M^2) + O(M \cdot K))$ .

**Experiment analysis**

**Datasets**

Two real datasets were used in the experiment. One dataset was the New York social network user check-in dataset, which is used to simulate different users distributed in different spaces to request services for different RCs at the same time. The other dataset was the Tokyo social network user check-in dataset [30]. A microservice dataset is a typical data application service system where a single application system is divided into ten microservices: reporting, administrator management, GIS, message transmission, data quality censor, data collection, data comprehensive maintenance, settlement, data visualization displays, and permission controls.

**Table 1** Values of regular parameters in different algorithms

| Parameter name             | MGR-NSGA-III                       | MGR-NSGA-III               | SPEA-II                    |
|----------------------------|------------------------------------|----------------------------|----------------------------|
| crossover                  | Simulated binary lineage crossover | Simulated binary crossover | Simulated binary crossover |
| crossoverProbability       | 0.9                                | 0.9                        | 0.9                        |
| crossoverDistributionIndex | 20                                 | 20                         | 20                         |
| mutation                   | Polynomial mutation                | Polynomial mutation        | Polynomial mutation        |
| mutationProbability        | 1.0/numberOfVariables()            | 1.0/numberOfVariables()    | 1.0/numberOfVariables()    |
| distributionIndex          | 20                                 | 20                         | 20                         |
| MaxIterations              | 30/50/100/150/200                  | 30/50/100/150/200          | 30/50/100/150/200          |
| MaxPopulationSize          | referencePoints.size()             | referencePoints.size()     | Same as MGR-NSGA-III       |

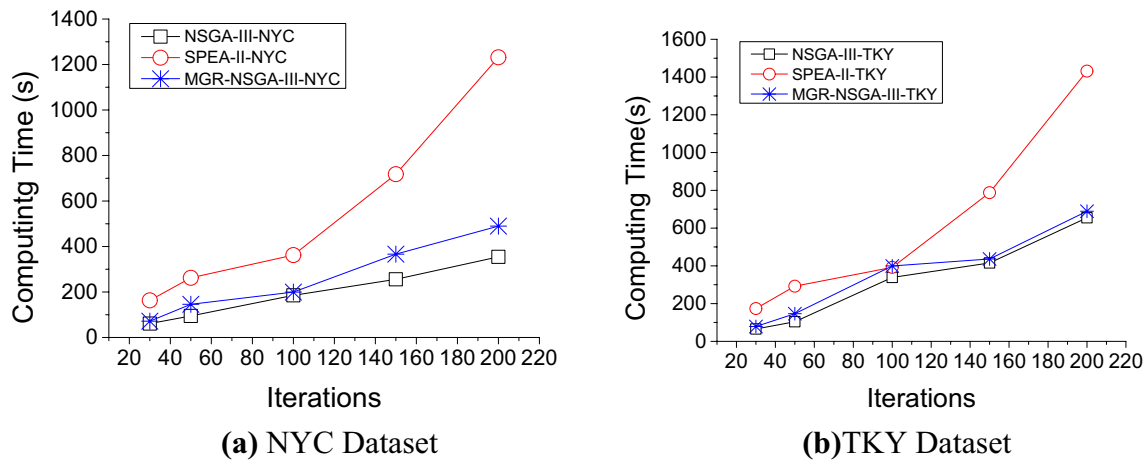


Fig. 4 Calculation time comparison of different iterations of different algorithms

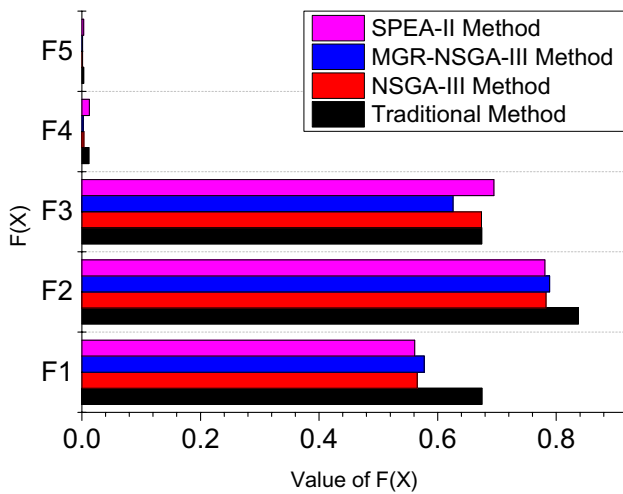


Fig. 5 Comparison of objective function under different microservice scheduling strategies

### Algorithms

NSGA-III and SPEA-II are chosen as competitor algorithms. Among the multi-objective optimization algorithms, SPEA2 and NSGA-III are two representative elite algorithms, which have become the standard for comparing the performance of other multi-objective optimization evolutionary algorithms. MGR-NSGA-III is developed from NSGA-III, thus, to demonstrate its effectiveness, NSGA-III is chosen. SPEA-II, as is known, is a classic multi-objective optimizer, which is often chosen as baseline competitor algorithm. Besides, SPEA2 can also be used to solve the problem of Microservices deployment problem. Considering all above reasons, SPEA2 and NSGA-III were selected to be compared with MGR-NSGA-III.

### Parameters

In this paper, MGR-NSGA-III is proposed to solve the microservices deployment and starting up strategy problem. The experimental parameters were as follows:

- (i) There are five resource centres in New York and Tokyo that are dedicated to the operation of the data service microservice system;
- (ii) Each resource centre has 300 units of computing resources and 300 units of storage resources;
- (iii) Starting a microservice instance requires 20 units of computing resources and 10 units of storage resources, and each instance can serve 10 users at the same time.

The MGR-NSGA-III value of regular parameter is shown in Table 1.

The main configuration of the computer used in the experiments is as follows: CPU Intel(R) Core (TM) i7-5600U 2.6 GHZ, 8G RAM memory.

After 20 operations to take the average running time for each algorithm, Fig. 4 shows the running time of different algorithms in different datasets. As the figure shows, solving the problem using SPEA-II would cost much more time than using NSGA-III and MGR-NSGA-III, especially when the number of iterations is larger. Additionally, MGR-NSGA-III requires a slightly longer time to compute in both datasets than NSGA-III. MGR-NSGA-III improves the selection and retention strategy of the elite solutions, it retains those elites with better lineage. Compared with SPEA-II and NSGA-III, these selection and retention calculations require a certain amount of resources, but they can get better solutions, so it has greater advantages in indicators such as average calculation rate, storage idle rate, and actual microservice idle rate.

From the optimal frontier solutions of these algorithms, several sets of solutions were selected to compare the traditional microservice deployment strategies without multi-objective function optimization. The results are shown in Fig. 4.

Figure 5 shows the deployment policy of the fixed average start service instance under the traditional combined microservice strategy after 20 operations for each algorithm. The resource idle rate is relatively high, and the actual usage rate of the service is low. The computation resource idle rate, storage resource idle rate, and actual idle rate of the traditional microservices deployment strategy in NSGA-III were 13.21%, 5.2%, and 16.67% higher, respectively. Compared with the traditional method, the calculation and storage resource idle rates of the traditional microservices deployment and startup strategy in MGR-NSGA-III were

60.51% and 2.2% higher, respectively, while the actual idle rate of microservices was 4.62% higher. In summary, the microservices deployment strategy based on multi-objective optimization has better service performance.

Furthermore, different evolutionary multi-objective algorithms have different performance. We will compare the MGR-NSGA-III that we proposed with classic methods as follows:

It can be seen from Figs. 6, 7 and 8 that the stability of the three multi-objective optimization algorithms is good under different evolutionary generation conditions. The range of the optimal solutions became consistent after 200 iterative generations. From the uniform variation of the distribution of the solution, the five objective functions of MGR-NSGA-III remained basically unchanged before the 100th generation, and the distribution of  $f(3)$

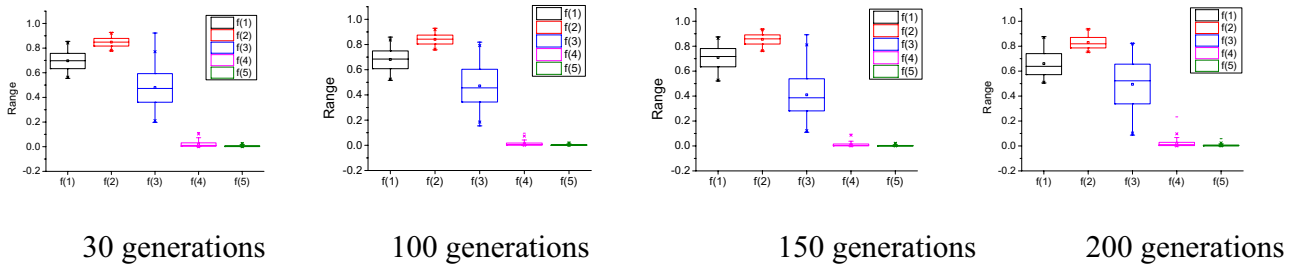


Fig. 6 Box chart of different generations for MGR-NSGA-III

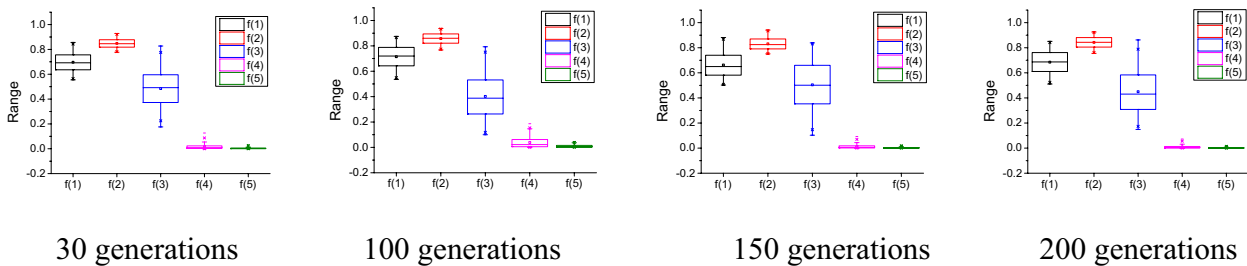


Fig. 7 Box chart of different generations for NSGA-III

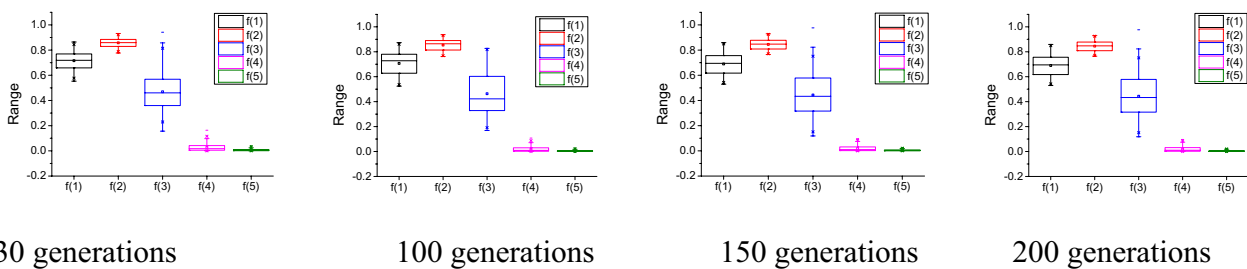
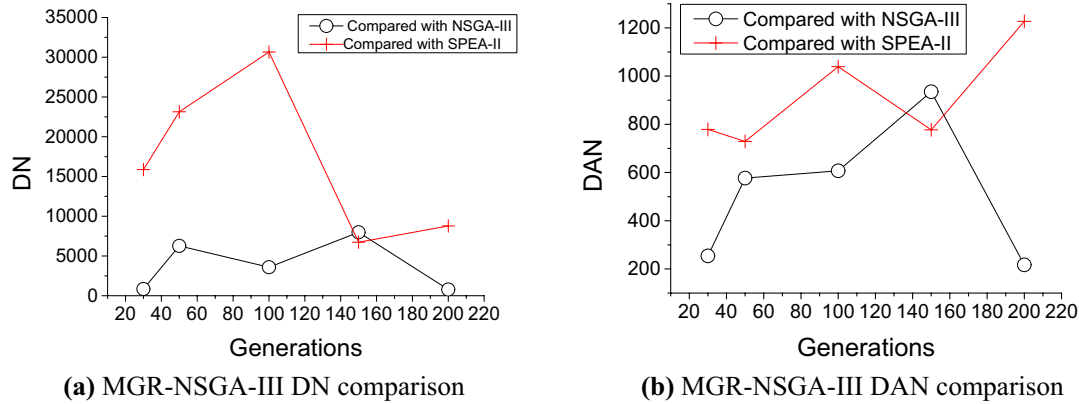


Fig. 8 Box chart of different generations for SPEA-II





**Fig. 9** Comparison of MGR-NSGA-III with NSGA-III and SPEA-II solutions in dominance

converges greatly at the 150th generation. The upper and lower quartile range changed from 0.32–0.6 to 0.23–0.52. The convergence speed of MGR-NSGA-III was faster than that of SPEA-II and NSGA-III. In NSGA-III and SPEA-II, there was no significant change in the value distribution

range of each target, and the final convergence value was slightly worse than that of MGR-NSGA-III. Finally, the upper quartile and lower quartile range converged between 0.3 and 0.6 in NSGA-III and SPEA-II.

**Definition 5** Dominating number (DN). Assuming population set *A* and population set *B*, the dominating number of *A* to *B* is defined as the following equation:

$$DN = \sum_{i=1}^{|A|} \text{Dom}(A_i, B), \text{ Dom}(A_i, B_j) = 1, \text{ if } B_j \text{ is completely dominated by } A_i, \text{ and vice versa } \text{Dom}(A_i, B_j) = 0.$$

**Definition 6** Dominating absolutely number (DAN). Assuming population set *A* and population set *B*, the dominating absolutely number of *A* to *B* is defined as the following equation:

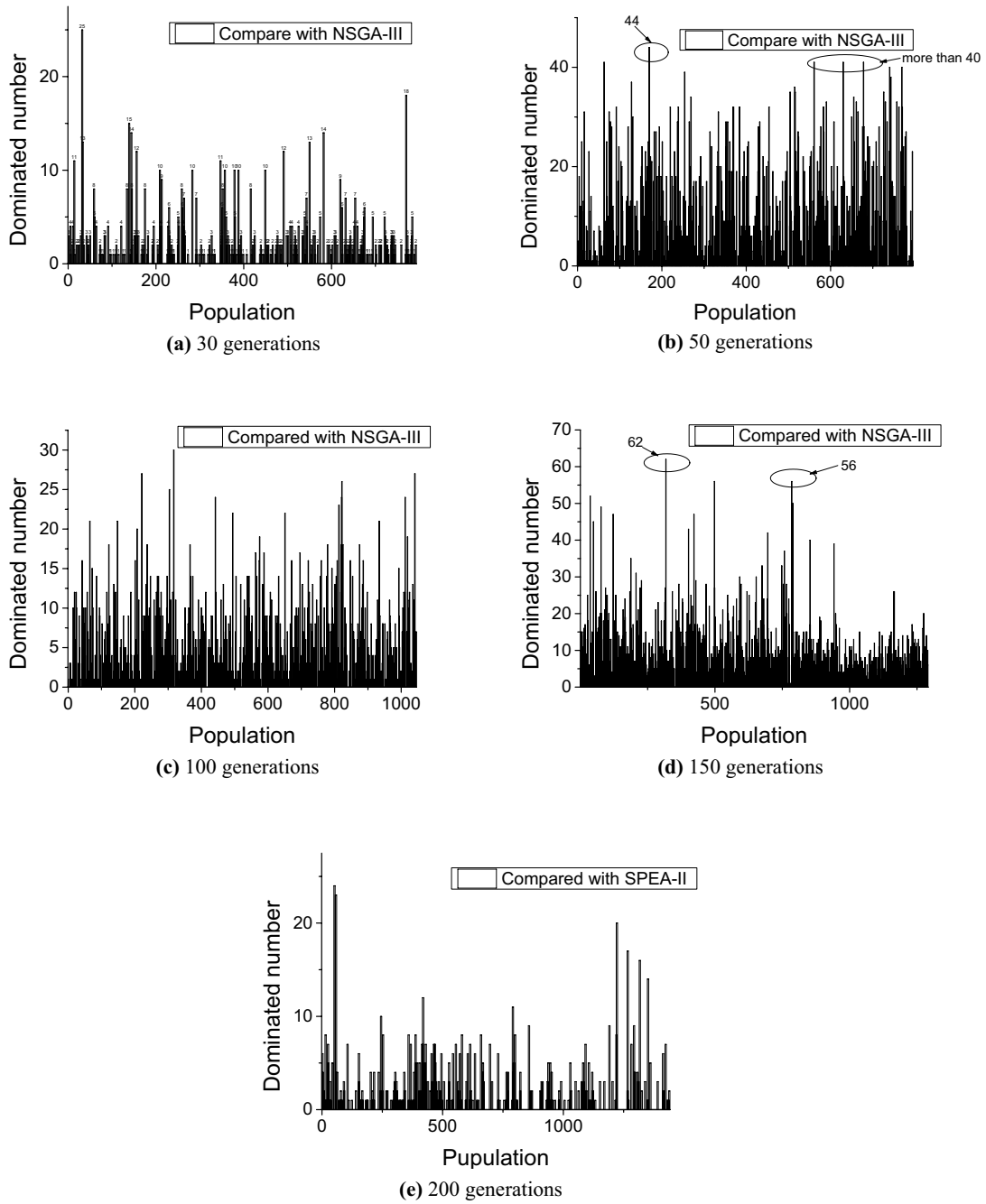
$$DAN = \sum_{i=1}^{|A|} \text{Dom}(A_i, B), \text{ Dom}(A_i, B) = 1, \text{ if } \exists B_j \in B, A_i \text{ completely dominates } B_j, \text{ and vice versa } \text{Dom}(A_i, B) = 0.$$

DN and DAN express the dominated relationship between the two population sets. The convergences of different algorithms were also compared by DN and DAN.

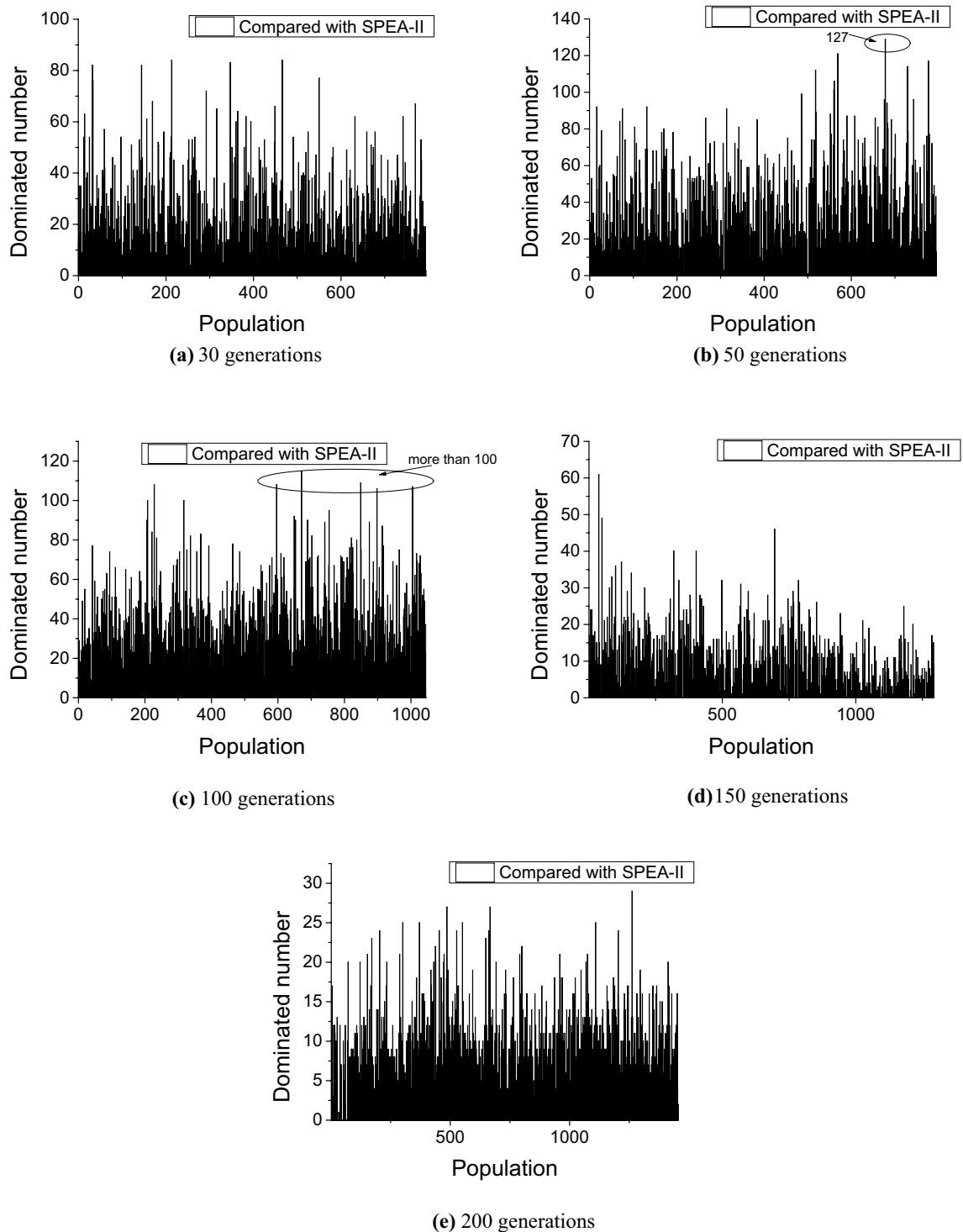
It can be seen from Fig. 9a that the convergence rate of the MGR-NSGA-III solution was higher. When the evolutionary calculation reached 50 generations, the NSGA-III and SPEA-II solutions were dominated by MGR-NSGA-III (DN of 6270 and 23,134, respectively). In addition, the DN of the solutions in MGR-NSGA-III was significantly high after 100 generations compared with SPEA-II (DN is 30,657). It can also be seen in Fig. 9b of the figure that after 50 generations, DAN in MGR-NSGA-III reached 577 and 729 compared with NSGA-III and SPEA-II, respectively. Afterwards, DAN reached 607 and 1,039. Therefore, MGR-NSGA-III is clearly more efficient in calculating the optimal strategy for running microservice instances.

Figures 10 and 11 show the dominating solutions num-

ber between MGR-NSGA-III and NSGA-III or SPEA-II. From Fig. 10, it can be seen that of the 50/100/150 generations the maximum DN of the population is more than 30. The maximum DN in 150 generations is even higher than 60 in Fig. 10d. Using MGR-NSGA-III, the number of dominated population can be increased. In addition, the



**Fig. 10** Comparison of MGR-NSGA-III with NSGA-III in dominating solutions for different generations



**Fig. 11** Comparison of MGR-NSGA-III with SPEA-II in dominating solutions for different generations

MGR-NSGA-III algorithm has a good convergence rate for computations of less than 200 generations. As shown in Fig. 11, the dominated number may be more than 120/100 in the 50/100th generation compared with SPEA-II.

A comparison between different microservice deployment and startup strategies based on multi-objective optimization was made. It can be seen from the red circles in Fig. 12, the number of started-up microservices with NSGA-III and SPEA-II is extremely high (as in Fig. 12a) or low (as in

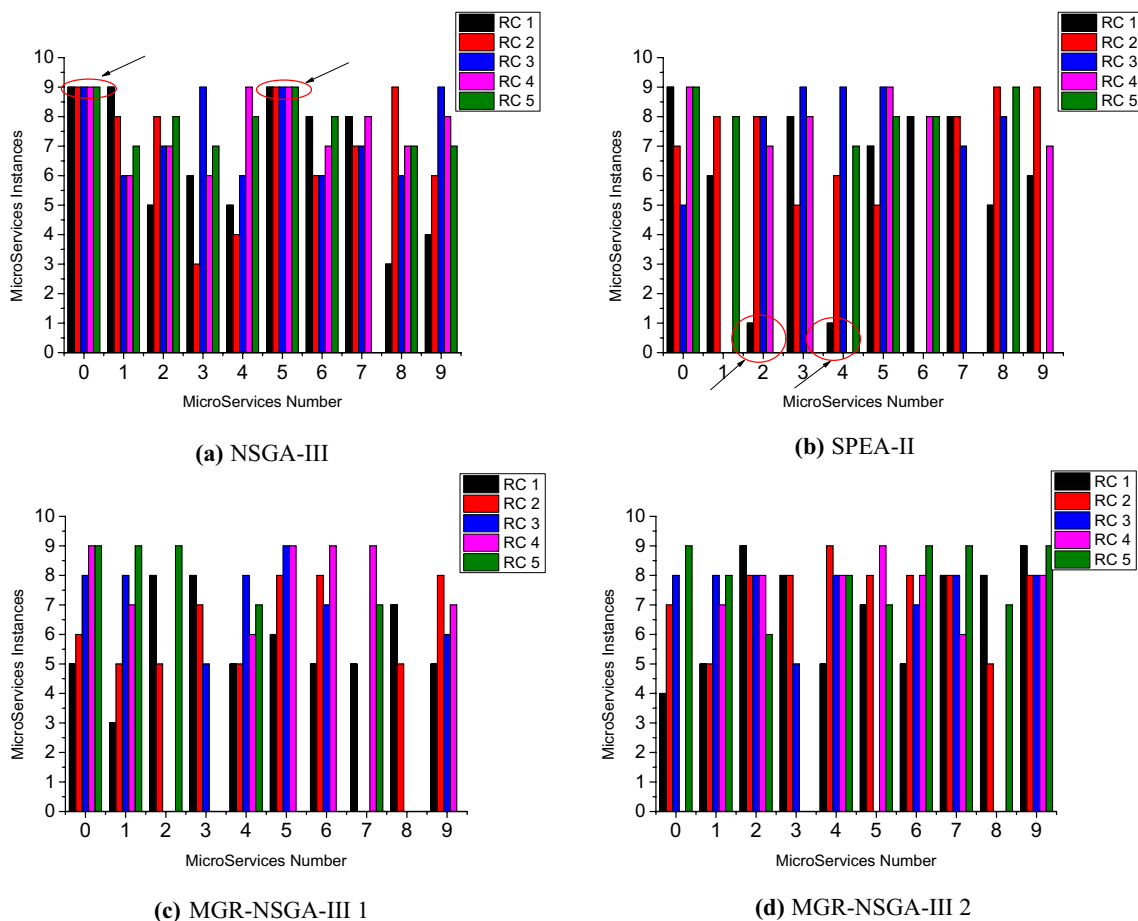


Fig. 12 Real deployment and startup results of microservices in different evolutionary algorithms

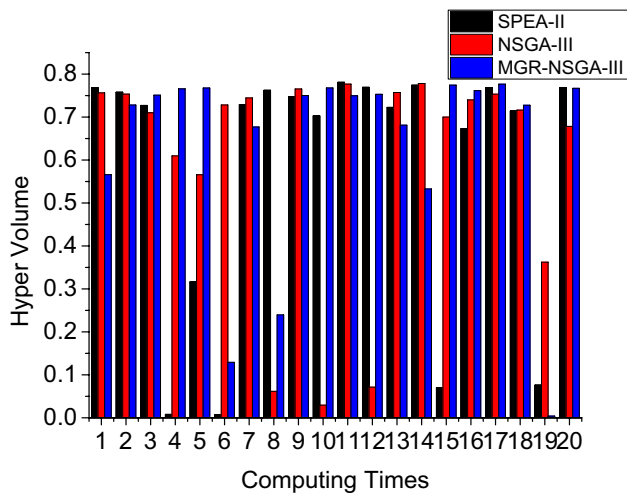


Fig. 13 Comparison of different algorithms in hyper volume

Fig. 12b) in different resource centres, indicating poor load balancing. Conversely, the startup load in MGR-NSGA-III is relatively balanced as in Fig. 12c, d.

Table 2 Average value of hypervolume in different algorithms

| Algorithms   | Average value of hyper volume with 20 times |
|--------------|---|
| SPEA-II      | 0.582211                                    |
| NSGA-III     | 0.602875                                    |
| MGR-NSGA-III | 0.633464                                    |

The hypervolume values of different algorithms are compared. The hyper volume [5] is popularly used to evaluate the convergence and distribution of MOEAs’ solutions.

NSGA-III, SPEA-II, and MGR-NSGA-III are used to solve the microservice deployment optimization problem. The calculations have been performed 20 times. It can be seen in Fig. 13 that MGR-NSGA-III represented by the blue part is slightly better than the other two algorithms, especially in the 4th and 5th calculation results, the hyper volume value is significantly higher than the other two algorithms. Moreover, the average values of the hyper volume of the three algorithms with 20 times are shown in Table 2,

MGR-NSGA-III algorithm can get the highest average value of hyper volume when comparing with SPEA-II and NSGA-III as well.

## Conclusion

Microservices split a complex application into a number of multiple sub-services with well-defined boundaries. The distributed deployment of these sub-services to different service centres provides services for users with the advantages of flexibility, scalability, and high availability. However, a series of problems have arisen in the use of microservices, and the optimization of deployment and startup microservices is one of the key challenges.

This paper introduces a knowledge-driven evolutionary algorithm for the deployment and startup problem of microservices for the first time. It improves this NSGA-III by considering the lineage knowledge of each generational population using MGR-NSGA-III. First, a microservices deployment and startup problem model based on multi-objective optimization was constructed. Some objective functions and constraints of the problem were defined. Second, NSGA-III was improved by knowledge driven to solve the microservice deployment and startup problem. Finally, several experiments were presented to evaluate the performance of different methods. A comprehensive evaluation of the algorithm's time efficiency, convergence degree and calculation effect were given. In conclusion, MGR-NSGA-III works well on microservice deployment and start-up problems.

In the future work, the heterogeneity of microservices and resource centres needs to be considered. Some users' service requirements should also be further studied. It is necessary to further improve the multi-objective algorithm to meet various users' demands.

**Acknowledgements** This work was supported by the National Natural Science Foundation of China (61773390), the Hunan Youth elite program(2018RS3081), the scientific key research project of National University of Defense Technology (ZK18-02-09, ZZKY-ZX-11-04) and the key project of 193-A11-101-03-01..

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright

holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

1. Alshuqayran N, Ali N, Evans R (2018) Towards micro service architecture recovery: an empirical study. In: IEEE international conference on software architecture (ICSA), 2018. IEEE, pp 47–4709
2. Back T, Andrikopoulos V (2018) Using a microbenchmark to compare function as a service solutions. In: European conference on service-oriented and cloud computing. Springer, pp 146–160
3. Bouzary H, Chen FF (2018) Service optimal selection and composition in cloud manufacturing: a comprehensive survey. *Int J Adv Manuf Technol* 97:795–808
4. Bouzary H, Chen FF (2019) A hybrid grey wolf optimizer algorithm with evolutionary operators for optimal QoS-aware service composition and optimal selection in cloud manufacturing. *Int J Adv Manuf Technol* 101:2771–2784
5. Cai X, Sun H, Zhang Q, Huang Y (2019) A grid weighted sum pareto local search for combinatorial multi and many-objective optimization. *IEEE Trans Syst Man Cybern* 49:3586–3598
6. Daya S, Van Duy N, Eati K, Ferreira CM, Glozic D, Gucer V, Gupta M, Joshi S, Lampkin V, Martins M (2016) Microservices from theory to practice: creating applications in IBM Bluemix using the microservices approach. IBM Redbooks
7. Deb K, Jain H (2014) An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE Trans Evolut Comput* 18:577–601
8. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: nSGA-II. *IEEE Trans Evol Comput* 6:182–197
9. Filip I-D, Pop F, Serbanescu C, Choi C (2018) Microservices scheduling model over heterogeneous cloud-edge environments as support for iot applications. *IEEE Internet Things J* 5:2672–2681
10. Hassan S, Bahsoon R, Kazman R (2019) Microservice transition and its granularity problem: a systematic mapping study. arXiv preprint [arXiv:1903.11665](https://arxiv.org/abs/1903.11665)
11. Heorhiadi V, Jamjoom HT, Rajagopalan S (2017) Failure recovery testing framework for microservice-based applications. Google Patents
12. Huang J, Li S, Duan Q, Yu R, Yu S (2017) QoS correlation-aware service composition for unified network-cloud service provisioning. In: 2016 IEEE global communications conference (GLOBECOM). IEEE, pp 1–6
13. Jamshidi P, Pahl C, Mendonça NC, Lewis J, Tilkov S (2018) Microservices: the journey so far and challenges ahead. *IEEE Softw* 35:24–35
14. Kwan A, Jacobsen H-A, Chan A, Samoojh S (2016) Microservices in the modern software world. In: Proceedings of the 26th annual international conference on computer science and software engineering. IBM Corp., pp 297–299
15. Lahmar F, Mezni H (2018) Multicloud service composition: a survey of current approaches and issues. *J Softw: Evolut Process* 30:e1947
16. Li B, Li J, Tang K, Yao X (2015) Many-objective evolutionary algorithms: a survey. *ACM Comput Surv* 48:13–30
17. Lloyd W, Ramesh S, Chinthalapati S, Ly L, Pallickara S (2018) Serverless computing: An investigation of factors influencing microservice performance. In: IEEE international conference on cloud engineering (IC2E), 2018. IEEE, pp 159–169



18. Na J, Lin K-J, Huang Z, Zhou S (2015) An evolutionary game approach on IOT service selection for balancing device energy consumption. In: 2015 IEEE 12th international conference on e-business engineering, 2015. IEEE, pp 331–338
19. Piccialli F, Benedusi P, Amato F (2018) S-InTime: a social cloud analytical service oriented system. *Future Gen Comput Syst* 80:229–241
20. Qu B, Zhu Y, Jiao Y, Wu M, Suganthan PN, Liang J (2018) A survey on multi-objective evolutionary algorithms for the solution of the environmental/economic dispatch problems. *Swarm Evolut Comput* 38:1–11
21. Ren Z, Wang W, Wu G, Gao C, Chen W, Wei J, Huang T (2018) Migrating web applications from monolithic structure to microservices architecture. In: Proceedings of the tenth Asia-Pacific symposium on internetware. ACM, pp 7–18
22. Samanta A, Li Y, Esposito F (2019) Battle of microservices: towards latency-optimal heuristic scheduling for edge computing. In: IEEE NetSoft
23. Seada H, Deb K (2015) Effect of selection operator on NSGA-III in single, multi, and many-objective optimization. In: Evolutionary computation (CEC). pp 2915–2922
24. Sharma D, Anandan R, Manikandan A, Narayanan K, Paul CS (2018) Building micro service for user engagement. *Int J Eng Technol* 7:420–422
25. Syahputra R, Soesanti I (2017) An artificial immune system algorithm approach for reconfiguring distribution network. In: AIP conference proceedings. AIP Publishing, 020017
26. Viggiano M, Terra R, Rocha H, Valente MT, Figueiredo E (2018) Microservices in practice: a survey study. arXiv preprint [arXiv :180804836](https://arxiv.org/abs/180804836)
27. Villamizar M, Garces O, Ochoa L, Castro H, Salamanca L, Verano M, Casallas R, Gil S, Valencia C, Zambrano A (2016) Infrastructure cost comparison of running web applications in the cloud using AWS lambda and monolithic and microservice architectures. In: 16th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid), 2016. IEEE, pp 179–182
28. Wang R, Zhang Q, Zhang T (2016) Decomposition-based algorithms using pareto adaptive scalarizing methods. *IEEE Trans Evol Comput* 20:821–837
29. Wang T, Li C, Yuan Y, Liu J, Adeleke IB (2019) An evolutionary game approach for manufacturing service allocation management in cloud manufacturing. *Comput Ind Eng* 133:231–240
30. Yang D, Zhang D, Zheng VW, Yu Z (2015) Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs. *IEEE Trans Syst Man Cybern Syst* 45:129–142
31. Zhang Q, Li H (2007) MOEA/D: a multiobjective evolutionary algorithm based on decomposition. *IEEE Trans Evol Comput* 11:712–731
32. Zhang Y, Tao F, Liu Y, Zhang P, Cheng Y, Zuo Y (2019) Long/short-term utility aware optimal selection of manufacturing service composition towards Industrial Internet platform. *IEEE Trans Ind Inf*
33. Zheng X, Wang L (2016) A collaborative multiobjective fruit fly optimization algorithm for the resource constrained unrelated parallel machine green scheduling problem. *IEEE Trans Syst Man Cybern: Syst* 99:1–11
34. Zhou J, Yao X (2017) Multi-population parallel self-adaptive differential artificial bee colony algorithm with application in large-scale service composition for cloud manufacturing. *Appl Soft Comput* 56:379–397
35. Zhu X (2018) Case II: micro platform, major innovation—WeChat-based ecosystem of innovation. In: China's technology innovators. Springer, pp 33–52

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.