Reinforcement Learning-Assisted Genetic Programming Hyper Heuristic Approach to Location-Aware Dynamic Online Application Deployment in Clouds

Longfei Felix Yan, Hui Ma, Gang Chen
{felix.yan,hui.ma,aaron.chen}@ecs.vuw.ac.nz
Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand

ABSTRACT

Location-Aware Dynamic Online Application dePloyment (LADOAP) in clouds is an NP-hard combinatorial optimisation problem. Genetic Programming Hyper-Heuristic (GPHH) has emerged as a promising approach for addressing LADOAP demands by dynamically generating Virtual Machine (VM) selection heuristics online. However, the performance of GPHH is impeded by long simulation times and low sampling efficiency. In this paper, we propose a novel hyper-heuristic framework that integrates Genetic Programming Hyper-Heuristic (GPHH) and Reinforcement Learning (RL) approaches to evolve rules for efficiently selecting location-aware Virtual Machines (VMs) capable of hosting multiple containers. The RL policy's value function acts as a surrogate model, significantly expediting the evaluation of generated VM selection rules. By applying this hybrid framework to LADOAP problems, we achieve competitive performance with a notable reduction in the number of required simulations. This innovative approach not only enhances the efficiency of VM selection but also contributes to advancing the state-of-the-art in addressing complex LADOAP challenges.

CCS CONCEPTS

• **Computing methodologies** → **Reinforcement learning**; **Genetic programming**.

KEYWORDS

genetic programming hyper heuristic, reinforcement learning, surrogate model, application deployment

ACM Reference Format:
Longfei Felix Yan, Hui Ma, Gang Chen. 2024. Reinforcement Learning-Assisted Genetic Programming Hyper Heuristic Approach to Location-Aware Dynamic Online Application Deployment in Clouds. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3638529.3654058

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07…$15.00
https://doi.org/10.1145/3638529.3654058

1 INTRODUCTION

Recent years have witnessed a surge in cloud services market, with global spending forecasting $679 billion in 2024 and over $1 trillion in 2027 [13]. Container-based application deployment is often utilised to meet cloud computing demands [21, 40, 50]. Containers allow multiple applications to be deployed on the same Virtual Machine (VM) by sharing one operating system, which can effectively reduce the resources required for application deployment [6]. Furthermore, containers provide isolated sandboxes for serverless applications, which are lightweight and secure [20].

For real-world applications, user requests generally span across the globe. VMs in data centres of different geographic locations must be considered to minimise the average response time [39]. Therefore, location-aware cloud environments have become increasingly popular for container-based application deployment [2]. In a location-aware cloud environment, customers can enjoy the opportunity of using the computing facilities that best suit their requirements according to their geographic locations [38]. This consideration of locations adds another layer of complexity to the possible solution combinations, which gives the already challenging container-based application deployment even a larger solution search space.

Rather than directly searching for solutions, using metaheuristics or hyper-heuristics can make computationally intractable problems easier [11]. However, it is well-known that metaheuristics can take long computational time and are not suitable for dynamic problems [4]. In contrast, hyper-heuristics can handle dynamic problems with fast online solutions. Genetic Programming Hyper-Heuristic (GPHH) is a popular approach that generates a population of tree-based computer programs to represent heuristics [7]. By automatically optimising the population in a manner similar to natural evolution, GPHH has been successfully applied to online resource allocation problems to reduce energy consumption [44–46]. Thus, GPHH has good potential for Location-Aware Dynamic Online Application dePloyment (LADOAP) problems, which have not been investigated in existing works.

The evaluation of heuristic rules for LADOAP problems is based on historical data collected, categorising them as data-driven optimisation problems [18]. Each evolved rule in GPHH needs to be evaluated separately through LADOAP simulations. However, simulations in GPHH can be computation-intensive and time-consuming [59]. This issue challenges the future use of GPHH for solving real-world LADOAP problems. Surrogate models are often utilised to approximate the real quality evaluation function and speed up the

optimisation process [52, 53]. Research on surrogate models for GPHH is still in an early stage. In [17, 59, 60], phenotypic representations have been developed for surrogate models in dynamic job shop scheduling tasks. These representations are fixed-length vectors that correspond to a sequence of decision situations regarding machine ranks obtained from GP heuristic phenotypes. Unlike job shop scheduling, LADOAP problems have a large number of machines (VMs) globally available for each decision. Using similar phenotypic representations would require a vector of possibly hundreds of entries, which may not effectively represent genotypic behaviours. On the other hand, the value function of RL's policy is a natural assessment of a sequence of decisions. This suggests that RL-based surrogate model for GPHH can fit well into the picture of LADOAP problems.

To make GPHH more efficient by reducing the total number of simulations required without hurting its effectiveness, we developed a novel surrogate model by utilizing time-tested reinforcement learning techniques, including the temporal difference learning method in this paper. We adopt a unique perspective, positing that the value function of a reinforcement learning (RL) policy can be depicted as a tree structure. This structure allows for evaluation and incremental enhancement through a GPHH based algorithm, which is referred to as the **GPHH-RL** algorithm in this paper.

1.1 Contributions

Through the development and experimental evaluation of the GPHH-RL algorithm, this paper achieves the following contributions:

(1) We propose a novel hyper-heuristic algorithm, GPHH-RL, to solve LODOAP problems efficiently and effectively.
(2) To our best knowledge, GPHH-RL is the first hyper-heuristic algorithm that utilises GP-based Q-functions as surrogate models to drive the evolution of useful heuristics.
(3) We conduct a comprehensive experimental analysis of GPHH-RL with respect to GPHH-based and classical heuristic baselines. Our results outperform all the baseline algorithms.

2 RELATED WORK

In this section, we first introduce related work in the domain of container-based application deployment in clouds. We then discuss the works pertaining to RL-Assisted Genetic Programming.

2.1 Container-Based Application Deployment in Clouds

The deployment of containerised applications in clouds has become a burgeoning technology due to its elasticity and lightweight deployment cost [5, 33]. Kubernetes-powered commercial container management platforms, such as Rancher [31] and OpenShift [15], deliver container deployment services in public clouds through a unified API. Effective strategies for container-based application deployment in clouds have become more essential than ever [39, 54].

Several studies have investigated queuing models when deploying containerised applications in clouds [34, 39, 47]. In this paper, our GPHH-RL algorithm also utilises an $M/M/1$ queue for modelling the average response time of applications. Due to the long-running nature of applications, queuing models help predict the processing time of deployed applications. However, these previous studies ignore the location information of the data centres [34, 47] or overlook the possibility of using shared VMs for multiple containers [39].

Another stream of research in the deployment of container-based applications in clouds focuses on application replication strategies [25, 37, 41]. Most of existing research deals with offline application deployment. In [37], the authors utilised Mixed Integer Linear Programming (MILP) and Large Neighbourhood Search (LNS) to select VMs for application deployment. Both MILP and LNS are not only time-consuming, but also require future application arrival information a priori. Hence, they are not suitable for dynamic online application deployment problems.

A closely related research field to LADOAP is Resource Allocation in Container-based clouds (RAC) [44, 46, 51]. The similarity lies in the fact that RAC problems also involve selecting VMs for application deployment. The difference is that RAC problems are from the perspective of cloud providers, not the perspective of cloud users. Energy consumption is the main concern in RAC problems [44, 46, 51], while user experience attributes such as application response time are not considered in RAC problems. Traditionally, RAC problems are solved by manually designed heuristics like Best-Fit algorithms [1, 9]. They are generally outperformed by GPHH counterparts [44, 46, 51].

2.2 Reinforcement Learning-Assisted Genetic Programming

Hybrid approaches that interweave RL with GP have received growing interest in recent years [42]. GP can effectively perform population-based random searches to explore useful solutions for a given problem. However, GP algorithms often suffer from low sampling efficiency and generalisability [10, 42]. On the other hand, RL algorithms convert the given problem into a sequential decision-making problem. They usually exhibit strong reasoning abilities [19]. Furthermore, RL-based surrogate models have low time complexity and speed up the evaluation process [14].

RL-assisted GP can fall into three major categories according to the role of RL: solution generation [27], population selection [14], and learnable objective functions [16, 30, 48, 55]. The goal of solution generation is to generate complete (or partial) solutions for the evolutionary search process to use. A GP solution is represented by a syntax tree that can be compiled as an expression of a callable function [55]. Researchers in [27] use Recurrent Neural Network (RNN)-based RL algorithm to generate the initial population, which is called population seeding. This technique has two-fold benefits. Firstly, RL-guided population seeding provides good starting populations facilitating the evolutionary process in GP. Second, GP helps RNN escape local optima by providing new solutions that are not derived from gradient descent.

Population selection can be performed by employing an ensemble population strategy [56]. As finding high-quality solutions with a single population search is a challenging task [58], an ensemble population strategy can be utilised to improve the search performance. The researchers in [14] design four search modes for four population strategies. The selection of population is dynamically

adjusted in the RL algorithm to balance the exploration and exploitation of the population search. This increases the chances of finding better solutions.

Using RL for learnable objective functions in GP is the most common way of integrating RL into GP [16, 30, 48, 55]. The objective function to evaluate the fitness of individual solutions plays a fundamental role in GP. By employing a learnable objective function, the explored information can be incorporated effectively to guide the search direction [42]. As RL formulates a problem as a sequence of decisions, the trajectory samples of the state-action space can be utilised to learn the objective function [16]. In particular, [16] proposes a model-based RL algorithm in which the environment is known. In other words, the expected reward for every state is known in advance. In practice, many environments are not known, and the interaction between RL decisions and the environment has to be explored through trial-and-error [26]. In this type of model-free RL algorithms, the learnable objective function, or the value function of the RL policy, is derived from sample trajectories of state-action pairs.

In summary, RL-assisted GP remains a relatively underexplored domain. Among the research directions of RL-assisted GP, the application of RL to construct a learnable objective function particularly piques our interest. Inspired by this approach, we are interested in investigating RL-based surrogate models that approximate objective functions. The RL-based surrogate models can capitalise on the synergistic fusion of RL's potent learning capabilities with GP's robust search capabilities. Currently, no empirical investigations or studies have been undertaken in this specific direction and we would like to be the first to tackle this research question.

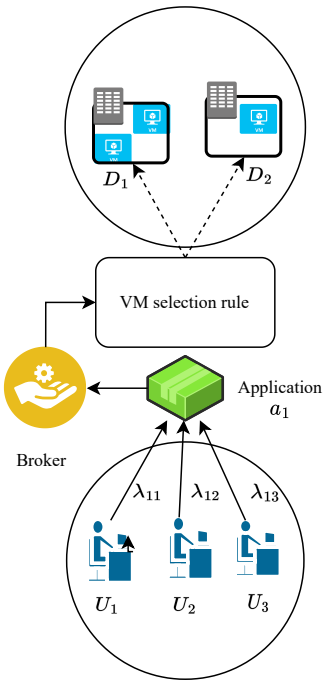

Figure 1: A simplified example of deploying one containerised application through a cloud application deployment broker. The application is associated with three global user centres and it will be deployed on a VM in one of the two geo-distributed data centres.

3 PROBLEM FORMULATION

Dynamically arrived applications can be deployed on VMs provided in geo-distributed data centres with the help of a cloud application deployment broker. A simple example is illustrated in Figure 1. The mathematical notation utilised for the problem formulation is presented in Table 1.

Given a time period T (e.g., one day), a set of container-based applications arrives dynamically to be allocated by cloud service brokers. We denote the set of applications as $\mathcal{A} = \{a_1, a_2, \cdots, a_{|\mathcal{A}|}\}$, where $|\mathcal{A}|$ is the number of applications that arrived during the given time period T. The arrival time of a_l is denoted by t_l. We use a set of user centres $\mathcal{U} = \{U_1, \cdots, U_k, \cdots, U_{|\mathcal{U}|}\}$ to represent global user centres. The request rate $\lambda_{lk}(T)$ is the number of requests made from the user centre U_k to the application a_l during T. Thus, the total workload W_l of a_l during T is:

$$W_l(T) = \sum_{k=1}^{|\mathcal{U}|} \lambda_{lk}(T). \tag{1}$$

The set of VM types is $\mathcal{M} = \{M_0, \cdots, M_i, \cdots, M_{|\mathcal{M}|}\}$. The locations of the data centres that provide the VMs are represented by the set $\mathcal{D} = \{D_0, \cdots, D_j, \cdots, D_{|\mathcal{D}|}\}$. The unit price of M_i provided by D_j is denoted by c_{ij}. The deployment cost of a_l in a new VM is $C_l(T) = c_{ij} \cdot o_l$, where o_l is the number of hours remaining during T after deployment. If a_l is deployed in an existing VM, $C_l(T) = 0$. We assume that all VM types are available in D_j. A rented VM is represented as V_{ijn}, where i implies the VM is of type M_i, j signifies that the VM is located in D_j, and n indicates the VM is the n-th instance of type M_i in D_j. The total number of VMs rented during T is $|\mathcal{V}|$. As the dynamically arrived applications need to be swiftly deployed, $|\mathcal{V}|$ will keep increasing during T.

A container-based application [34] is deployed on only one VM instance. Due to the flexibility of using containers [6], one VM instance can host multiple container-based applications as long as its capacity allows. The available capacity of a VM, V_{ijn}, is denoted by μ_{ijn}, which is the maximum workload that V_{ijn} is capable of processing per time unit [39]. When a new containerised application a_l is deployed in V_{ijn}, the capacity μ_{ijn} is reduced by $W_l(T)$. To avoid system overload [29], we set the upper bound for the capacity utilization rate as 85%.

Every application has an independent queue of application requests from user centres. The average resource consumption of a long-running application is observed to be relatively stable [32]. Therefore, following many recent works [35–37, 39], we adopt a $M/M/1$ queue proposed in [49] to model the application request queue of the deployed applications. We denote the capacity of V_{ijn} by μ_{ijn}. By Little's law [23], the Average Processing Time (ARP) of an application a_l in V_{ijn} is:

$$\text{ARP}_{ijnl} = \frac{1}{\mu_{ijn} - W_l(T)}. \tag{2}$$

The set of applications hosted in V_{ijn} is denoted by $\mathcal{A}_{ijn} = \{a_l, \cdots, a_r\}$ and their workloads are denoted by $\Lambda_{ijn}(T) = \{W_l(T), \cdots,$

Table 1: Mathematical notation

Notation	Definition
T	The given time period
$\mathcal{A}$	The set of dynamically arrived applications
$\mathcal{U}$	The set of user centres
$\mathcal{D}$	The set of data centres
$\mathcal{M}$	The set of VM types
M_i	The i-th VM type in $\mathcal{M}$
U_k	The k-th user centre in $\mathcal{U}$
a_l	The l-th application in $\mathcal{A}$
t_l	The arrival time of a_l
D_j	The j-th data centre in $\mathcal{D}$
$\lambda_{lk}(T)$	The request rate from U_k to a_l during T
c_{ij}	The unit price of M_i in D_j
o_l	Remaining hours during T, starting from t_l
C_l	The deployment cost of a_l
V_{ijn}	The n-th VM instance of type M_i in D_j
$W_l(T)$	The workload of a_l during T
μ_{ijn}	The capacity of V_{ijn}
ARP_{ijnl}	The average processing time of a_l in V_{ijn}
$\mathcal{A}_{ijn}$	The applications deployed in V_{ijn}
$\Lambda_{ijn}(T)$	The workloads of $\mathcal{A}_{ijn}$ during T
$\mathrm{ANL}_l(T)$	The average network latency of a_l during T
ART_l	The average response time of a_l
π	The reinforcement learning policy
h	The h-th step in RL
S_h	The state at step h in RL
A_h	The action to be taken at step h in RL
R_h	The reward at step h in RL
$Q_\pi(S_h, A_h)$	The Q-value of S_h and A_h given π
δ_{h-1}	The TD error at step $(h-1)$

$W_r(T)\}$. The total workload $W_{ijn}(T)$ in V_{ijn} is the sum of all the application workloads in it, defined as:

$$W_{ijn}(T) = \sum_{W_l(T) \in \Lambda_{ijn}(T)} W_l(T). \tag{3}$$

Each application in the hosted VM is assigned a proportion of the VM capacity according to its workload. For example, the ARP of a_l in V_{ijn} is calculated by:

$$\mathrm{ARP}_{ijnl} = \frac{1}{W_l(T)/W_{ijn}(T) \cdot (\mu_{ijn} - W_{ijn}(T))}. \tag{4}$$

To satisfy the assumption of the $M/M/1$ queue model, we have the constraint that

$$\mu_{ijn} > W_{ijn}(T). \tag{5}$$

This constraint ensures that the workload in V_{ijn} will never exceed its capacity.

The Average Network Latency (ANL) of a_l deployed in data centre D_j during T is defined as:

$$\mathrm{ANL}_l(T) = \frac{\sum_{k=1}^{|\mathcal{U}|} \lambda_{lk}(T) I_{jk}}{W_l(T)}, \tag{6}$$

where I_{jk} is the network latency between the data centre D_j and the user centre U_k. We can now define the Average Response Time (ART) of a_l during T as the sum its ARP and ANL:

$$\mathrm{ART}_l = \mathrm{ANL}_l(T) + \mathrm{ARP}_{ijnl}. \tag{7}$$

We formulate the LADOAP problem in this paper as a single-objective optimisation problem to accommodate flexible weighting schemes to different objectives. The objective function of deploying a given set of applications $\mathcal{A}$ during T is defined as:

$$f_{\mathcal{A}}(T) = \sum_{l=1}^{|\mathcal{A}|} \left(\widetilde{\mathrm{ART}}_l \cdot w_1 + \widetilde{C}_l \cdot w_2 \right), \tag{8}$$

where $\widetilde{\mathrm{ART}}_l$ and $\widetilde{C}_l$ are normalised ART and $C_l(T)$, and w_1 and w_2 are the weights for ARTs and deployment costs, respectively. $w_1 \in [0,1]$ and $w_2 \in [0,1]$, $w_1 + w_2 = 1$. The normalisation is performed to scale ARTs and costs with their maximum and minimum values so that they are within the range from 0 to 1. For example, the normalisation formula for ART_l is:

$$\widetilde{\mathrm{ART}}_l = \frac{\mathrm{ART}_l - \min(\mathrm{ART})}{\max(\mathrm{ART}) - \min(\mathrm{ART})}, \tag{9}$$

where $\min(\mathrm{ART})$ is the minimum ART and $\max(\mathrm{ART})$ is the maximum ART. $\widetilde{C}_l(T)$ can be derived similarly to (9). Our aim is to assign VMs to geo-distributed data centres to host the arrived applications in $\mathcal{A}$ with minimised ARTs and costs, which is defined as:

$$\min \quad f_{\mathcal{A}}(T). \tag{10}$$

In the next sections, we introduce how to use an RL-assisted GPHH algorithm to obtain a VM selection rule based on predicted priority scores to solve the problem. During training, RL will learn a surrogate model to facilitate the evaluation of rules in GPHH.

4 PROPOSED METHOD

An overview of our proposed GPHH-RL algorithm is demonstrated in Figure 2. GPHH-RL utilises an RL-based surrogate model to efficiently evaluate rules generated by GPHH with much fewer simulations while maintaining the effectiveness.

4.1 GPHH-RL for LADOAP

We propose a novel hybrid approach GPHH-RL for LADOAP problems, which enhances efficient learning of GPHH with RL-based surrogate models. The computationally expensive simulations in LADOAP problems prevent GPHH from having a more efficient heuristic evolution. Employing a surrogate model can substantially reduce the number of simulations required for generating good heuristic rules in LADOAP problems. In GPHH-RL, Q-functions act as surrogate models by using the expressive power of GP trees.

The terminal and function sets utilised in GPHH-RL is listed in Table 2. The terminal set contains the workload $W_l(T)$, the cost c_ij, the average response time ART_l, the capacity μ_{ijn}, the arrival time t_l, and the ephemeral constants 1 and -1. The function set contains addition, subtraction, multiplication, analytic quotient, negative, maximum, and minimum. The negative function simply multiplies the input by -1. Sine and cosine are also included in the function set to enhance the universal approximation power of GP [57].

The overview of the GPHH-RL algorithm is shown in Figure 2 and the pseudo-code for GPHH-RL is described in Algorithm 1.

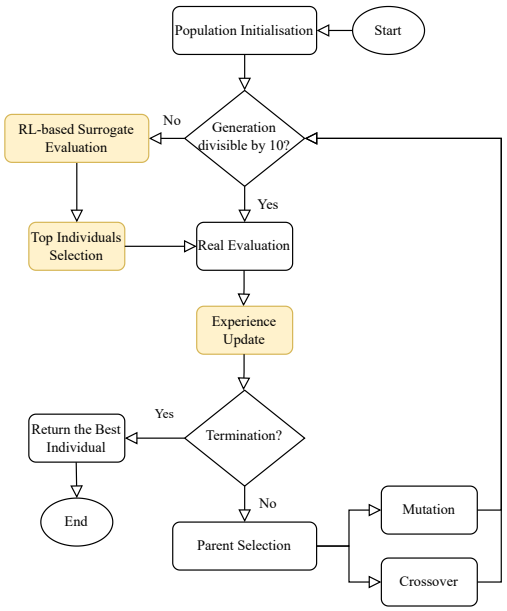

Figure 2: Overview of the GPHH-RL algorithm.

We first initialise the population of size N with the ramped-half-and-half strategy [22]. The population is evaluated by simulations to determine real fitness. Simulations generate experience as illustrated in Algorithm 2. The experience of a heuristic rule p contains lists of selected VMs, rewards, costs and the Q-table associated with it. We store a batch of the best experiences by their real fitness in each generation. The batch size N_e is called the experience size in Table 5. According to the real evaluation period in Table 5, full simulation is performed for all the population individuals intermittently in certain generations. The experience stored is updated with the best N_e heuristic rules according to their real fitness. The update replaces experience entries of lower fitness with new ones, but the size of the experience is maintained. In other generations, surrogate models are used for the TD evaluation. TD-Eval$(\cdot)$ takes an experience and a heuristic rule as inputs. It calculates the TD error by following (13). In this work, we always take the best experience for the TD evaluation. Therefore, the replay size is 1. The heuristic rules are first sorted by their TD errors. The best N_e heuristic rules are evaluated by their real fitness to update the stored experience.

Evolutionary procedures in GPHH-RL are standard. At the beginning of each generation, parents are selected through tournament selection. The evolutionary operators consist of mutation and crossover. For selected parents, mutation or crossover is conducted. In other words, mutation and crossover will not be conducted together for the same selected individuals. After retaining the best individuals, the generated offspring replace the rest of the original population at the end of the generation. The iterative evolutionary procedures continue until the specified total number of generations, Gen, has been met.

The procedure of VM selection is explained in Algorithm 3. We first examine existing VMs and assign priority scores to them. The available new VMs in each data centre are also assigned priority scores. The VM with the highest priority score is selected as the VM to deploy the arrived application. If the selected VM belongs to the existing pool of VMs, the reward only considers ART as the cost is 0. If the remaining capacity of the selected VM is below the minimum capacity threshold, it is removed from the existing VM pool. For newly rented VMs, the reward takes account of both ART and cost. If the remaining capacity of the newly rented VM is above the minimum capacity threshold, it is added into the pool of VMs.

Table 2: Terminal and function sets for GPHH-RL

Terminal	Description
Workload	The workload of an application during T
Cost	The deployment cost of a VM during T
ART	The ART of an application during T
Capacity	The available capacity of a VM
Arrival	The arrival time of an application
Constant	1 and -1
Function	**Description**
+, -, ×, negative, cosine, sine, max, min	Standard arithmetic operations
Analytic quotient	$\frac{x_1}{\sqrt{1+x_2^2}}$

Algorithm 1 GPHH-RL for LADOAP

```
1:  procedure GPHH-RL(𝒜, P_0)
2:      P ← P_0 = {p_1^0, ··· , p_N^0}
3:      expList ← []
4:      for r ← 1 to N do
5:          exp ← Simulation(𝒜,P[r])
6:          expList updates exp
7:      for g ← 1 to Gen do
8:          P ← TournamentSelection(P)
9:          P′ ← Evolution(P)
10:         if g%10 = 0 then
11:             for r ← 1 to N do
12:                 exp ← Simulation(𝒜,P′[r])
13:                 expList updates exp
14:         else
15:             for r ← 1 to N do
16:                 TD-Eval(expList[0],P′[r])
17:             P′ ← sort(P′)
18:             for r ← 1 to N_e do
19:                 exp ← Simulation(𝒜,P′[r])
20:                 expList updates exp
21:         P ← P′
22:     p_best ← pick the best individual from P
23:     return p_best
```

Algorithm 2 Simulation in LADOAP

```
procedure SIMULATION(𝒜, p)
    R_sum ← 0
    Q-table ← []
    VMpool ← []
    VMlist ← []
    Rlist ← []
    Clist ← []
    for l ← 1 to |𝒜| do
        VMpool, R_l, Q_l, vm, c ← VMselect(p, 𝒜[l], VMpool)
        Q-table appends Q_l
        Rlist appends R_l
        VMlist appends vm
        Clist appends c
        R_sum ← R_sum + R_l
    fitness ← R_sum/|𝒜|
    p.update(fitness)
    exp ← (VMlist, Rlist, Clist, Q-table)
    return exp
```

Algorithm 3 The procedure of selecting VM

```
procedure VMSELECT(p_r, a_l, VMpool)
    VMselected ← None
    scoreMax ← −99999
    Q-value ← 0
    cost ← 0
    for vm in VMpool do
        score ← p_r(vm, a_l)
        if scoreMax < score then
            scoreMax ← score
            VMselected ← vm
    for vm-new in 𝒟 do
        score ← p_r(vm-new, a_l)
        if scoreMax < score then
            scoreMax ← score
            VMselected ← vm-new
    Q-value ← scoreMax
    if VMselected is vm then
        vm.capacity ← vm.capacity − W_l(T)
        if vm.capacity < μ_min then
            VMpool removes vm
        R_l ← −ÃRT_l · w_1 − 0
    else
        vm-new.capacity ← vm-new.capacity − W_l(T)
        if vm-new.capacity > μ_min then
            VMpool appends vm-new
        R_l ← −ÃRT_l · w_1 − C̃_l · w_2
        cost ← C̃_l
    return VMpool, R_l, Q-value, VMselected, cost
```

4.2 RL-Based Surrogate Model

Temporal Difference (TD) learning is a fundamental concept within the realm of reinforcement learning, which combines the principles of Monte Carlo approaches and dynamic programming [43]. TD methods iteratively update the Q-function by learning from experiences collected directly from the interaction with environments. As a result, the TD error, i.e., the prediction error between two adjacent states, can be minimised [43].

In the context of RL, we first define states in our GPHH-RL algorithm. The state S_h is a feature vector of attributes used for making VM selection decisions at step h. Each attribute is represented by a separate terminal type used in GP, as listed in Table 2. An action A_h corresponds to the decision of selecting any specific VM in step h. The RL policy π is a mapping from a given state S_h to an action A_h. The Q-function of the policy π and the state-action pair (S_h, A_h) is denoted by $Q_\pi(S_h, A_h)$. It is defined as the expected cumulative reward during T recursively:

$$Q_\pi(S_h, A_h) = \mathop{\mathrm{E}}_{S_{h+1}}\left[R_{h+1} + \max_A Q_\pi(S_{h+1}, A)\right], \tag{11}$$

where R_{h+1} is the reward obtained from step h, and A is an action in the action space. In GPHH-RL, $Q_\pi(S_h, A_h)$ is expressed through GP trees. The reward R_h is defined as:

$$R_h = -\widetilde{\mathrm{ART}}_h \cdot w_1 - \widetilde{C}_l \cdot w_2. \tag{12}$$

The maximised cumulative reward will jointly minimise ARTs and deployment costs of all applications.

We can now define the TD error at step $(h-1)$ as:

$$\delta_{h-1} = \left|R_h + \max_A Q_\pi(S_h, A) - Q_\pi(S_{h-1}, A_{h-1})\right|. \tag{13}$$

The TD error δ_{h-1} is the non-negative estimation error of $Q(S_{h-1}, A_{h-1})$ that is only known at the next step h, since $Q_\pi(S_h, A)$ is required to calculate δ_{h-1}. We start by calculating the TD error from $h = 1$ with $h \in \{1, \cdots, |\mathcal{A}| - 1\}$.

Using the TD error defined in (13) as the fitness to be minimised for the GP individuals, the estimation of the Q-values between adjacent states will become more and more accurate through the evolutionary optimisation process. Naturally, the Q-functions can be used as surrogate models for estimating the quality of heuristic rules according to its definition.

5 EXPERIMENTAL EVALUATION

5.1 Dataset

Our VM pricing scheme is collected from AWS [3] in January 2024, as shown in Table 3. Five M6g VM types are considered in each data centre. For test scenarios with 8 data centres, the data centres are located at Northern Virginia, Northern California, Dublin, Singapore, Tokyo, Sydney, Sao Paulo and Mumbai. For test scenarios with 15 data centres, the 7 extra data centres are located at London, Paris, Frankfurt, Stockholm, Hong Kong, Seoul, and central Canada. Our 81 global user centres are from 34 countries on 6 continents based on the simulation in the Sprint IP backbone network databases [37].

In our simulation, we group user centres into 6 continents. Each application is assumed to have a focused continent. We randomly pick 10 to 20 extra user centres from other continents. The focused continent occupies about 50-80% of the total application workload. The rest all of the workload is shared by user centres from other continents. The proportion of the workload for each user centre is determined by its population size. The workload of each application is ranged from 52 to 304 [37].

We design 8 different test scenarios to test our algorithm. The scenarios are listed in Table 4. Four scenarios have 15 data centres, and the other 4 have 8 data centres. The training scenario has 300 applications and 15 data centres. The best heuristic rule is derived from the training scenario and tested on all the scenarios.

Table 3: AWS M6g VM Configurations in N. Virginia, USA

VM type	CPU	Hourly rate
medium	1	$0.0385
large	2	$0.77
xlarge	4	$0.154
2xlarge	8	$0.308
4xlarge	16	$0.616

Table 4: Test scenarios

Scenario	Applications	Data Centre
1	300	8
2	250	8
3	200	8
4	150	8
5	300	15
6	250	15
7	200	15
8	150	15

5.2 Baselines

As previous studies on containerised application deployment in clouds do not simultaneously consider dynamic online deployment of applications that can share VMs, we adapted two algorithms in similar problems as our main baselines for LADOAP problems. The first is a co-evolutionary GPHH algorithm for solving RAC problems [46]. This work has a two-level selection process: the VM selection and the Physical Machine (PM) selection. For a fair comparison, we first adapt it by replacing the original terminals in the VM selection level with ones relevant in LADOAP problems: workload, cost and available capacity. Second, since the PM selection part is irrelevant, we assign VMs to geo-distributed data centres with a greedy algorithm. The selection of data centre is determined by the lowest ANL. The population size in GPHH is changed to 1024 since it does not have two populations any more. This algorithm is termed as GPHH-DC.

The second baseline is a Best-Fit algorithm adapted to LODOAP problems [1, 9]. Best-Fit algorithms are classic but popular approaches for solving bin-packing problems [28], which can be formulated into both RAC and LADOAP problems. The Best-Fit algorithm deals with two level selection problems: VM selection and Data centre selection. The algorithm will first try to find a VM with with the fastest ARP in the existing VM pool. If no available VM is found, the algorithm will rent a new VM in the data centre with the smallest ANL. The new VM should have the minimal cost and ARP in the chosen data centre.

5.3 Parameter Settings

The parameter settings in GPHH-RL include parameters for both GP and RL algorithms. They are listed in Table 5. GPHH-RL was implemented in DEAP [12].

Table 5: Parameter Settings

Parameter	Description
Initialisation	ramped-half-and-half
Initial depth	from 1 to 2
Crossover rate	90%
Mutation rate	10%
Mutated component depth	from 0 to 2
Maximum depth	8
Number of generations	100
Populations size	1024
Selection	tournament (size = 5)
Hall of fame	1
Experience size	128
Replay size	1
Real evaluation period	every 10 generations

5.4 Main Results

Table 6 demonstrates the main results of GPHH-RL in 8 different scenarios, compared with GPHH-DC and Best-Fit. We conducted 30 independent runs for all algorithms. From the table, it is obvious that GPHH-RL outperforms two other baseline algorithms in all scenarios. We applied Wilcoxon signed rank tests with a significance level of 0.05 to the main results. The null hypothesis is that there is no difference between the results of GPHH-RL and other baselines. It turns out that the null hypothesis can be rejected.

We can also observe that both GPHH-RL and GPHH-DC are much better than Best-Fit. This makes sense as the heuristic in Best-Fit is a manually designed greedy algorithm that can easily get stuck in a local optimum. In contrast, GPHH-RL and GPHH-DC search for heuristics evolutionarily, without being confined by human experience. They are more likely to explore heuristics that are closer to the global optima.

Another crucial benefit of GPHH-RL is the reduced number of simulations. In the current parameter settings listed in Table 5, GPHH-RL has 10 generations of full simulation evaluations. For the other 90 generations, each generation would only require 128 simulations for the experience update. This would save $(1024-128)\cdot 90 = 80640$ simulations in 100 generations. Out of 102400 total simulations, GPHH-RL can reduce up to 78.75% of the simulations. Generally speaking, cloud computing simulations involve complex procedures such as communication between components that would prolong execution time [24]. In contrast, our surrogate model only calculates the TD errors between Q-values and avoids the execution time consumed by the simulation procedures. The saving of the simulation time can either be used to speed up the optimisation process, or maneuver computational resources to some bottleneck points to further enhance the optimisation performance.

Table 6: Test results in 8 different scenarios

Scenario	GPHH-RL	GPHH-DC	Best-Fit
1	**0.1246 ± 0.0003**	0.1267 ± 0.00007	0.1874
2	**0.1247 ± 0.0003**	0.1268 ± 0.00009	0.1915
3	**0.1275 ± 0.0004**	0.1293 ± 0.00010	0.1850
4	**0.1249 ± 0.0003**	0.1263 ± 0.00010	0.1848
5	**0.1215 ± 0.0003**	0.1238 ± 0.00007	0.1822
6	**0.1216 ± 0.0003**	0.1239 ± 0.00009	0.1880
7	**0.1243 ± 0.0005**	0.1262 ± 0.00010	0.1860
8	**0.1216 ± 0.0003**	0.1232 ± 0.00010	0.1816

Table 7: Test results compared with multi-objective GPHH

Scenario	GPHH-RL	MO-GPHH
1	**0.1246 ± 0.0003**	0.1323
2	**0.1247 ± 0.0003**	0.1323
3	**0.1275 ± 0.0004**	0.1348
4	**0.1249 ± 0.0003**	0.1325
5	**0.1215 ± 0.0003**	0.1289
6	**0.1216 ± 0.0003**	0.1288
7	**0.1243 ± 0.0005**	0.1312
8	**0.1216 ± 0.0003**	0.1289

5.5 Further Analyses

To have a deeper insight into the heuristic designed by GPHH-RL, we examine one of the best heuristics in Figure 3. One thing to notice is that even though we have 6 terminal types, only 5 are used in this heuristic. The missing one is the workload. As workload can be utilised to obtain ART, it is possibly true that having ART alone is sufficient enough for constructing good heuristics. There are also unutilised functions such as max and min. This suggests that we may have redundant terminal and function types in our GPHH design. By reducing unused terminals and functions, we may have more effectively evolved heuristics.

Another observation is that cosine function is used heavily in this heuristic. The heuristic can be expressed as a ratio between two cosine functions. As cosine is a periodic function that regularise outputs from -1 to 1, the output range of this heuristic is bounded. We will investigate whether famous neural network activation functions like sigmoid would have a similar effect in the evolution of GPHH algorithms in our future work.

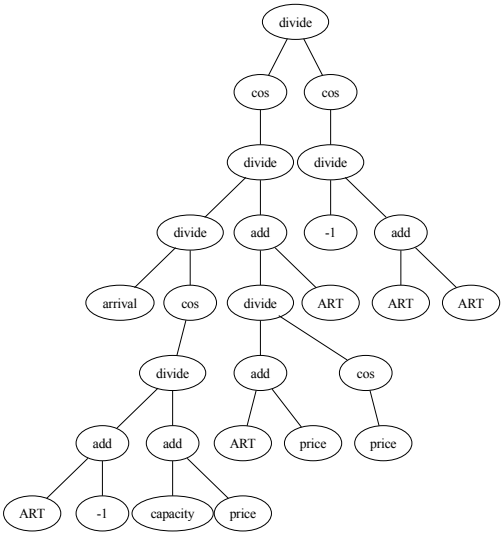

Figure 3: Tree representation of the best VM selection rule.

We make further comparisons with a multi-objective GPHH algorithm designed for RAC problems [8]. The results are shown in Table 7. Similar to GPHH-DC, we replaced the original RAC-related terminals to be the same as what used in GPHH-RL and name the adapted algorithm as MO-GPHH. One key difference in MO-GPHH is that the original GPHH algorithm in [8] does not support multiple containers in one VM and we kept it that way. After obtaining a Pareto front of heuristics, we convert them into single objective fitness based on (8). The heuristic with the best converted single objective fitness is employed for testing scenarios. We notice that the heuristics obtained from MO-GPHH are quite stable, with negligible standard deviation. It is obvious that GPHH-RL outperforms MO-GPHH significantly. This indicates the effectiveness of supporting multiple containers in one VM in LADOAP problems.

6 CONCLUSIONS

The overarching goal of this paper is to improve the efficiency of GPHH algorithms for LADOAP problems while maintaining the effectiveness. By combining GPHH with an RL-based surrogate model, we achieve this goal successfully. A novel GPHH-RL algorithm is proposed to solve the dynamic application deployment in clouds in an online manner.

The experimental results show that GPHH-RL can significantly reduce the number of simulations without hurting the performance. Further analyses indicate that GPHH-RL can learn insightful heuristics without using all the terminals and functions, suggesting that there is room to improve the terminal and function representation in this work.

This paper is the first paper employing GP-based Q-function to approximate objective functions in surrogate models. We expect that the fast-developing field of RL-assisted GP will get inspiration from this paper and advance further. We would like to investigate the comprehensive benefits of using RL-based surrogate models in GP in the near future.

ACKNOWLEDGMENTS

The authors wish to acknowledge the use of New Zealand eScience Infrastructure (NeSI) high performance computing facilities. This work was partly supported under Grant VUW-FSRG-10114, administered by Victoria University of Wellington.

REFERENCES

[1] Susanne Albers, Arindam Khan, and Leon Ladewig. 2021. Best fit bin packing with random order revisited. *Algorithmica* 83 (2021), 2833–2858.

[2] Yasser Aldwyan, Richard O Sinnott, and Glenn T Jayaputera. 2021. Elastic deployment of container clusters across geographically distributed cloud data centers for web applications. *Concurrency and Computation: Practice and Experience* 33, 21 (2021), e6436.

[3] Amazon. 2024. Amazon EC2 On-Demand Pricing. https://aws.amazon.com/ec2/pricing/on-demand/. (accessed Jan. 29, 2024).

[4] Zahra Beheshti and Siti Mariyam Hj Shamsuddin. 2013. A review of population-based meta-heuristic algorithms. *Int. j. adv. soft comput. appl* 5, 1 (2013), 1–35.

[5] Thad Benjaponpitak, Meatasit Karakate, and Kunwadee Sripanidkulchai. 2020. Enabling live migration of containerized applications across clouds. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2529–2538.

[6] David Bernstein. 2014. Containers and cloud: From lxc to docker to kubernetes. *IEEE cloud computing* 1, 3 (2014), 81–84.

[7] Edmund K Burke, Mathew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R Woodward. 2009. Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence: Collaboration, fusion and emergence*. Springer, 177–201.

[8] Yuheng Chen, Tao Shi, Hui Ma, and Gang Chen. 2022. Automatically design heuristics for multi-objective location-aware service brokering in multi-cloud. In *IEEE International Conference on Services Computing (SCC)*. IEEE, 206–214.

[9] Edward G Coffman, János Csirik, Gábor Galambos, Silvano Martello, Daniele Vigo, et al. 2013. Bin Packing Approximation Algorithms: Survey and Classification. In *Handbook of combinatorial optimization*. Springer, 455–531.

[10] Kenneth De Jong. 2017. Evolutionary computation: a unified approach. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 373–388.

[11] Ke-Lin Du, MNS Swamy, et al. 2016. *Search and optimization by metaheuristics*. Springer.

[12] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research* 13, 1 (2012), 2171–2175.

[13] Gartner. 2023. Gartner Says Cloud Will Become a Business Necessity by 2028. https://www.gartner.com/en/newsroom/press-releases/2023-11-29-gartner-says-cloud-will-become-a-business-necessity-by-2028. (accessed Dec. 01, 2023).

[14] Yangyang Guo, Hao Wang, Lei He, Witold Pedrycz, PN Suganthan, and Yanjie Song. 2023. *A Reinforcement Learning-assisted Genetic Programming Algorithm for Team Formation Problem Considering Person-Job Matching*. arXiv preprint arXiv:2304.04022.

[15] Red Hat. 2024. Red Hat OpenShift. https://www.redhat.com/en/technologies/cloud-computing/openshift. (accessed Jan. 29, 2024).

[16] Daniel Hein, Steffen Udluft, and Thomas A Runkler. 2018. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence* 76 (2018), 158–169.

[17] Torsten Hildebrandt and Jürgen Branke. 2015. On using surrogates with genetic programming. *Evolutionary computation* 23, 3 (2015), 343–367.

[18] Yaochu Jin, Handing Wang, Tinkle Chugh, Dan Guo, and Kaisa Miettinen. 2018. Data-driven evolutionary optimization: An overview and case studies. *IEEE Transactions on Evolutionary Computation* 23, 3 (2018), 442–458.

[19] Michael I Jordan and Tom M Mitchell. 2015. Machine learning: Trends, perspectives, and prospects. *Science* 349, 6245 (2015), 255–260.

[20] Ali Kanso and Alaa Youssef. 2017. Serverless: beyond the cloud. In *Proceedings of the 2nd International Workshop on Serverless Computing*. ACM, 6–10.

[21] Ioannis Korontanis, Antonios Makris, and Konstantinos Tserpes. 2024. A Survey on Modeling Languages for Applications Hosted on Cloud-Edge Computing Environments. *Applied Sciences* 14, 6 (2024), 2311.

[22] JR Koza. 1992. Genetic Programming: on the programming of computers by means of natural selection.

[23] John DC Little and Stephen C Graves. 2008. Little's law. In *Building intuition: insights from basic operations management models and principles*, Dilip Chhajed and Timothy J. Lowe (Eds.). Springer.

[24] Najme Mansouri, R Ghafari, and B Mohammad Hasani Zade. 2020. Cloud computing simulators: A comprehensive review. *Simulation Modelling Practice and Theory* 104 (2020), 102144.

[25] Marwa F Mohamed. 2016. Service replication taxonomy in distributed environments. *Service Oriented Computing and Applications* 10 (2016), 317–336.

[26] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. *Foundations of machine learning*. MIT press.

[27] Terrell Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel faissol, and Brenden K Petersen. 2021. Symbolic Regression via Deep Reinforcement Learning Enhanced Genetic Programming Seeding. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 24912–24923.

[28] Aniket Murhekar, David Arbour, Tung Mai, and Anup Rao. 2023. *Dynamic Vector Bin Packing for Online Resource Allocation in the Cloud*. arXiv preprint arXiv:2304.08648.

[29] Seyed Mohammad Reza Nouri, Han Li, Srikumar Venugopal, Wenxia Guo, MingYun He, and Wenhong Tian. 2019. Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. *Future Generation Computer Systems* 94 (2019), 765–780.

[30] Swarna Kamal Paul and Parama Bhaumik. 2020. A reinforcement learning agent based on genetic programming and universal search. In *4th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, 122–128.

[31] Rancher. 2024. Enterprise Kubernetes Management. https://www.rancher.com/. (accessed Jan. 29, 2024).

[32] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the third ACM symposium on cloud computing*. Association for Computing Machinery, 1–13.

[33] Fabiana Rossi, Valeria Cardellini, Francesco Lo Presti, and Matteo Nardelli. 2020. Geo-distributed efficient deployment of containers with Kubernetes. *Computer Communications* 159 (2020), 161–174.

[34] Fabiana Rossi, Matteo Nardelli, and Valeria Cardellini. 2019. Horizontal and vertical scaling of container-based applications using reinforcement learning. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 329–338.

[35] Tao Shi, Hui Ma, and Gang Chen. 2020. Seeding-based multi-objective evolutionary algorithms for multi-cloud composite applications deployment. In *2020 IEEE International Conference on Services Computing (SCC)*. IEEE, 240–247.

[36] Tao Shi, Hui Ma, Gang Chen, and Sven Hartmann. 2020. Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment. *IEEE Transactions on Parallel and Distributed Systems* 31, 8 (2020), 1954–1969.

[37] Tao Shi, Hui Ma, Gang Chen, and Sven Hartmann. 2021. Cost-effective web application replication and deployment in multi-cloud environment. *IEEE Transactions on Parallel and Distributed Systems* 33, 8 (2021), 1982–1995.

[38] Tao Shi, Hui Ma, Gang Chen, and Sven Hartmann. 2021. Location-aware and budget-constrained service brokering in multi-cloud via deep reinforcement learning. In *Service-Oriented Computing: 19th International Conference, ICSOC*. Springer, 756–764.

[39] Tao Shi, Hui Ma, Gang Chen, and Sven Hartmann. 2023. Auto-Scaling Containerized Applications in Geo-Distributed Clouds. *IEEE Transactions on Services Computing* 16 (2023), 4261–4274.

[40] Vindeep Singh and Sateesh K Peddoju. 2017. Container-based microservice architecture for cloud applications. In *International Conference on Computing, Communication and Automation (ICCCA)*. IEEE, 847–852.

[41] Sarra Slimani, Tarek Hamrouni, and Faouzi Ben Charrada. 2021. Service-oriented replication strategies for improving quality-of-service in cloud computing: a survey. *Cluster Computing* 24 (2021), 361–392.

[42] Yanjie Song, Yutong Wu, Yangyang Guo, Ran Yan, Ponnuthurai Nagaratnam Suganthan, Yue Zhang, Witold Pedrycz, Yingwu Chen, Swagatam Das, Rammohan Mallipeddi, et al. 2023. *Reinforcement Learning-assisted Evolutionary Algorithm: A Survey and Research Opportunities*. arXiv preprint arXiv:2308.13420.

[43] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[44] Boxiong Tan, Hui Ma, and Yi Mei. 2018. A genetic programming hyper-heuristic approach for online resource allocation in container-based clouds. In *AI 2018: Advances in Artificial Intelligence: 31st Australasian Joint Conference*. Springer, 146–152.

[45] Boxiong Tan, Hui Ma, and Yi Mei. 2019. A hybrid genetic programming hyper-heuristic approach for online two-level resource allocation in container-based clouds. In *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2681–2688.

[46] Boxiong Tan, Hui Ma, Yi Mei, and Mengjie Zhang. 2020. A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds. *IEEE Transactions on Cloud Computing* 10, 3 (2020), 1500–1514.

[47] Laszlo Toka, Gergely Dobreff, Balazs Fodor, and Balazs Sonkoly. 2020. Adaptive AI-based auto-scaling for Kubernetes. In *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 599–608.

[48] Mathurin Videau, Alessandro Leite, Olivier Teytaud, and Marc Schoenauer. 2022. Multi-objective genetic programming for explainable reinforcement learning. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 278–293.

[49] Jordi Vilaplana, Francesc Solsona, Ivan Teixidó, Jordi Mateo, Francesc Abella, and Josep Rius. 2014. A queuing theory model for cloud computing. *The Journal of Supercomputing* 69 (2014), 492–507.

[50] Xili Wan, Xinjie Guan, Tianjing Wang, Guangwei Bai, and Baek-Yong Choi. 2018. Application deployment using Microservice and Docker containers: Framework and optimization. *Journal of Network and Computer Applications* 119 (2018), 97–109.

[51] Chen Wang, Hui Ma, Gang Chen, Victoria Huang, Yongbo Yu, and Kameron Christopher. 2023. Energy-Aware Dynamic Resource Allocation in Container-Based Clouds via Cooperative Coevolution Genetic Programming. In *International*

Conference on the Applications of Evolutionary Computation (Part of EvoStar). Springer, 539–555.

[52] Handing Wang, Yaochu Jin, and Jan O Jansen. 2016. Data-driven surrogate-assisted multiobjective evolutionary optimization of a trauma system. *IEEE Transactions on Evolutionary Computation* 20, 6 (2016), 939–952.

[53] Handing Wang, Yaochu Jin, Chaoli Sun, and John Doherty. 2018. Offline data-driven evolutionary optimization using selective surrogate ensembles. *IEEE Transactions on Evolutionary Computation* 23, 2 (2018), 203–216.

[54] Sheng Wang, Zhijun Ding, and Changjun Jiang. 2020. Elastic scheduling for microservice applications in clouds. *IEEE Transactions on Parallel and Distributed Systems* 32, 1 (2020), 98–115.

[55] Tonghao Wang, Xingguang Peng, Tao Wang, Tong Liu, and Demin Xu. 2024. Automated design of action advising trigger conditions for multiagent reinforcement learning: A genetic programming-based approach. *Swarm and Evolutionary Computation* 85 (2024), 101475.

[56] Guohua Wu, Rammohan Mallipeddi, and Ponnuthurai Nagaratnam Suganthan. 2019. Ensemble strategies for population-based optimization algorithms–A survey. *Swarm and evolutionary computation* 44 (2019), 695–711.

[57] Xin Yao. 1999. Universal approximation by genetic programming. In *Foundations of Genetic Programming*. Springer Berlin, Heidelberg, 66–67.

[58] Wenxing Ye, Weiying Feng, and Suohai Fan. 2017. A novel multi-swarm particle swarm optimization with dynamic learning strategy. *Applied Soft Computing* 61 (2017), 832–843.

[59] Fangfang Zhang, Yi Mei, Su Nguyen, Kay Chen Tan, and Mengjie Zhang. 2022. Instance rotation based surrogate in genetic programming with brood recombination for dynamic job shop scheduling. *IEEE Transactions on Evolutionary Computation* 27, 5 (2022), 1192–1206.

[60] Fangfang Zhang, Yi Mei, Su Nguyen, Mengjie Zhang, and Kay Chen Tan. 2021. Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation* 25, 4 (2021), 651–665.