Self-Adaptive Multi-Population Genetic Algorithms for Dynamic Resource Allocation in Shared Hosting Platforms

Azam Shirali[1,*], Javidan Kazemi Kordestani[2], Mohammad Reza Meybodi[3]

[1]*Department of Electrical, Computer and IT Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran*

[2]*Department of Electrical, Computer and IT Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran*

[3]*Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran*

azam1.shirali@gmail.com*, javidan.kazemi@gmail.com, mmeybodi@aut.ac.ir

Abstract

This paper presents a self-adaptive multi-population approach based on genetic algorithm (GA) for solving dynamic resource allocation in shared hosting platforms. The proposed method, self-adaptive multi-population genetic algorithm (SAMPGA), is a multi-population GA strategy aimed at locating and tracking optima. This approach is based on preventing populations from searching in the same areas. Two adaptations to the basic approach are then proposed to further improve its performance. The first adapted algorithm, memory-based SAMPGA, is based on using explicit memory to store promising solutions and retrieve them upon detecting change in the environment. The second adapted algorithm, immigrants-based SAMPGA, is aimed at improving the technique used by SAMPGA to maintain a sustainable level of diversity needed for quick adaptation to the environmental changes. An extensive set of experiments is conducted on a variety of dynamic resource allocation scenarios, to evaluate the performance of the proposed approach. Results are also compared with those of self-organizing random immigrants GA using three well-known performance metrics. The experimental results indicate the effectiveness of the proposed approach.

Keywords: Shared hosting platforms, Dynamic resource allocation, Multi-population genetic algorithm, Dynamic optimization problems

1. Introduction

The development in technology creates an ever-increasing need for computing infrastructures capable of efficiently response to the user's requests. On the other hand, the expansion of dedicated hosting platforms is not justifiable for economic reasons (e.g. power consumption, maintenance costs, etc.). Thus, there is a strong incentive for services to use shared resources, called shared hosting platforms (SHPs) [1]. SHPs provide massive integrated computing resources with a different computational power accessibility for a large number of services simultaneously.

Shared platforms are growing in popularity, and hosting environments are moving rapidly towards the use of such platforms with the goal of reducing costs and subsequently increasing the developers' utilization and satisfaction. Meanwhile, virtualization technologies have been employed to provide the possibility of using shared resources in the platforms [2]. Development, flexibility, shared use of resources and Internet-based presentation are the most significant features of virtualization, which plays a prominent role in facilitating the use of resources on SHPs.

One of the main challenges for the hosting platforms is resource allocation. Resources should be allocated according to the users' requests with the least possible delay and expense in order to be beneficial for providers [3]. Moreover, the resource allocation should be done in a way that the quality of service (QoS) requirements and service level agreements (made between providers and users) are met. In many cases, this leads to dynamism and uncertainty. The resource allocation mechanisms should consider the current status of each resource, and seek optimal utilization of resources by employing algorithms for better allocation of physical and virtual resources to the users [4].

We have considered resource allocation for dynamic servers similar to the problem of resource allocation in the real world. Failure to access a number of servers during the allocation process is among the issues that make the allocation problem in a SHP an uncertain and dynamic problem. In the course of the allocation process, a number of servers may be out of reach and not available for a period of time due to hardware and software problems, low battery, network failure, lack of routes, etc. In such a situation, optimal and efficient resource allocation would be an uncertain optimization problem. Although the optimality is specified and definite, it undergoes changes over time, and hence the optimization algorithm should be able to track the optimum of the problem with the maximum possible accuracy.

The problem of resource allocation is an NP-complete problem and the exact solution, even for small examples of the problem (a small number of servers and services), cannot quickly be found [5]. Thus, many studies have been conducted on the resource allocation problem (RAP) using greedy algorithms or metaheuristic techniques including evolutionary algorithms, swarm intelligence algorithms, etc. In this paper, we define the dynamic resource allocation problem (DRAP) for SHPs as a constrained dynamic optimization problem (DOP), and attempt to solve this problem within a reasonable elapsed time by suggesting algorithms based on a genetic algorithm (GA).

The rest of the paper is organized as follows: Section 2 discusses related works. System description, problem definition and problem formulation are described in Section 3. In Section 4, we explain our proposed self-adaptive GAs for dynamic resource allocation in SHPs. An experimental study for analyzing the impact of the proposed approach is presented in Section 5. Finally, conclusions and directions for future research are discussed in Section 6.

2. Related works

The existing policies for allocating resources in hosting platforms fall into three main categories. The first category is system oriented and the goal is to reduce the response time of the whole system [6], increase the throughput of the whole system, balance the workload [7][8], enhance the system reliability, increase the utilization [6][7][8][9][10][11][12], reduce the energy consumption [13][14][11][15] or some combination of these. The second category is user oriented and the goal is to optimize efficiency, maintain QoS [1][16], increase the fairness between services [8] or a combination of these for each service. The third category includes algorithms with the purpose of combining the first and second categories. The studies undertaken so far have attempted to investigate and improve one or more issues related to resource allocation challenges.

Our resources allocation problem considers the dynamic nature of the servers and hosts the services with the aim of increasing the minimum yield for services that the yield is based on the amount of resources allocated to the service. Following the increase in yield, increased fairness between services during resource allocation as well as increased performance and utilization of resources during the allocation process is regarded. In fact, we have considered a combination of system-oriented and user-oriented goals.

Many systems have been recommended for resource allocation in static environments [1][16][17][18][19] and dynamic environments [20][6][7][8][13][14][21][12] for SHPs. To the best of our knowledge, the dynamic element of the environment is included in the workload in most of these studies and the environment is static from other perspectives. Although we have considered resource allocation for a static workload, our RAP has dynamic servers (as a results of network issues, hardware and software problems, etc.) similar to the problem of resource allocation in the real world. In [20], resource allocation delays the response to the request for as long as possible to deal with the dynamic and unpredictable workload. The overhead is reduced as much as possible by this policy, and consequently decreases the failure rate (reduces the problem instances for which the algorithm cannot find any allocation).

We have considered resource allocation for an arbitrary number and various dimensions of the resources, however, in [16][8][20] resource allocation was considered for the CPU only. In other hand, we have divided the resources into the two categories that is similar to [7]. Resources are classified into load-dependent and load-independent categories, and each category is allocated accordingly.

The issue of QoS and service level agreements considered as criteria for resource allocation was addressed in several studies. In our paper, the resources are allocated by maintaining QoS requirements, allocating resources based on the best effort for services without QoS requirements and presenting an allocation undertaking that there will be no need for migration. In [1], a shared cluster is employed to book resources intelligently, monitor the behavior and condition of the services with regard to resource use by applying various techniques and ensure maximum profits for providers according to the services status and their QoS requirements. We also consider QoS requirements for allocation in a dynamic environment, while [1] considers the case of a static environment. In [16], in a static environment, requests with the same resource demands are categorized in the same classes, and then the allocation is performed for these classes, which leads to the minimum possible use of resources. The objective function is a linear problem optimizing the average deviation between the share of the resource allocated to the service and its desired resource share. However, this allocation exhibits unfair behavior among services. In the second stage of optimization, this paper considers a bias for maximizing the least yield by using a quadratic objective function [22] to increase fairness. Fairer behavior is produced by maximizing the least yield among the services. Unlike [16], we encourage a fair allocation by maximizing the minimum service satisfaction across service requests.

One of the goals that led to the popularity of SHPs was the increase in utilization of computing resources. We also define the utilization increase based on yield that the yield is based on the amount of resources allocated to the service. In [6], the utilization is increased by achieving the optimal response time for services, where the resource allocation is performed based on optimizing the service-yield average which is based on the response time of the services. In [7][8], the increase of utilization has been considered as the criterion for resource allocation in SHPs. These researchers limit the system disturbances at the start and end of a service using a dynamic placement, and respond to further requests by reducing placement changes on the one hand and providing the same workload on all the servers as far as possible on the other hand. In [8], as in our work, fairness among services is considered, with the aim of offering the same level of satisfaction among services. Furthermore, in [12], optimum utilization machines have been obtained by employing the parallel GA, thus expediting the search for the best placement. In the scheduling model of this system, the lists of idle resources and virtual machine (VM) requests are updated after any changes, and then physical resources are allocated to the VM requests.

Some studies aim at increasing utilization during resource allocation by reducing the energy consumption. For example, [14] offers a dynamic resource allocation in the form of an energy-aware approach, based on the costs for the data center. The proposed system examines performance periodically to predict future resource requirements and enables the data center to estimate costs by balancing future needs and available resources and to provide a proper price for customers. In [13], a balance is created between demand capacity and energy consumption via migration of overloaded servers' workloads to servers with a lower workload, or by turning off the servers with low workloads.

In other studies [9][10] the issue of static and dynamic resource allocation has been investigated with the aim of reducing the number of servers required and energy consumption to respond to requests. In [9], a GA and linear programming utilize resource needs as inputs and use the lowest possible number of servers that can provide a specific computing time and reduce the energy consumption.

Due to their adaptive nature, evolutionary algorithms (EAs) have proven to be good optimizers for DOPs [23]. Over the past years, various attempts have been done by researchers to improve the efficiency of traditional EAs on DOPs.

One way of addressing DOPs is to simply restart EAs from scratch upon detecting change in the environment. However, it is more sophisticated to develop other approaches that make use of information gathered from previous environments. Early EAs for solving DOPs endeavored to increase population diversity after detecting a change in the environment. For example, Cob [24] incorporated a hyper-mutation operator into a GA which temporarily creates a dramatic increase in the mutation rate after a change occurs. Vavak et al. [25] [26] also suggested a variable local search for controlling the mutation size.

Another approach for dealing with DOPs is to maintain a sustainable level of diversity during the entire optimization process. For example, Grefenstette [27] proposed the random immigrants GA, where at each generation a part of the population is replaced by randomly generated individuals. Other examples of mechanisms for maintaining diversity are fitness sharing [28] and sentinel placement [29].

Apart from the above approaches, many researchers have equipped EAs with a memory scheme that restores useful information about previously found positions in the search space. This technique is especially useful in periodical or recurrent environments, where the optima may repeatedly reappear in regions near to their previous positions [23].

A group of studies has distributed the individuals of EAs among multiple populations where each population is responsible for handling a separate area of the search space. SOS [30], shifting balance GA [31] and multinational GA [32] are three well-known examples of multi-population GAs for DOPs. In this paper, we discuss the implementation and application of the self-adaptive multi-population GAs to solve the DRAP for SHPs.

3. System Description

A SHP includes servers that are connected to each other and provide shared resources among the services. The challenge is to allocate resources appropriately and collectively, and different studies pursue different goals with respect to the allocation process. In this paper, the allocation of resources is such that it can provide QoS, provide fairness between services, increase the utilization of the platform and maximize the utility functions of the service. In the following, the resource allocation system is

defined and implemented in the hosting platforms in order to try to optimize the allocation problem and achieve the above goals.

3.1 Problem Definition

A resource allocation system controls how hosting resources are shared between different services. Each service includes one or more VM instances, and the resource allocation system is responsible for sending the services to desirable servers. This system allocates resources accurately and fairly between VMs through VM monitors [33]. VM instances can migrate between servers when needed, but this is not within the scope of this research, and here we assume that VMs do not migrate between servers. In this system, the VM technology makes it possible to accurately share resources, including memory, disk space, CPU, etc., between VM instances. The assumptions and limitations for this resource allocation system are as follows:

- Shared servers in hosting platforms are considered to be homogeneous, meaning that all servers share the same resources in terms of type and capacity.
- The workload of services is considered to be static, meaning that each service requires a fixed and definite resource and the amount of requested resource does not change during the allocation process.
- Each service includes just one instance of a VM; services with multiple VMs are not within the scope of this research.
- Migration of VM instances between servers is not in the scope of this research, and therefore we assume that VMs do not migrate between servers.
- The shared resources available on the servers vary, and we have considered two to six resources in various scenarios.

3.2 Services and Resources

Each server will share several resources to respond to the services in our proposed resource allocation system. We have divided these resources into the two categories of rigid resources and fluid resources. Therefore, the needs are also divided into fluid needs and rigid needs.

When a rigid need requests a certain amount of resource, the service cannot benefit from allocating a larger amount of resources, nor can it start by allocating a smaller portion of the resource of the response to the request. However, if a fluid need requests a specific amount of resource, this is the largest part of the resource that can be used if the service is on a single server. Such a service cannot benefit from allocating a larger amount of resources, but it can start by allocating a smaller portion of the requested resource to the request, although the performance level will decrease.

The following example from [19] clarifies this point. We assume that a service has two fluid needs, one of which requires 40% of the server's input/output bandwidth, while the other requires 60% of the

server processor. In this example, this service cannot fully benefit from both resources due to the interconnection between input/output and computing.

Our allocation system calculates the yield for the fluid needs of each service. This yield is based on the ratio of the amount of resource allocated to the relevant service and the maximum amount of the requested fluid resource. For example, if a fluid need, such as a CPU need, is 60%, but only 42% of the processor is allocated, then the yield will be equal to $42 \div 60 = 0.7$.

We assume that the yield of all resources corresponding to fluid needs in a service is linearly related in this allocation system. This means that for the above example, if only 20% of the input/output bandwidth is allocated (half the amount that could potentially have been used), then only 30% of the CPU can be used (this is also half the amount that could potentially be used - because of interdependence between I/O and computation) [19]. In this case, if the percentage of the fluid resource (such as the CPU) is up to half the requested percentage, thus consumption of other resources will be halved. In general, we calculate the yield on all fluid needs in a service and refer to it as the service's yield.

The obtained yield of a service will take a value between zero and unity, based on the above definitions, where zero corresponds to the case where no resource has been allocated to the related service, although the allocation system sets the lower limit for the service yield based on QoS requirements. In the previous example, if the minimum yield of the service is limited to 0.4 in order to meet QoS requirements, this means that the allocated CPU portion of the need must be at least 24% ($0.4 \times 60\% = 24\%$), and responding to the need will not be possible with less than this proportion of the processor. Fluid needs of services that have QoS requirements are called "constrained fluid needs". Fluid resources in a best-effort model are allocated to services that do not have any QoS requirements.

We have assumed that the rigid needs are completely independent of the fluid needs in order to simplify the allocation process in this resource allocation system. The question arises as to how the exact amount of these allocated resources is determined. Certainly, a SHP can host only a limited number of services at one time. The reasoning behind this research is based on the resources allocated to the services and optimization of their calculated yield.

We reason that if all the resources allocated from each source to the services are optimal, then it can be said that almost all the resource-management goals are optimally addressed. Our proposed allocation system allocates the resources based on the QoS requirements of the services, and it uses the best-effort model when a service does not have any QoS requirements. In other words, the minimum required fluid resources are allocated based on the QoS requirements of each service initially, and parts without the use of resources are distributed equitably according to the QoS requirements of the services after allocating the minimum requirements. Generally, this resource allocation system works by increasing the lowest yield for services where maintaining the QoS is important, and allocating resources based on best effort for services that do not have any QoS requirements. Following this increase in yield,

providing greater fairness among services during the allocation of resources is considered, as well as increasing the performance and utility of resources during the allocation process. Since this method is independent of a specific purpose and addresses the goals of resource management generally, it can be used for many issues. For example, [34] provides an effective method for the parallel work-scheduling problem by optimizing the yield.

3.3 Goals and Constraints

According to the definition in the previous section, the yield of a service is defined between a certain positive minimum value and a value of unity. In cases where the yield is less than the minimum value, the resource allocation procedure fails. Informally, the yield is determined by the quality satisfaction of a service. Because services hosted by these platforms have different QoS requirements, they will have different minimum required yields. Therefore, for comparing service performance among services, we define the scaled yield:

$$Scaled\ Yield = \frac{yield - Minimum\ yield}{1 - Minimum\ yield} \quad (1)$$

For example, consider two services with minimum required yields of 0.2 and 0.4, where the yield is 0.8 for the first service and 0.85 for the second service. The scaled yield of both services will be 0.75. In other words, each service has a yield of 75% between the lowest and highest yields. For a service with a best-effort allocation model, the lowest yield is equal to zero and the scaled yield equals the actual yield.

Eq. (1) is not defined for the situation where the lowest yield is equal to unity. Such a service is always satisfactory, and its scaled yield is defined as unity in a valid and correct allocation, and otherwise is zero. If the scaled yield value of a service is negative, then the resource allocation fails. Moreover, if a rigid need encounters no response for a service, that resource allocation will fail. In the remainder of this article, the term 'yield' should be understood as the scaled yield, unless it is explicitly stated otherwise.

3.4 Problem Formulation

In the following description of the allocation problem, we formulate and consider our RAP as a linear program:

We consider a number of services N (N > 0) with i = 1, ..., N. The SHP includes H homogeneous physical servers (H > 0) defined as h = 1, ..., H. For each accessible server, there are d types of resources, identified as j = 1, ..., d.

For each service i:

- r_{ij} is a number between 0 and 1 that specifies the need of service i to resource j.
- δ_{ij} is a binary value, considered to be 1 if r_{ij} is a rigid need and 0 if it is a fluid need.

- $\hat{y}_i$ is a number between 0 and 1 that specifies the minimum required yield of the service requirement based on the QoS requirements of that service.

The linear program of the problem will be as follows based on the variables we have defined, where e_{ih} is a binary variable equal to 1 if service i runs on server h, otherwise it is 0. We specify the non-scaled yield of service i on server h with the variable y_{ih}, which is equal to 0 if the service is not implemented on server h. Y is equal to the achieved minimum yield over all services. The limitations of the linear program are as follows, taking into account these definitions:

$$\forall\, i,h \qquad e_{ih} \in \{0\,,\,1\} \quad , \quad y_{ih} \in Q \quad , \quad 0 \leq y_{ih} \leq 1 \tag{2}$$

We specify the scope and domain of the variables with these limitations where $0 \leq Q \leq 1$ is a rational number.

$$\forall\, i \qquad \sum_h e_{ih} = 1 \tag{3}$$

As previously mentioned, a service is completely encapsulated in a VM and runs only on one accessible server. Eq. (3) specifies this limitation.

$$\forall\, i,h \qquad 0 \leq y_{ih} \leq e_{ih} \tag{4}$$

Eq. (4) describes the fact that a service can obtain a non-scaled yield significantly greater than zero only on the accessible server on which it is executed.

$$\forall\, i \qquad \sum_h y_{ih} \geq \hat{y}_i \tag{5}$$

We show that a service must have a yield greater than or equal to its minimum required yield based on limitation (5). Of course, only one of these terms is non-zero.

$$\forall\, h,j \qquad \sum_i r_{ij}\left(y_{ih}\left(1 - \delta_{ij}\right) + e_{ih}\delta_{ij}\right) \leq 1 \tag{6}$$

Based on limitation (6), we show that the portion of resource j on server h that has been allocated to services has, at most, a value of unity, i.e., when that resource is fully allocated to the corresponding service. In fact, if r_{ij} is a fluid need, then $\delta_{ij}=0$ and the part of resource j used on server h is $r_{ij} \times y_{ih}$ (the maximum usable part multiplied with the yield). On the other hand, if r_{ij} is rigid need, then $\delta_{ij}=1$ and the part of resource j which is used on server h is r_{ij}.

$$\forall\, i \qquad \sum_h y_{ih} \geq \hat{y}_i + Y\left(1 - \hat{y}_i\right) \tag{7}$$

Finally, we determine that the minimum yield over all services Y is no greater than the yield of any other services based on limitation (7).

The current RAP is an NP-complete decision problem. For example, if two servers are considered, each of which supplies only one resource equally, and all services need only a rigid resource, the RAP is divided into two parts, each of which is NP-complete [5]. This result would be obtained even if all services needed only a fluid resource and no service needed the least yield with respect to the QoS requirements, i.e., it would still be an NP-complete problem [35].

The solution presented as a linear program to solve the problem in section 3, provides an exact solution, but one which is performed over an exponential time period. This exact solution takes a long time even for small problem instances (a small number of servers and services). Therefore, we present three developed algorithms based on a multi-population GA for dynamic resource allocation in SHPs from the literature. Lack of access to some servers during the allocation process creates a dynamic and uncertain environment.

4. Proposed Self-Adaptive Multi-Population Genetic Algorithms

In this section, first we describe the outline of our basic approach in detail. Two adaptations to the basic approach are then proposed to further enhance its performance by adding memory and random immigrant.

4.1. Self-Adaptive Multi-Population Genetic Algorithm

In the following, a self-adaptive multi-population GA (SAMPGA) is presented to sovle the DRAP. The proposed method is similar in many ways to the one discussed by Blackwell in [36]. The pseudo code for the proposed SAMPGA is shown in Figure 1.

Algorithm 1. Self-adaptive multi-population genetic algorithm (SAMPGA)

```
01: t := 0 and initialize the first population P_1(0) randomly;
02: Evaluate population P_1(0);
03: repeat
04:   if M_free = 0 then
05:     Generate a new free population;
06:   else if M_free > n_excess then
07:     Remove the worst free population;
08:   end-if
09:   for each population n do
10:     if population n is excluded then
11:       Randomize n;
12:     else
13:       P'_n(t) := selectForReproduction (P_n(t));
14:       Crossover (P'_n(t), p_c); //p_c is the crossover probability
15:       Mutate (P'_n(t), p_m); //p_m is the mutation probability
16:       Evaluate the interim population P'_n(t);
17:       P_n(t + 1) := P'_n(t);
18:     end-if
19:   end-for
```

```
20:  Test for change;
21:  for each population n do
22:  |  Test for convergence;
23:  end-for
24:  Apply exclusion;
25:  t := t + 1;
26: until the termination condition is met // e.g., t > tmax
```

Figure 1. Pseudo code for the self-adaptive multi-population genetic algorithm (SAMPGA)

4.1.1. Genetic Algorithm

This sub-section describes the key components, i.e. genetic representation, population initialization, fitness function, selection scheme, crossover, and mutation, for designing of the GA for the dynamic resource allocation in SHP.

A. Genetic Representation

In the proposed SAMPGA, each individual I is represented by a vector of positive integers $\vec{I} = \{s_1^I, s_2^I, \dots, s_N^I\}$ for each service $i = 1,2, \dots, N$. Hence, if service i is allocated to server h, then the s_i^I is set to h.

B. Population Initialization

In SAMPGA each individual corresponds to a potential solution to the DRAP. The initial population $P_1 = \{\vec{I}_1, \vec{I}_2, \dots, \vec{I}_m\}$ is composed of a certain number of individuals. For each individual, the allocation of each service to a server is randomly generated.

C. Fitness Function

The resource allocation system is formulated to maximize the least yield among all services. The objective function of our problem, denoted as Y, is given by:

$$Y = \min_{i \in N} \left(min_{j \in \{1,\dots,d\}} \frac{r_{ij}(\hat{y}_i(1-\delta_{ij})+\delta_{ij})}{(1-\hat{y}_i)r_{ij}(1-\delta_{ij})} \right) \quad (8)$$

Where j is the set of indices such that $(1 - \hat{y}_i)r_{ij}(1 - \delta_{ij})$ is non-zero.

In order to calculate the fitness of a feasible resource allocation we use the above Y and we define $f = H \cdot (1 + Y)$. In other words, to calculate the fitness, the least achieved yield over all services is multiplied by the number of available servers and added to the same number of available servers. Therefore, the fitness value for feasible chromosomes can range between H and 2H.

D. Selection Scheme

Selection is one of the most important components of the GA which is used to improve the average quality of the population by giving more chance to the higher-quality chromosomes for being exploited

in the next generation. There is a wide range of selection techniques in the literature. In this paper, we use the tournament selection and the tournament selection size is 5.

E. Crossover and Mutation

In this work, we adopt a one-point crossover to exchange partial chromosemes, i.e. sub-allocations, between two chromosomes. Given two parent chromosomes, the one-point crossover works as follows: (*a*) randomly chosen crossover point is selected, and (*b*) the contents of the chromosomes beyond the chosen point are swapped.

After the crossover operation is completed, the population will undergo the mutation operation. The mutation operator randomly moves two services between two different servers.

E. Repair operator

Each newly generated chromosome may be an infeasible solution (resource allocation). In fact, some servers may be overloaded and unable to respond to a service. In this case, a repair cycle is used to convert an infeasible solution into a feasible one. To this end, after an infeasible solution is generated, the algorithm uses a greedy approach to redirect the services from overloaded servers to less-loaded ones. The fitness value of an infeasible allocation is equal to the number of allocated servers that are not overloaded during the service. Therefore, it can range from 0 to H, where H is equal to the number of available servers.

4.1.2. Creating and Removing a Population

The SAMPGA starts with a single population and adaptively determines the number of populations either by creating new populations into the search space, or by removing redundant populations.

In the proposed algorithm, populations are categorized into two groups: (1) *free* populations, whose expansion, i.e. the distance between its farthest individuals, is larger than a predefined radius r_{conv}, and (2) *converged* populations. Once the expansion of a free population becomes smaller than a radius r_{conv}, it is converted to a converged population. In SAMPGA, when the number of free populations (M_{free}) drops to zero, a free population is initialized in the search space for locating undetected or new emerging optima. On the other hand, free populations are removed from the search space if M_{free} is higher than a threshold n_{excess}. The mentioned process can be summarized as follows:

$$M(0) = 1$$
$$M(t) = \begin{cases} M(t-1) + 1\ , & M_{free} = 0 \quad or \quad M_{free} < n_{excess} \\ M(t-1) - 1\ , & M_{free} > n_{excess} \end{cases} \tag{9}$$

where $M(t)$ is the number of populations at time step t, and n_{excess} is the maximum number of allowed free populations.

4.1.3. Check the status of the populations

As stated earlier, the status of each population is determined by the radius r_{conv}. In the SAMPGA, we define a radius called the convergence radius (r_{conv}), with the goal of improving local searches in promising areas and reducing the loss of objective function evaluations. A population in which the distance between the farthest individuals of its population is less than this radius is called a convergent population, and it remains in the next iteration and continues to search in its relevant area. However, a population in which the distance between the farthest individuals is greater than this radius is called a free population and we treat it in the next iteration based on Eq. (9). To calculate this radius, Eq. (10) is used:

$$r_{conv} = \frac{X}{\alpha M} \tag{10}$$

Here, X is the range of the search space which is calculated using Hamming distance. As explained earlier, the length of each chromosome is equal to the number of services, i.e. each gene represents a specific service. The maximum Hamming distance between two farthest solution occurs when all of their genes are different, which is equal to the number of services. M is the number of current populations.

This strategy ensures that at least one free population is exploring the search space with the aim of finding new optima.

4.1.4. Overlapping and redundant search

One of the key challenging task of addressing DOPs using multi-population approach is to keep each population on a distinct sub-area in the fitness landscape. On the other hand, several studies have confirmed that overcrowding of populations in the same sub-area of the search space is not effective. Most researchers have been using exclusion, which was proposed by Blackwell and Branke [37] over continuous search space, to differ the search areas covered by the populations and preventing them from redundant search.

In this paper, we also used the concept of exclusion to differ the search territory of populations and avoid redundant search. To this end, we define an exclusion radius r_{excl} to check the overlap between every pair of populations. Once the distance between the best individuals of two populations becomes smaller than the threshold r_{excl}, exclusion operator triggers and randomely re-initializes the weaker population, the one with lower fitness value, in the search space. The radius r_{excl} is computed as follows:

$$r_{excl} = \frac{X\alpha}{M} \quad (11)$$

where X is the range of the search space. Here, we use the Manhattan distance to calculate X which is equal to $(length\ of\ the\ chromosome) \times (number\ of\ server - 1)$. M is the number of current populations.

4.1.5. Detection and response to changes

The RAP considered in this paper is dynamic in the same way as in the real world, meaning that yield of service changes over time as a results of network issues, server failure, etc. In other words, some servers may become unreachable during the resource allocation process. Threfore, two types of change occur in the considered DRAP:

I. Once a server becomes unreachable. In this case, the resource allocation should be performed without taking the corresponding server into account.

II. Once a server becomes available after being unresponsive. In this case, the corresponding server may be able to increase the yield of the system and must be added to the resource allocation system for hosting the services.

Therefore, the proposed model should be able to properly handle the above mentioned changes. In order to handle type I changes, the first step is to detect such changes. Several methods have been suggested for detecting changes in dynamic environments. See for example [23] for a good survey on the topic. The most widely used method consists in reevaluate memories of the algorithm for detecting inconsistencies in their corresponding fitness values. In this work, we considered the best solution of the algorithm. At each iteration, the algorithm reevaluates the best-found solution and compares its current fitness value against that from the previous iteration. Thus, if an inconsistency is found from this comparison, then we will assume that the environment has changed. Once a type I change is detected, all individuals are re-evaluated. The type II changes are known to the algorithm. In this case, we should only re-evaluate the global best solution to see if the new available server(s) could improve the performance of the resource allocation process.

4.2. Memory-Based SAMPGA

The idea of adding memory to evolutionary algorithms has attracted the attention of many authors. When an optimum is detected again in a previous position, the memory can recall that position and immediately move the population towards this new optimum. Additionally, memory can be useful for maintaining diversity and guiding the algorithm to promising areas after re-initialization. On the other hand, although additional memory may use information from the past, this may mislead the evolving process, preventing the exploration of new areas and the discovery of new optima.

In the process of dynamic resource allocation, it is possible to encounter repetitive environments in SHPs. Access to, and lack of access to, servers, may occur at any time, and if we store information from the environment before a server leaves, that information can be very useful when the server returns to the hosting platform. The allocation process improves its performance by using knowledge saved from the environment, and the result is more optimal than when repeating solutions are not considered. Therefore, we use memory enhancement in the SAMPGA and call it the memory-based self-adaptive multi-population GA (MSAMPGA).

Before adding memory to the algorithm, some questions should be answered. First, should the memory be implicit or explicit? Implicit memory-based methods in dynamic environments are based on the storage of additional information from individuals to be used in the future. However, in explicit memory methods, a separate memory with a specific capacity is used, so that some individuals (usually the best individuals) is stored from the current generation and reused in subsequent generations or environments. In this method, if there is a certain similarity between the current environment and the previous environment, previous appropriate solutions are used and the performance of the algorithm is improved.

If explicit memory is used, then the following three questions must be answered. First, which individuals will be stored in each memory storage event? Second, when the memory is filled, what strategy or strategies can be used for replacements? Third, how are individuals retrieved from memory and used? Answering to these questions is very difficult, and various strategies have been suggested in different studies. Reference [38] gives further details and comparisons of these strategies.

We used explicit memory in MSAMPGA in order to develop the SAMPGA. Briefly, MSAMPGA operates as follows:

(i) At the commencement of the run, allow the SAMPGA to run in its standard manner.

(ii) If a new free population is created:
- Add the best individual in the new population to the memory.
- Identify the best individual among all the populations and replace instead of the nearest individual in the memory (If the best individual among all the populations is in the same new population, it is clear that these two steps are combined and only the best individual from the new population is added to the memory.).

(iii) If a free population is removed:
- Delete the worst individual from the memory.

(iv) On the condition that no environmental change has been detected, return to step (i).

(v) Upon detecting a change in the environment, go to step (vi).

(vi) Re-evaluate the individuals in the memory.

(vii) For all the individuals in the memory, specify the server that has the highest workload and replace by the server that has the lowest workload.

(viii) Replace the worst individual of the populations by the individuals in the memory, provided that they have more fitness.

After retrieving the individuals from the memory, the individuals in the memory remain unchanged. The number of individuals in the memory is always the same as the total number of populations with these replacement, removal and retrieval strategies in the memory. After each change, the individuals stored in the memory have some useful information about current environment, adding an individual from the memory to each of the populations will help the population to achieve earlier convergence and faster adaptation to environmental changes.

In the studies to date, replacement strategies for the most similar and the nearest individuals in the memory have been used. We also use the replacement strategy of the nearest individual in the memory in the MSAMPGA. We use the Manhattan distance to calculate how near the individuals are to each other.

When a change occurs in the process of resource allocation to hosted services, after the individuals in the memory are re-evaluated, a server that has a greater workload reallocates the hosting of one of its services to a server that has a lesser workload. The reason for doing this is that it is usually the service with the lowest yield among all the services that will be hosted on the server with the highest workload. Since the main goal of our DRAP is to maximize the lowest yield, the chance of a higher yield for this service will be increased by moving this service to the server with the lowest workload. In this way, and using the knowledge available in the environment, the memory has a much greater effect in increasing the yield.

4.3. Immigrants-Based SAMPGA

As stated earlier, convergence is the most important challenge in dynamic environments which vary with time. Many methods have been suggested to prevent premature convergence, addressing this problem by preserving or creating diversity in the population. One of the effective methods is the random immigrant GA (RIGA) developed by Grefenstette in 1992 [27] with the goal of maintaining diversity during the runtime in dynamic environments. In RIGA, a fraction of the current population is replaced by new random individuals in each generation of the run. The individuals replaced may be random or may be the worst individuals i in the population. If the number of genes in an individual is high and the local optimum has a fitness value greater than the mean fitness of all possible solutions of the search space, the probability of survival of new random individuals is very low in general, because the selection methods used in the GA select the best solutions directly or indirectly and the probability that the fitness of new random individuals is higher than the current individuals is very small. Therefore, RIGA suggests the above method of replacing random individuals with the aim of preserving the diversity of populations.

However, in an environment with small changes, the RIGA may be distorted before the change takes place, leading to a decrease in performance. If the intensity of environmental changes is not high, RIGA may not have much effect on performance even when the change occurs, because the former population may still be good and suitable for the new environment. For this reason, the immigrant method in the GA was introduced in order to be used in an elitism-based immigrants algorithm (EIGA), which uses an individual replacement strategy based on the substitution of the elite of the previous iteration.

In EIGA, in each iteration, after combination and selection according to the standard GA, the elite of the previous iteration are used as the basis for migration, such that the elite are mutated several times with a specific probability and become immigrants to replace the worst individuals in the current population. Using the elite of the former population in order to produce immigrants keeps them close to the current population, which is essential for increasing the performance of the GA in a dynamic environment [39].

Here, with the help of the unique properties of the immigrants, we use an improved elitist method in the SAMPGA, which we call the immigrants-based self-adaptive multi-population GA (ISAMPGA).

We have tried to solve and run the problem completely realistically in allocating the considered dynamic resource. Since the number of non-accessible servers varies at each change in the environment, the intensity of the change varies over the runtime and the allocation process includes low, moderate and severe changes. With this in mind, the ISAMPGA is designed and operates as follows:

(i) At the commencement of the run, allow the SAMPGA to run in its standard manner.
(ii) Identify the population with the best individual among all populations and replace its worst individual by random immigrant.
(iii) For each free population, replace the worst individual by an immigrant created based on the previous elite.
(iv) On the condition that no environmental change has been detected, return to step (i).
(v) Upon detecting a change in the environment, go to step (vi).
(vi) For each convergent population, replace the worst individual by a random immigrant.
(vii) For each free population, replace the worst individual by an immigrant created based on the elite before the environment change.

The strategy we used in designing this algorithm is to achieve good performance when faced with any kind of change. The use of random immigrants and elitism-based immigrants covers both low-intensity and extreme-intensity changes. When the environmental changes are small, the elitism-based immigrant maintains the algorithm close to the environment and provides more adaptability to the

environment. An elitism-based immigrant will be produced by creating a mutation in the elite of the previous iteration. The mutation operator used in this algorithm is a single point.

Moreover, the use of a randomized immigrant for the best population in each iteration leads to escape from local optima, and wasted objective function evaluations are prevented by more efficient searching. One of the difficulties that algorithms face after an environment change is convergence. For an algorithm that converges, maintaining diversity and dispersion in the environment after a change in the environment takes a great deal of time and results in a significant drop in performance. Therefore, in the ISAMPGA, when changes occur in the environment, convergent populations will replace their worst individual by a random immigrant to increase diversity in the population and to give more effective coverage of the search space.

5. Experimental study

5.1. Experimental Settings

In order to evaluate the performance of our algorithms on dynamic resource allocation in SHPs, various simulations are conducted on different resource allocation scenarios. Each scenario is composed of a combination of the number of services N, the number of requested resources, the QoS requirements ρ, as well as the rigid and fluid needs. The following settings were adopted for all scenarios:

- In all scenarios, the hosting platform has 64 homogeneous servers ($H = 64$).
- Experiments are performed for a different number of services $N \in \{100, 200, 500\}$.
- Each service has a d-dimensional resource need, where $d \in \{2, 4, 6\}$ is equal to an even number with $\frac{d}{2}$ fluid needs and $\frac{d}{2}$ rigid needs. For example, a scenario with $d = 4$ has 2 fluid needs and 2 rigid needs.
- The variable $\rho \in \{0.00, 0.25, 0.50\}$ represents the QoS requirement.
- Needs for each service are generated by a normal distribution. Normal distributions with mean $\mu = 0.1$ and standard deviation $\sigma = 0.05$ were used for fluid needs and normal distributions with mean $\mu = 0.5$ and standard deviations $\sigma = 0.25$, 0.5 and 1 were used for rigid needs in different scenarios.

Thus, 81 different scenarios (1×3×3×3×1×3) are generated, which will be used in the experiments. The data sets relating to each of these scenarios are stored in files and used for all experiments in order to make a more accurate and fairer comparison between the algorithms.

All the algorithms used similar random seeds for starting each run. Moreover, the occurrence time of change is random and the number of changes is between 90 and 120 changes per run of the algorithm. As the occurrence time of environmental changes during the resource allocation is unpredictable in the real world, the changes are assumed to be random and probabilistic in the experiments. In fact, the server's exit time and return time, and the time the server is out of reach are all completely random. In

the experiments, all conditions are the same for all algorithms in order to achieve a fairer comparison of the algorithms' performances. All aspects of changes are also the same for all algorithms.
Unless stated otherwise, the default values for the parameters of the proposed approach are according to Table 1.

For each run of an algorithm 500,000 fitness evaluations is considered as the termination condition. For each experiment of an algorithm, 50 independent runs are executed each with a different random seeds. Finally, all the experiments were run on a 64-bit OS x64-based processor with Intel Core i7-5500U CPU at 2.40 GHz and 8.0 GB RAM.

Table 1. Default parameter settings for the proposed self-adaptive multi-population GAs

Parameter	Default value
Number of chromosomes in each population	10
Maximum number of free populations n_{excess}	7
α	2.5
Crossover probability P_c	0.25
Mutation probability P_m	0.1
Tournament selection size	5

5.2. Comparison metrics

For measuring the efficiency of the tested algorithms, we considered the offline performance, best error before change and offline error which are well-known metrics for dynamic environments [23].

5.2.1. Offline Performance

Offline performance is computed as the average of the best values found so far at each time step:

$$P_{off}=\frac{1}{T}\sum_{t=1}^{T} e_t^* \quad (12)$$

where $e_t^*=\max\{e_1, e_2,\ldots, e_t\}$, e_t is the t^{th} evaluation and T is the maximum number of function evaluations. This measurement is problematic since it has to be ensured that the best solution found so far is the best solution for the current environment [40].

5.2.2. Best Error Before Change

The other measure, which was first proposed by [41] as the accuracy method and later named best error before the change [23], is calculated as the average of the minimum fitness error achieved by the algorithm at the end of each environment, i.e., just before a new change:

$$bbc = \frac{1}{K}\sum_{k=1}^{K}(h_k - f_k) \quad (13)$$

where f_k is the fitness value of the best solution obtained by the algorithm before the kth change occurs, h_k is the optimum value of the kth environment and K is the total number of environments. In this study, we use both measures to evaluate the performance of the proposed algorithms. For more detail on the difference between the two measures mentioned, interested readers are referred to [42].

5.2.3. Offline Error

The last measure is the performance measure suggested by [40] which is defined as the average of the smallest error found by the algorithm in every time step:

$$E_{off} = \frac{1}{T}\sum_{t=1}^{T} e_t^* \quad (14)$$

where T is the maximum number of fitness evaluations so far and e_t^* is the minimum error obtained by the algorithm at time step t. We assume that a function evaluation made by the algorithm corresponds to a single time step.

5.3. Sensitivity analysis of the proposed methods

In this sub-section, effects of key parameters (i.e., α, population size and the number of free populations n_{excess}) and components (i.e. distance measure) on the performance of the proposed SAMPGA were analyzed on different scenarios.

5.3.1. Effect of different distance measures for convergence radius and exclusion radius, and parameter α

As described in Section 4, the convergence radius (r_{conv}) and the exclusion radius (r_{excl}) are two critical parameters of the proposed method. In order to calculate these parameters, we should first focus on solving one important challenge related to the status of the populations and overlapping among populations: how to suitably measure distance between two candidate solutions in the search space.

Different distance measures can be used to calculate the distance between two candidate solution in discrete problem space. In this work, we used the Hamming distance and the Manhattan distance. Table 2 and Table 3 show different combinations of distance measures for calculating r_{conv} and r_{excl}. Four possible combonations were used as follows:

- $r_{conv}: Hamming, r_{excl}: Manhattan$
- $r_{conv}: Hamming, r_{excl}: Hamming$
- $r_{conv}: Manhattan, r_{excl}: Manhattan$
- $r_{conv}: Manhattan, r_{excl}: Manhattan$

Table 2. Best error before change of SAMPGA for α = 2.5 with different measuring modes for r_{conv} and r_{excl}

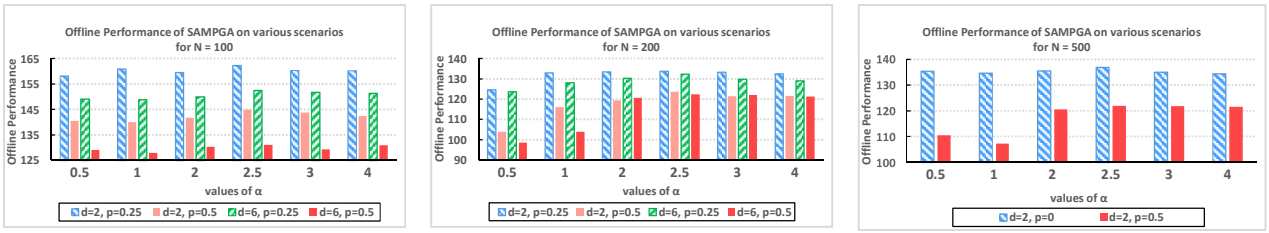
N	d	ρ	Calculate Distance (convergence radius – exclusion radius)			
			Hamming-Manhattan	Hamming-Hamming	Manhattan-Hamming	Manhattan-Manhattan
100	2	0.25	25.44±0.37	**24.92±0.24**	27.10±0.24	27.29±0.21
100	2	0.5	42.33±0.35	**41.24±0.34**	44.04±0.30	44.26±0.24
100	6	0.25	**34.79±0.23**	35.89±0.16	36.55±0.18	36.58±0.15
100	6	0.5	55.10±0.19	**54.32±0.16**	54.93±0.16	55.00±0.19
200	2	0.25	53.13±0.07	**53.08±0.08**	53.90±0.07	53.80±0.07
200	2	0.5	**56.84±0.19**	64.14±0.22	63.93±0.18	63.73±0.24
200	6	0.25	**56.03±0.06**	56.95±0.07	57.50±0.08	57.58±0.06
200	6	0.5	**53.74±0.18**	65.16±0.17	65.36±0.17	65.88±0.20
500	2	0.0	**51.19±0.09**	51.69±0.07	51.73±0.10	51.81±0.06
500	2	0.5	**54.34±0.26**	65.55±0.20	65.00±0.17	65.62±0.17

Table 3. Offline error of SAMPGA for α = 2.5 with different measuring modes for r_{conv} and r_{excl}

N	d	ρ	Calculate Distance (convergence radius – exclusion radius)			
			Hamming-Manhattan	Hamming-Hamming	Manhattan-Hamming	Manhattan-Manhattan
100	2	0.25	25.48±0.44	**24.96±0.30**	27.22±0.28	27.36±0.30
100	2	0.5	41.21±0.40	**40.94±0.34**	43.85±0.33	44.18±0.26
100	6	0.25	**34.82±0.26**	36.10±0.23	36.53±0.23	36.92±0.26
100	6	0.5	53.26±0.20	**52.36±0.23**	53.11±0.26	53.22±0.29
200	2	0.25	**52.83±0.22**	53.11±0.21	53.96±0.21	53.81±0.22
200	2	0.5	**52.87±0.21**	60.59±0.19	60.86±0.25	61.04±0.19
200	6	0.25	**56.21±0.18**	57.02±0.21	57.65±0.19	57.66±0.18
200	6	0.5	**56.38±0.16**	61.88±0.19	62.11±0.17	62.45±0.20
500	2	0.0	**50.44±0.24**	51.62±0.25	51.63±0.20	51.59±0.20
500	2	0.5	**53.56±0.24**	62.78±0.22	62.72±0.18	62.83±0.18

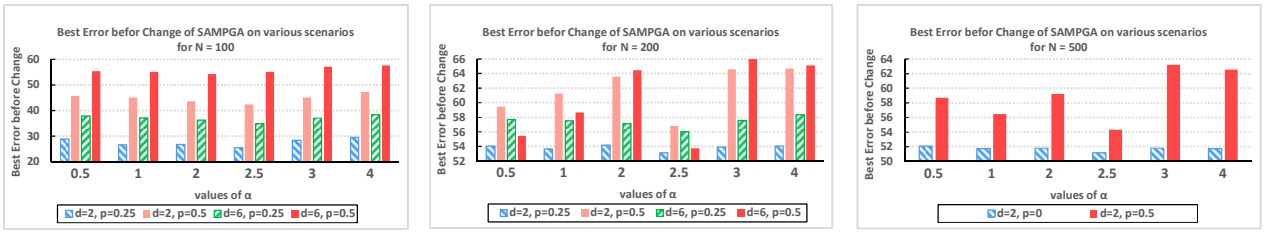
Regarding Table 2 and Table 3, it is observed that the combination of Hamming distance and Manhattan distance are the best distance measures for computing r_{conv} and r_{excl}, respectively.

As the second part of this experiment, we investigate the effect of parameter α on the performance of SAMPGA for scenarios with different combinations of resource need d and QoS ρ. The experimental results for different values of α are illustrated in Figure 2 to Figure 4.



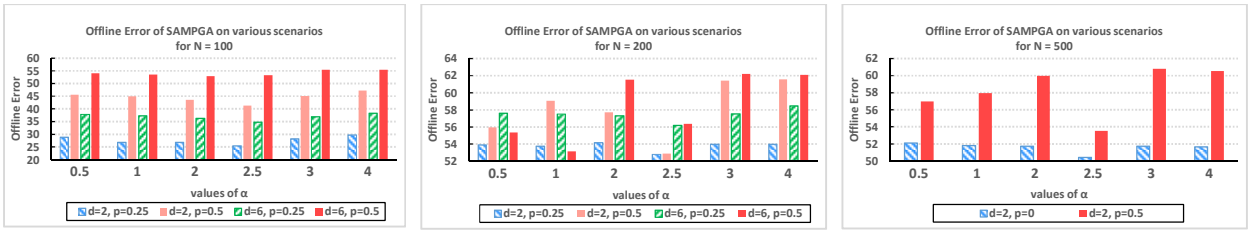
Higher values mean better efficiency.

Figure 2. Offline performance of SAMPGA in various scenarios for different values of α (r_{conv} : Hamming - r_{excl} : Manhattan)



Lower values mean better efficiency.

Figure 3. Best error befor change of SAMPGA in various scenarios for different values of α (r_{conv} : Hamming - r_{excl} : Manhattan)



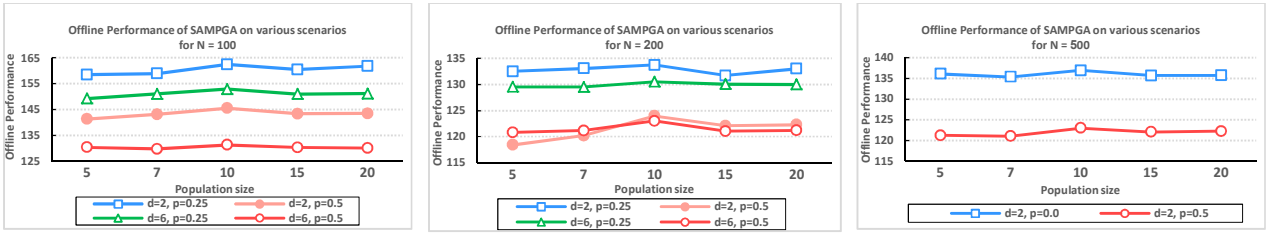
Lower values mean better efficiency.

Figure 4. Offline error of SAMPGA in various scenarios for different values of α (r_{conv} : Hamming - r_{excl} : Manhattan)

From Figure 2 to Figure 4, it can be seen that SAMPGA with $\alpha = 2.5$ provides best results.

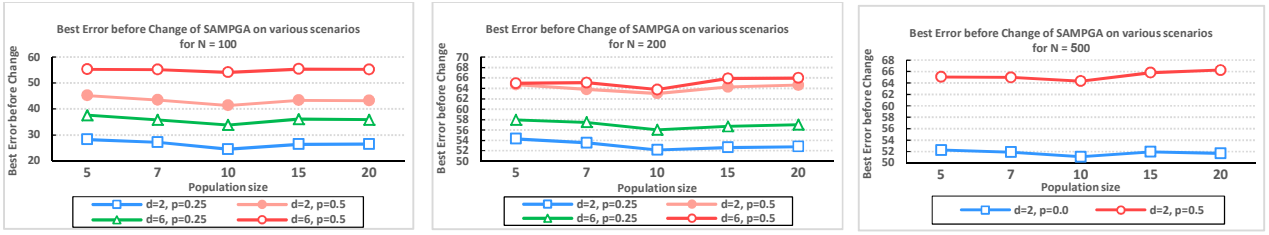
5.3.2. Effect of population size

This experiment examines the effect of the number of chromosomes in each population. To analyze the effect of the population size of each population on the performance of the SAMPGA, the algorithm was run for different number of individuals per population. The obtained results for scenarios with different combinations of resource need d and QoS ρ are summarized in Figure 5 to Figure 7.



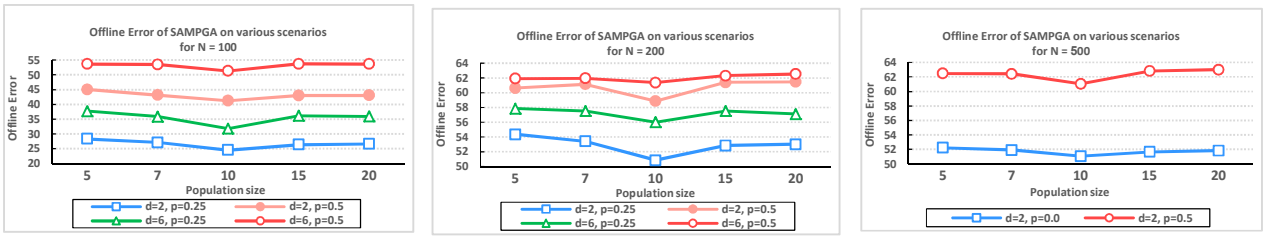
Higher values mean better efficiency. Note thath

Figure 5. Offline performance of SAMPGA in various scenarios for different values of the population size



Lower values mean better efficiency.

Figure 6. Best error before change of SAMPGA in various scenarios for different values of the population size



Lower values mean better efficiency.

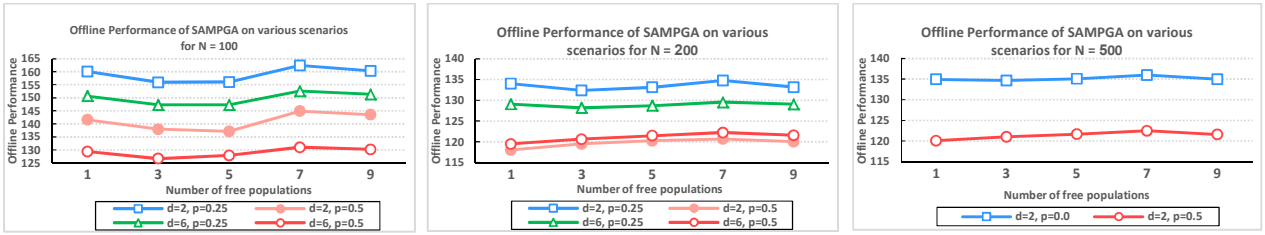
Figure 7. Offline error of SAMPGA in various scenarios for different values of the population size

As expected, the population size has a great impact on the performance of the proposed algorithm. From one hand, a large population size induces more computational effort to evolve and this degrades the performance of the SAMPGA. From the other hand, a small population size increases the probability of falling into local optima which is detrimental to the performance of the SAMPGA. As can be observed in the figures, the algorithm performs better when each population has 10 individuals.

5.3.3. Effect of the number of free populations n_{excess}

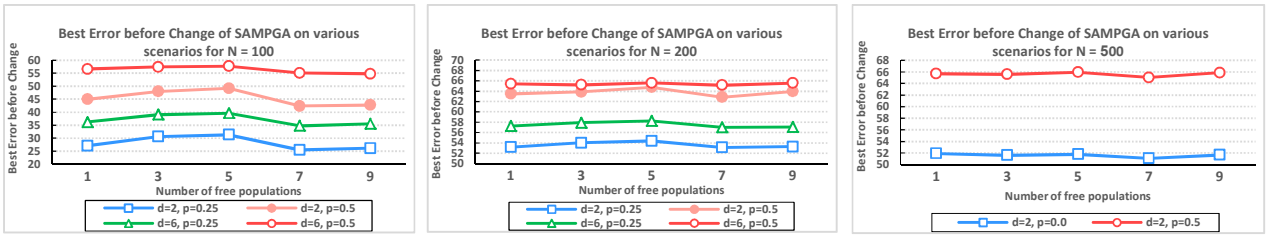
The main goal of this experiment is to investigate the effect of parameter n_{excess} on the performance of SAMPGA for scenarios with different combinations of resource need d and QoS ρ. As we mentioned in Section 4, free populations are responsible for exploring the search space to locate new promising areas. Therefore, it is expected that the parameter n_{excess} has a significant influence on the behavior of the proposed method. On the one hand, a very small value of n_{excess} will result in that the SAMPGA fails to sustain adequate diversity needed for locating promising areas of the search space or new emerging optima. On the other hand, a large value of n_{excess} will create an excessive number of populations which is known to be mainly destructive.

The results of experiments with different values for parameter n_{excess} are presented in Figure 8 to Figure 10.



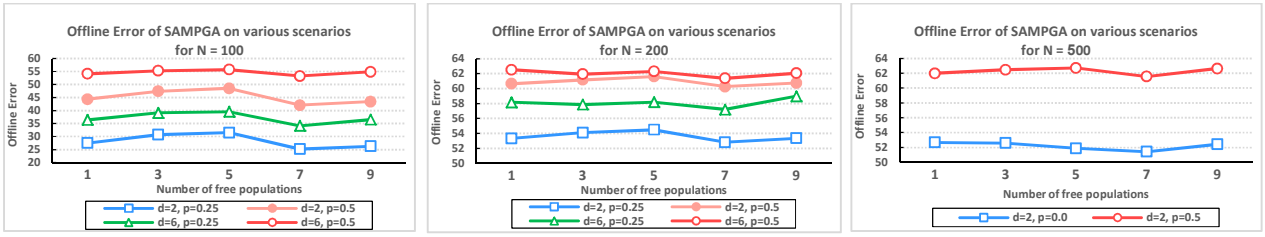
Higher values mean better efficiency.

Figure 8. Offline performance of SAMPGA in various scenarios for different numbers of free populations



Lower values mean better efficiency.

Figure 9. Best error before change of SAMPGA in various scenarios for different numbers of free populations



Lower values mean better efficiency.

Figure 10. Offline error of SAMPGA in various scenarios for different numbers of free populations

From the above figures, the best results are observed when $n_{excess} = 7$.

5.4. Results and discussion

In this section, we validate the performance of our algorithms based on a set of experiments with a subset of scenarios described in Section 5.1 (the full results for the complete set of experiments can be found in the supplementary file). We also compare our methods with self-organizing random immigrants GA (SORIGA) [43], a popular and highly regarded GA for DOPs. The values of parameters for the SORIGA are set according to the values given in [43]. The numerical results are given in Table 4 and Table 5.

To determine the differences among tested algorithms, we applied nonparametric tests. First, we applied the Friedman's test ($P < 0.05$) to verify whether significant differences exist at group level, that is, among all algorithms. In case of such differences exist, then a Wilcoxon test ($\alpha = 0.05$) is applied to compare our proposed methods with SORIGA. The last column of Table 4 and Table 5 includes the p-value of the Wilcoxon test. The outcomes of the Wilcoxon test revealing that no significant differences exist between the algorithms, are highlighted with an asterisk symbol.

Table 4. Performance measurements of four algorithms in various scenarios using best-error-before-change

N	d	ρ	SORIGA	SAMPGA	MSAMPGA	ISAMPGA	p-value
100	2	0.25	33.16±0.09	26.32±0.26	25.64±0.25	**21.95±0.37**	0.00
100	2	0.5	50.17±0.09	42.50±0.24	41.79±0.32	**36.39±0.45**	0.00
100	6	0.25	40.54±0.06	35.12±0.18	35.04±0.18	**31.72±0.51**	0.00
100	6	0.5	55.96±0.07	52.30±0.10	52.08±0.11	**50.27±0.17**	0.00
200	2	0.25	53.53±0.06	52.19±0.08	51.28±0.66	**49.72±0.12**	0.00
200	2	0.5	59.84±0.10	59.37±0.09	59.45±0.09	**59.14±0.09**	0.00

200	6	0.25	56.84±0.07	55.75±0.29	55.43±0.30	**53.73±0.17**	0.00
200	6	0.5	62.75±0.05	62.38±0.06	**62.37±0.05**	62.50±0.05	0.00
500	2	0.0	50.45±0.07	50.48±0.08	50.37±0.08	**49.86±0.07**	0.00
500	2	0.5	**64.74±0.13**	65.53±0.21	64.97±0.13	64.84±0.09	0.22*
500	4	0.0	52.06±0.06	52.04±0.06	51.95±0.06	**51.46±0.06**	0.00
500	6	0.0	52.51±0.07	52.50±0.07	52.41±0.08	**51.96±0.07**	0.00

The best results are highlighted in *boldface*.

Table 5. Performance measurements of four algorithms in various scenarios using offline error

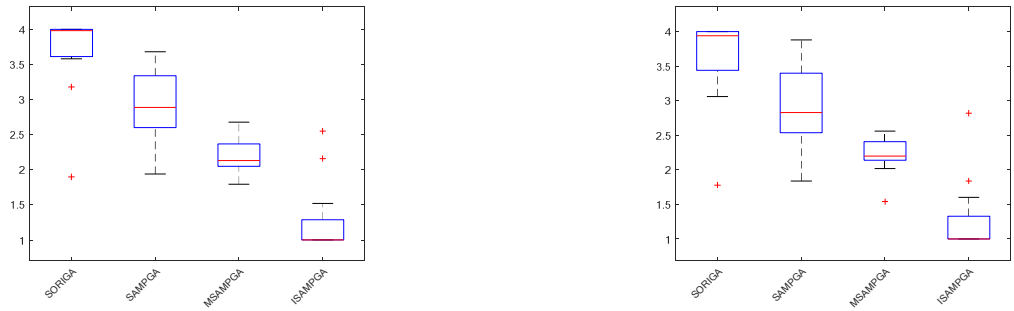
N	d	ρ	SORIGA	SAMPGA	MSAMPGA	ISAMPGA	p-value
100	2	0.25	33.42±0.14	27.34±0.23	26.67±0.23	**23.20±0.42**	0.00
100	2	0.5	50.33±0.10	43.44±0.25	42.69±0.37	**37.72±0.48**	0.00
100	6	0.25	40.70±0.12	35.76±0.18	35.77±0.15	**32.54±0.46**	0.00
100	6	0.5	56.05±0.08	52.57±0.10	52.36±0.12	**50.71±0.21**	0.00
200	2	0.25	53.59±0.08	52.33±0.07	51.91±0.19	**50.06±0.14**	0.00
200	2	0.5	59.73±0.10	59.36±0.10	59.42±0.09	**59.16±0.10**	0.00
200	6	0.25	56.94±0.08	55.97±0.14	55.69±0.10	**53.76±0.34**	0.00
200	6	0.5	62.76±0.04	62.38±0.05	**62.36±0.05**	62.50±0.03	0.00
500	2	0.0	50.45±0.08	50.50±0.08	50.38±0.08	**49.89±0.07**	0.00
500	2	0.5	**64.89±0.07**	65.61±0.10	65.03±0.07	64.92±0.06	0.40*
500	4	0.0	52.08±0.05	52.07±0.05	51.98±0.04	**51.49±0.04**	0.00
500	6	0.0	52.46±0.06	52.45±0.05	52.36±0.06	**51.94±0.05**	0.00

The best results are is highlighted in *boldface*.

Different conclusions can be drawn from the comparisons in Table 4 and Table 5. First, the algorithm ISAMPGA is clearly the best-performing algorithm on majority of scenarios. Considering Table 4 and Table 5, the ISAMPGA is the best in 10 out 12 problem instances. Besides, the SORIGA is the worst of four. It is outperformed by our best-performing algorithm on 11 out of 12. Note that for the problem instance with N=500, d=2 and ρ=0.5, the result of SORIGA is better than the result of our best-performing algorithm. However, this superiority is not statistically significant (p-value>0.05). The reason why SORIGA exhibits such poor performance is that SORIGA is unable to locate and track multiple changing optima simultaneously. Moreover, SORIGA is constantly trying to maintain the diversity, which may degrade its performance. Second, it is observed that the use of memory component and random-immigrant is favorable to the performance of the basic SAMPGA. However, ISAMPGA is more preferable than MSAMPGA.

In order to have a better judgement about the optimality of the tested algorithms, we have performed non-parametric statistical tests using all obtained results. The analysis is divided in in two groups based on (*a*) offline error, and (*b*) best-error-before-change. The results are depicted in Figure 11 using boxplot. It is important to note that the lower the rank the better is the algorithm performance. Each boxplot in the figure correspond to a GA variant with lines at the lower quartile, median, and upper quartile data values, where the whiskers are the lines extending from each end of the box to show the extent of the rest of the data.

Based on the figure, it is observed that ISAMPGA outperforms the other counterparts in terms of both offline error and best-error-before-change.



(*a*) (*b*)

Figure 11. Ranking of the methods using Friedman test in different scenarios. The graphics summarized the rankings according the performance measure: (a) best error before change and (b) offline error

To conclude the above analysis, we conducted pairwise comparisons with the four algorithms using the Wilcoxon's signed rank test. This time, all the reported values from Table 4 and Table 5 were used as input for the tests. The results of the performed pairwise comparisons are shown in Table 6, where the obtained p-value for every pair of algorithms are listed in the last column.

Table 6. Pairwise comparisons using the Wilcoxon's signed rank test in all tested scenarios

Algorithm 1	vs.	Algorithm 2	p-value
ISAMPGA		MSAMPGA	$\mathbf{4.1129 \times 10^{-5}}$
		SAMPGA	$\mathbf{2.6616 \times 10^{-5}}$
		SORIGA	$\mathbf{2.6666 \times 10^{-5}}$
MSAMPGA		SAMPGA	$\mathbf{7.0716 \times 10^{-5}}$
		SORIGA	$\mathbf{1.1435 \times 10^{-4}}$
SAMPGA		SORIGA	$\mathbf{9.1600 \times 10^{-4}}$

The values in *boldface* correspond to statistically significant differences between the involved algorithms (p-value<0.05)

From Table 6, we can reach the conclusion that our proposed algorithms are significantly different from the SORIGA. This can be deduced from the lower ranks of our proposed methods in the Friedman test (see Figure 11). All in all, we can conclude that our proposed methods are more effective.

Despite the results reported in Tables 4 to Table 6, we also analyzed the algorithms behavior over time. In Figure 12 and Figure 13 we plotted the evolution of the average current fitness error over time. For the sake of readability, only 20 changes per run were considered. Moreover, the results were averaged over 50 runs.

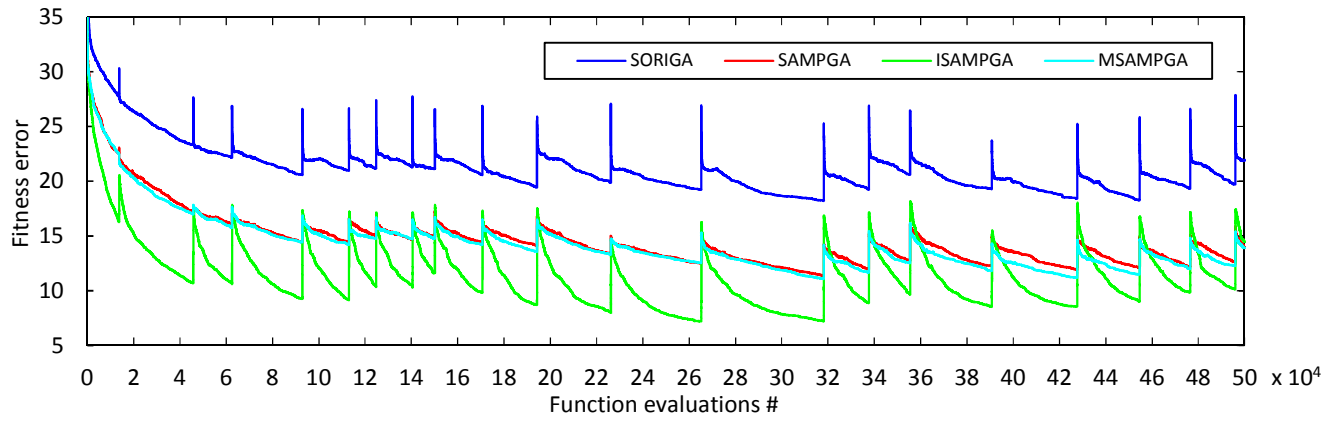


Figure 12. Evolution of average error over time for (H=64, N=100, d=2, ρ=0)

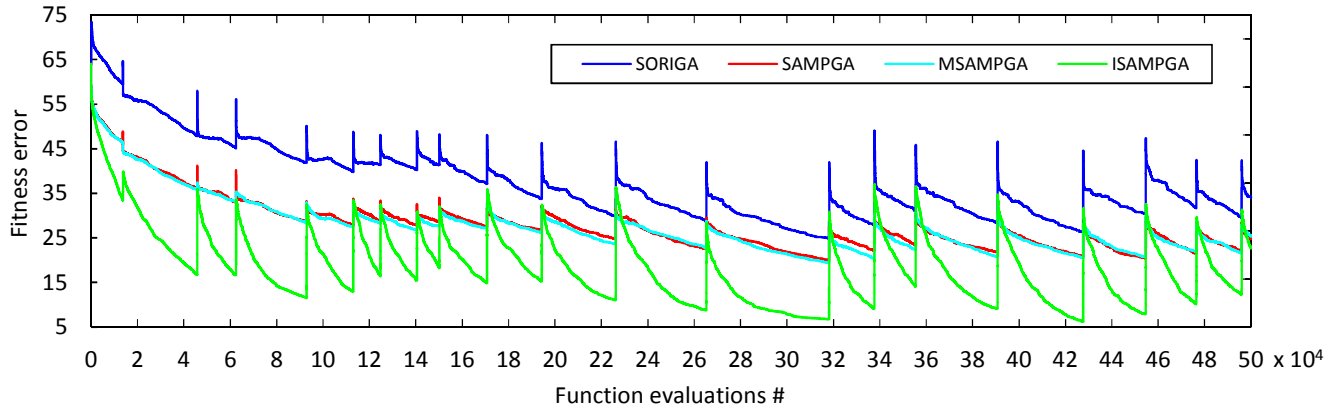


Figure 13. Evolution of average error over time for (H=64, N=100, d=2, ρ=0.5)

Regarding Figure 12 and Figure 13, the first thing that stands out is that SORIGA has the poorest performance during the whole search process. From the curves, it is noticeable that the SAMPGA can MSAMPGA have more or less the same performance. Finally, as can be observered in the figures, the ISAMPGA has a much better performance than the other three algorithms.

As the final experiment, we have measured the running time of SAMPGA, MSAMPGA, ISAMPGA and SORIGA (see Table 7).

Table 7. Average running time (in seconds) of different GAs for various scenarios with H=64, d=2, ρ=0.25 over 50 runs

Method	N=100	N=200	N=500
SORIGA	**0.22**	**0.32**	**0.56**
SAMPGA	0.24	0.36	0.59
MSAMPGA	0.27	0.48	0.71
ISAMPGA	0.25	0.41	0.83

As expected, the running time of algorithms increases with an increase in number of services. As can be observed in Table 7, the SORIGA require less time than all other GA variants for different

number of service values considered. However, the difference between the running time of our algorithms and the SORIGA is not crucial.

6. Conclusion

A novel dynamic resource allocation in shared hosting platform was presented and discussed in this paper. The inaccessibility of the servers during the resource allocation has been considered, and different algorithms have been proposed to handle arising challenges.

The first proposed algorithm, SAMPGA, is a genetic algorithm with a multi-population scheme to efficiently handle exploration, optimum tracking, change detection, and premature convergence. Simulation results showed that the SAMPGA outperformed SORIGA. The reason is that the SAMPGA benefits from multi-population scheme which is helpful for locating and tracking multiple optima simultaneously.

The second algorithm, MSAMPGA, is a combination of SAMPGA and explicit memory that yields better results. MSAMPGA can speed up the adaptation of the populations with new environments after the change.

The third extension to SAMPGA is ISAMPGA which aim at dynamic resource allocation using random immigrants. The remarkable features of the random immigrant improved the efficiency of SAMPGA. ISAMPGA helps the converged population to track the shifted optima after the change by generating diversity.

The considered shared hosting platforms in this paper were homogeneous. Future reasearch may focus on applying the proposed methods for heterogeneous platforms. Moreover, the examined platforms were hosting services that had static workloads where the amount of requested resources is not changing during allocation and each service required static and determined resources. Another possible future research direction could be devoted to simulating situations where the workload is also dynamic.

References

[1] B. Urgaonkar, P. Shenoy, and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms," *SIGOPS Oper Syst Rev*, vol. 36, no. SI, pp. 239–254, Dec. 2002.

[2] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen, "Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure," in *2006 IEEE International Conference on Autonomic Computing*, 2006, pp. 5–14.

[3] S. S. Manvi and G. Krishna Shyam, "Resource Management for Infrastructure as a Service (IaaS) in Cloud Computing: A Survey," *J. Netw. Comput. Appl.*, vol. 41, pp. 424–440, May 2014.

[4] V. P. Anuradha and D. Sumathi, "A Survey on Resource Allocation Strategies in Cloud Computing," in *International Conference on Information Communication and Embedded Systems (ICICES2014)*, 2014, pp. 1–7.

[5] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)," 1979, p. 3.

[6] K. Shen, H. Tang, T. Yang, and L. Chu, "Integrated Resource Management for Cluster-based Internet Services," *SIGOPS Oper Syst Rev*, vol. 36, no. SI, pp. 225–238, Dec. 2002.
[7] A. Karve *et al.*, "Dynamic Placement for Clustered Web Applications," in *Proceedings of the 15th International Conference on World Wide Web*, New York, NY, USA, 2006, pp. 595–604.
[8] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguade, "Utility-based Placement of Dynamic Web Applications with Fairness Goals," in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium*, 2008, pp. 9–16.
[9] J. Rolia, A. Andrzejak, and M. Arlitt, "Automating Enterprise Application Placement in Resource Utilities," in *Self-Managing Distributed Systems*, Springer, Berlin, Heidelberg, 2003, pp. 118–129.
[10] M. Bichler, T. Setzer, and B. Speitkamp, "Capacity Planning for Virtualized Servers," Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 1025862, Nov. 2007.
[11] A. Xiong and C. Xu, "Energy Efficient Multiresource Allocation of Virtual Machine Based on PSO in Cloud Data Center," *Mathematical Problems in Engineering*, 2014.
[12] Z. Zheng, R. Wang, H. Zhong, and X. Zhang, "An Approach for Cloud Resource Scheduling Based on Parallel Genetic Algorithm," in *2011 3rd International Conference on Computer Research and Development*, 2011, vol. 2, pp. 444–447.
[13] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics," in *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, 2008, pp. 326–335.
[14] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing Energy and Server Resources in Hosting Centers," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, New York, NY, USA, 2001, pp. 103–116.
[15] C. T. Joseph, K. Chandrasekaran, and R. Cyriac, "A Novel Family Genetic Approach for Virtual Machine Allocation," *Procedia Comput. Sci.*, vol. 46, pp. 558–565, Jan. 2015.
[16] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-based Network Servers," in *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, 2000, pp. 90–101.
[17] G. Pacifici, M. Spreitzer, A. N. Tantawi, and A. Youssef, "Performance Management for Cluster-based Web Services," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 12, pp. 2333–2343, Dec. 2005.
[18] D. Kumar, B. Sahoo, B. Mondal, and T. Mandal, "A Genetic Algorithmic Approach for Energy Efficient Task Consolidation in Cloud Computing," *Int. J. Comput. Appl.*, vol. 118, no. 2, pp. 1–6, May 2015.
[19] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, "Resource Allocation Algorithms for Virtualized Service Hosting Platforms," *J. Parallel Distrib. Comput.*, vol. 70, no. 9, pp. 962–974, Sep. 2010.
[20] S. Ali, J.-K. Kim, H. J. Siegel, and A. A. Maciejewski, "Static Heuristics for Robust Resource Allocation of Continuously Executing Applications," *J. Parallel Distrib. Comput.*, vol. 68, no. 8, pp. 1070–1080, Aug. 2008.
[21] H. Nguyen Van, F. Dang Tran, and J.-M. Menaud, "Autonomic Virtual Resource Management for Service Hosting Platforms," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, Washington, DC, USA, 2009, pp. 1–8.
[22] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and Stretch Metrics for Scheduling Continuous Job Streams.," in *SODA*, 1998, vol. 98, pp. 270–279.
[23] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary Dynamic Optimization: A Survey of the State of the Art," *Swarm Evol. Comput.*, vol. 6, pp. 1–24, Oct. 2012.
[24] H. G. Cobb, "An Investigation into the Use of Hypermutation as an Adaptive Operator in Genetic Algorithms Having Continuous, Time-Dependent Nonstationary Environments," Technical Report AIC-90-001, Naval Research Laboratory, Washington, USA, 1990.
[25] F. Vavak, K. A. Jukes, and T. C. Fogarty, "Performance of a Genetic Algorithm with Variable Local Search Range Relative to Frequency of the Environmental Changes," *Genet. Program.*, pp. 22–25, 1998.

[26] F. Vavak, K. Jukes, and T. C. Fogarty, “Learning the Local Search Range for Genetic Optimisation in Nonstationary Environments,” in *, IEEE International Conference on Evolutionary Computation, 1997*, 1997, pp. 355–360.
[27] J. J. Grefenstette, “Genetic Algorithms for Changing Environments,” in *PPSN*, 1992, vol. 2, pp. 137–144.
[28] H. C. Andersen, “An Investigation into Genetic Algorithms, and the Relationship between Speciation and the Tracking of Optima in Dynamic Functions,” *Queenslan Univ. Technol. Honours Thesis Novemb.*, 1991.
[29] R. W. Morrison, “Designing Evolutionary Algorithms for Dynamic Environments,” PhD Thesis, George Mason University, Fairfax, VA, USA, 2002.
[30] J. Branke, T. Kaussler, C. Smidt, and H. Schmeck, “A Multi-population Approach to Dynamic Optimization Problems,” in *Evolutionary Design and Manufacture*, Springer, London, 2000, pp. 299–307.
[31] F. Oppacher and M. Wineberg, “The Shifting Balance Genetic Algorithm: Improving the GA in a Dynamic Environment,” in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 1*, San Francisco, CA, USA, 1999, pp. 504–510.
[32] R. K. Ursem, “Multinational GA Optimization Techniques in Dynamics Environments,” in *Genetic and Evolutionary Computation Conference*, 2000, pp. 19–26.
[33] P. Barham *et al.*, “Xen and the Art of Virtualization,” in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, New York, NY, USA, 2003, pp. 164–177.
[34] M. Stillwell, F. Vivien, and H. Casanova, “Dynamic Fractional Resource Scheduling for HPC Workloads,” in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010, pp. 1–12.
[35] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, “Resource Allocation Using Virtual Clusters,” in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Washington, DC, USA, 2009, pp. 260–267.
[36] T. Blackwell, “Particle Swarm Optimization in Dynamic Environments,” in *Evolutionary Computation in Dynamic and Uncertain Environments*, Springer, Berlin, Heidelberg, 2007, pp. 29–49.
[37] T. Blackwell and J. Branke, “Multi-swarm Optimization in Dynamic Environments,” in *Applications of Evolutionary Computing*, Springer, Berlin, Heidelberg, 2004, pp. 489–500.
[38] K. Trojanowski and Z. Michalewicz, “Evolutionary Optimization in Non-stationary Environments,” *J. Comput. Sci. Technol.*, vol. 1, no. 2, Mar. 2000.
[39] S. Yang, H. Cheng, and F. Wang, “Genetic Algorithms With Immigrants and Memory Schemes for Dynamic Shortest Path Routing Problems in Mobile Ad Hoc Networks,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 40, no. 1, pp. 52–63, Jan. 2010.
[40] J. Branke and H. Schmeck, “Designing Evolutionary Algorithms for Dynamic Optimization Problems,” in *Advances in Evolutionary Computing*, Springer, Berlin, Heidelberg, 2003, pp. 239–262.
[41] K. Trojanowski and Z. Michalewicz, “Searching for Optima in Non-stationary Environments,” in *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, 1999, vol. 3, p. 1850.
[42] A. E. Ranginkaman, J. Kazemi Kordestani, A. Rezvanian, and M. R. Meybodi, “A Note on the Paper ‘A Multi-population Harmony Search Algorithm with External Archive for Dynamic Optimization Problems’ by Turky and Abdullah,” *Inf. Sci.*, vol. 288, pp. 12–14, Dec. 2014.
[43] R. Tinós and S. Yang, “A Self-organizing Random Immigrants Genetic Algorithm for Dynamic Optimization Problems,” *Genet. Program. Evolvable Mach.*, vol. 8, no. 3, pp. 255–286, Sep. 2007.