Leveraging LLM in Genetic Programming Hyper-Heuristics for Dynamic Microservice Deployment

Zhengxin Fang[1]✉, Hui Ma[1], Gang Chen[1], Sven Hartmann[2], and Chen Wang[3]

[1] Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, New Zealand
{zhengxin.fang, hui.ma, aaron.chen}@ecs.vuw.ac.nz
[2] Department of Informatics, Clausthal University of Technology, Germany
sven.hartmann@tu-clausthal.de
[3] National Institute of Water and Atmospheric Research, New Zealand
chen.wang@niwa.co.nz

Abstract. Microservice deployment in cloud computing is a challenging combinatorial optimization problem due to the complex dependencies among microservices and the intricate trade-offs among different QoS requirements, e.g., minimizing Energy Consumption (EC) vs. minimizing Communication Overhead (CO). Recently, some hyper-heuristics methods, particularly Genetic Programming Hyper-Heuristics (GPHH), have been proposed to automatically generate heuristics for solving dynamic microservice deployment problems. Meanwhile, Large Language Models (LLMs) are becoming popular for solving various domain-specific problems thanks to their strong ability to learn problem-related knowledge. However, hybridizing GPHH with LLM by combining their abilities in solving complex optimization problems remains unexplored. In this paper, we propose an LLM-enhanced Genetic Programming Hyper-Heuristic (LLM-GPHH) algorithm to evolve heuristics for the dynamic deployment of applications composed of microservices, to jointly optimize EC and CO. Our experiments on real-world datasets demonstrate the effectiveness of the newly proposed LLM-GPHH.

Keywords: LLM · genetic programming · hyper-heuristics · dynamic microservice deployment · cloud computing

1 Introduction

Recent years have witnessed an increasing number of applications being developed by composing microservices due to their advantages in terms of scalability, maintainability and resilience [14]. A microservices application comprises multiple independent microservices, each responsible for a specific function and communicating with others via data transmission. Microservice applications deployed in container-based clouds become popular thanks to containers'

lightweight nature and scalability [14]. The deployment of microservice applications in container-based clouds requires deploying microservices to containers, which are then allocated to virtual machines (VMs). Subsequently, VMs are allocated to Physical Machines (PMs). The process of deploying microservices applications in container-based clouds gives rise to the problem of **M**icroservice **D**eployment in **C**ontainer-based clouds (called the MDC problem), which is NP-hard [4, 16, 17].

In real-world practices, microservice applications arrive dynamically in cloud data centers in real-time. As a result, a *VM selection heuristic* is required to either select existing VMs or create new VM instances for real-time microservice applications deployment. Similarly, a suitable PM should be selected for newly created VM instances to achieve more effective resource utilization through a *PM selection heuristic*. Improper *VM selection heuristic* and *PM selection heuristic* causes poor CPU and memory allocation, increasing the Energy Consumption (EC) in cloud data centers [2]. In addition, improper deployment of microservices increases the communication data volumes between different PMs through the physical network, resulting in a large Communication Overhead (CO) in cloud data centers, which affects the performance of microservice applications [1]. Therefore, in this paper, we aim to propose an effective method to automatically generate *VM selection heuristic* and *PM selection heuristic* to minimize both EC and CO in cloud data centers [4] for dynamic MDC problems.

Various heuristics have been proposed in the literature to solve MDC problems [7, 8, 11]. However, manually designed heuristics can lose effectiveness due to the dynamic workload arriving in clouds [20]. Hyper-heuristics (HHs), in particular Genetic Programming Hyper-Heuristics (GPHHs), have been proposed to learn heuristics automatically for many complex dynamic combinatorial optimization problems [16,17,20]. Nevertheless, the randomness of genetic operators (i.e., crossover and mutation) of GP do not explicitly exploit the patterns among high-fitness individuals [9], leading to the high possibility of producing bad individuals during evolution. In recent years, the rapid rise of Large Language Models (LLMs) has introduced new possibilities for the exploitation of GP. LLMs can take a sequence of tokens as input and generate a new sequence of tokens, leveraging a large amount of semantic knowledge during training. This means that LLMs have the potential to capture the complex patterns among tokens (i.e. nodes of the GP tree) in high-fitness GP individuals, then output new promising GP individuals by generating new tokens.

Based on the above analysis, this paper proposes a LLM-enhance GPHH (LLM-GPHH) for generating *VM selection heuristics* and *PM selection heuristics* in dynamic MDC problems to jointly optimize CO and EC. The contributions of this paper are as follows.

- A novel *LLM evolution* is proposed in LLM-GPHH to capture the complex relationship among tokens in high-fitness individuals of GPHH and generate new promising individuals to improve the performance of GPHH in terms of jointly optimizing CO and EC for dynamic MEC problems.

- We conduct an experimental evaluation of LLM-GPHH based real-world cloud dataset. Our experiments demonstrate that the proposed LLM-GPHH can effectively generate better heuristics than existing algorithms in most scenarios.
- To the best of our knowledge, this is the first approach that combines LLM with GPHH to automatically generate heuristics for dynamic MDC problems. Ablation studies are also conducted to show the effectiveness of *LLM evolution* in some scenarios.

2 Related Work

Various heuristics have been proposed to solve microservice deployment problems as they are easy to understand and apply. For example, the Best-Fit [11] heuristic deploys applications with the most suitable resource (i.e., VM or PM), which has enough capacity (CPU and memory) to host the application. To minimize the CO among PMs, the *Min-cut* [21] mechanism has been applied in [7] to partition an application into several groups of microservices, which can then be allocated to the same PMs. Resource utilization is enhanced by allocating groups of microservices to the most-loaded machine. Similar to [7], groups of containers are allocated in [8] using a concurrent container scheduling algorithm. Specifically, the process of microservice deployment in [8] is modelled as the minimum cost flow problem (MCFP) in a weighted network, on which containers with dependencies will be merged into an aggregator node to be subsequently allocated to VMs. Although the approaches in [7, 8] can optimize both CO and EC, they assume that workloads are known in advance. As a result, their methods can lose effectiveness for handling MDC problems where microservice deployment requests and workloads are arriving dynamically [17].

To automatically learn effective heuristics for dynamic microservice deployment, a hybrid GPHH based approach, named Hybrid-Evo, is proposed in [17]. This approach learns VM selection heuristics while combining the Best-Fit heuristic with its learned PM selection heuristics. Specifically, Hybrid-Evo applies Best-Fit to generate new PM instances. These two GPHH based approaches [16, 17] only consider EC, without paying attention to microservices and inter-service communication. However, existing GPHH methods do not explicitly exploit the patterns among good individuals. LLM is a potential solution to enhance the exploitation ability of GPHH. For example, Romera et al. [10] applied LLM for mathematical discovery by capturing the complex patterns in the equations.

As summarized, GPHH is particularly suitable for handling dynamic MDC problems, and LLMs can be used to enhance the exploitation ability of GPHH due to LLM's proficiency in capturing complex but promising patterns and semantic relationships in good heuristics. In this paper, we will propose a novel GPHH algorithm (i.e., LLM-GPHH) that applies LLM to enhance the abilities of GPHH to automatically generate heuristics that can effectively solve dynamic MDC problems.

3 Problem Definition

Microservice applications can be represented as Directed Acyclic Graphs (DAGs) with weighted edges [13]. Specifically, an application is denoted by $G(M, E)$ with M as a set of nodes where each $m_i \in M$ represents a microservice. Meanwhile, E stands for a set of directed edges with weights where each $e_{i,j} \in E$ represents direct data flow between m_i and m_j with its weight quantifying the corresponding communication data volume.

In line with existing research [4, 5, 15], each microservice can only run on a single container at any time. We assume that we are given a list C of containers, a set V of VM types, and a set P of PM types in a container-based cloud where containers are assigned to VM instances and VMs are assigned to PMs. We represent a container c_i by a pair $(\zeta^{cpu}(c_i), \zeta^{mem}(c_i))$ where the first denotes the CPU requirement, and the second denotes the memory requirement of the container. We represent a VM type γ_t by a tuple = ($\Omega^{cpu}(\gamma_t)$, $\Omega^{mem}(\gamma_t)$, $\pi^{cpu}(\gamma_t)$, $\pi^{mem}(\gamma_t)$), where the first two denote the CPU and memory capacity, and the last two give the CPU and memory overhead of the VM type. Every VM instance v_i belongs to one VM type γ_t. We represent a PM type τ_t by a tuple $(\Omega^{cpu}(\tau_t), \Omega^{mem}(\tau_t), E^{idle}(\tau_t), E^{full}(\tau_t))$, where $\Omega^{cpu}(\tau_t)$, $\Omega^{mem}(\tau_t)$, $E^{idle}(\tau_t)$, and $E^{full}(\tau_t)$ denote the CPU capacity, the memory capacity, the EC per time unit for the idle state, and the EC per time unit for fully loaded state, respectively. Every PM instance p_i belongs to one PM type τ_t.

Following the non-linear energy model in [3], we use Eq. (1) to express the EC of a PM instance p_i.

$$E(p_i) = E^{idle}(p_i) + (E^{full}(p_i) - E^{idle}(p_i)) \times (2\mu^{cpu}(p_i) - (\mu^{cpu}(p_i))^{1.4}) \quad (1)$$

where $E^{idle}(p_i)$ and $E^{full}(p_i)$ are the EC of the PM instance p_i when it is idle and fully loaded, respectively. $\mu^{cpu}(p_i)$ is the CPU utilization level [3] of the PM instance p_i.

Assuming that n_t PM instances are active at timestamp t, the EC at timestamp t is calculated as follows:

$$EC(t) = \sum_{i=1}^{n_t} E(p_i) \quad (2)$$

To determine CO, we use a matrix to represent the deployment of microservices to PMs. Let X denote a binary matrix, in which $x_{i,j}$ is 1 if microservice m_i is deployed on PM instance p_j, and is 0 otherwise. Another matrix A is used to quantify the communication data volume between each pair of microservices, i.e., in A the value of $a_{i,j}$ indicates the communication data volume between microservice m_i and microservice m_j.

Assuming that n_t denotes the number of active PMs at timestamp t, and m_t stands for the number of microservices deployed at timestamp t, the CO at timestamp t can be calculated through Eq. (3):

$$CO(t) = \sum_{i=1}^{n_t} \sum_{j=1}^{n_t} a_{i,j} \cdot (1 - \sum_{k=1}^{m} x_{i,k} \cdot x_{j,k}) \tag{3}$$

Note that the term $(1-\sum_{k=1}^{m} x_{i,k} \cdot x_{j,k})$ in Eq. (3) equals 1 if the microservices m_i and m_j are on different PMs, and otherwise it is 0.

In this paper, for the joint optimization of CO and EC in the dynamic MDC problem, the optimization objective (J) is defined over a time period T in Eq. (4):

$$J = \omega \cdot \int_0^T Nor(CO(t))dt + (1-\omega) \cdot \int_0^T Nor(EC(t))dt \tag{4}$$

where $0 < \omega < 1$ controls the importance of the CO, while $1 - \omega$ controls the importance of the EC. Since CO and EC are equally important, ω is set to 0.5 in this paper. $Nor(CO(t))$ and $Nor(EC(t))$ are normalized $CO(t)$ and $EC(t)$ by using the common min-max normalization technique [6].

4 The proposed approach: LLM-GPHH

4.1 Overview

The flowchart of LLM-GPHH is shown in Fig. 1. LLM-GPHH starts with creating an *initial population* that consists of randomly generated individuals. Each individual includes a *VM selection heuristic* and a *PM selection heuristic*. Both heuristics are represented as trees. Individuals in every generation are evaluated by the optimization objective.

There are two ways to generate new individuals for the next generation population. On the one hand, selection is used to select a subset of individuals, and traditional evolutionary operators, i.e., crossover, mutation and reproduction, are applied to this subset of individuals to generate new individuals. On the other hand, in our newly proposed *LLM evolution*, high-fitness individuals will be stored in an *archive* based on an *archive* update strategy. Subsequently, LLM is leveraged to capture the implicit knowledge among high-fitness individuals in the *archive*, to generate improved new *VM selection heuristics* and *PM selection heuristics*. The details of the LLM-GPHH are described as follows. In Fig. 1, the main contributions of this paper are highlighted in blue.

4.2 Representation

LLM-GPHH adopts a tree structure to represent the *VM selection heuristic* and the *PM selection heuristic*, as commonly used in GPHH [18]. Examples of the *VM selection heuristic* and *PM selection heuristic* are shown in Fig. 2. Each tree in Fig. 2 has one root, multiple non-leaf nodes and leaf nodes. The non-leaf nodes include elementary functions, e.g., $+, /, -, \times$. Each leaf node, named terminal, represents one of the problem-dependent features of the MDC problem.

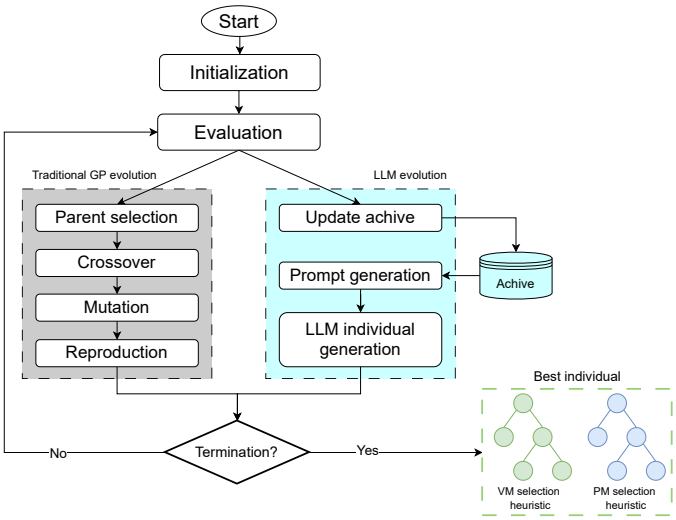


Fig. 1: The overall process of LLM-GPHH

To capture the features of dynamic MDC problems, we elaborately design a set of terminals, which are shown in Table 1. Specifically, the *VM selection heuristic* uses VM-specific terminals. The *PM selection heuristic* uses PM-specific terminals. Besides the terminals already used in existing research [16,17], we designed four new terminals (highlighted in red). The main idea of these newly designed terminals is to capture the features of 1) communication data volume within VMs, or PMs (i.e., VIC or PIC) and 2) external communication data volume of VMs or PMs (i.e., VOC or POC), which are important for optimizing the CO during the evolution.

4.3 Fitness Evaluation

To evaluate the performance of individuals, the *VM selection heuristic* and *PM selection heuristic* of each individual are used to select VMs and PMs for microservice applications. Firstly, the min-cut heuristic [4, 7] is utilized to divide the microservice application into different partitions, such that the communication data volumes across each partitions are minimal. Then, microservices in the same partitions are deployed together to the same VM.

Fig. 2 illustrates how to use examples of a *VM selection heuristic* (i.e, $\frac{CC}{LVC} - (LVM + COV)$) and a *PM selection heuristic* (i.e., $(VMC + VMM) + PMC$) for MDC problems. According to the table in Fig. 2, the priority values of VM_1, VM_2 and VM_3 in Fig. 2 are calculated by the *VM selection heuristic*, i.e, $\frac{21}{10} - (0.7 + 1.2) = 0.8$, $\frac{14}{11} - (0.9 + 0.2) = 0.17$ and $\frac{15}{1.8} - (3.1 + 4.4) = 0.83$. Therefore, VM_3 is selected for the microservices deployment as it has the highest priority

Table 1: Terminals used by LLM-GPHH. The newly designed terminals are highlighted in blue

VM-related	**Description**
CC	CPU requirement of a container
CM	Memory requirement of a container
LVC	Lefting CPU of a VM instance
LVM	Lefting memory of a VM instance
COV	The CPU overhead of a VM instance
MOV	The Memory overhead of a VM instance
VOC	communication data volume transfer externally by the PM that a VM is located
VIC	communication data volume within the PM that a VM is located
PM-related	**Description**
VMC	The CPU capacity of a VM instance
VMM	The memory capacity of a VM instance
LPC	Lefting CPU of a PM
LPM	Lefting memory of a PM
PMC	The CPU capacity of a PM
PMM	The memory capacity of a PM
PCore	The number of cores of a PM
POC	communication data volume transfers externally of a PM
POC	communication data volume within a PM

value. Similarly, for each newly created VM instance, PM_1 is selected based on the *PM selection heuristic* shown in Fig. 2. After the deployment of newly arrived microservices and newly created VMs into the selected VMs and PMs, the fitness value of an individual is calculated by Eq. (4).

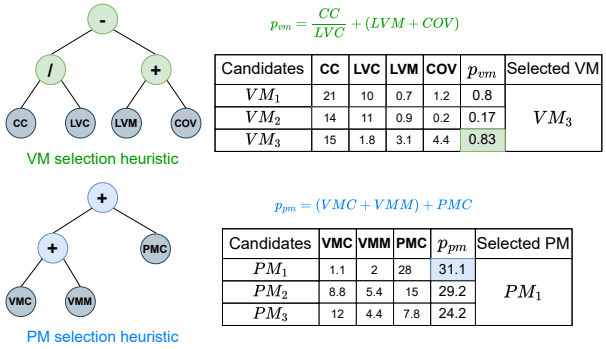
4.4 Traditional GP Evolution.

The traditional evolutionary operators of LLM-GPHH include crossover, mutation, and reproduction. Crossover exchanges a sub-tree of one heuristic (i.e., *VM selection heuristic* or *PM selection heuristic*) in the parent individual with a sub-tree of the same heuristic type in the second parent individual. The mutation is applied to an individual by replacing a sub-tree of a heuristic with a randomly generated sub-tree. The process of reproduction is straightforward and directly preserves selected individuals to the next generation.

4.5 LLM Evolution

The process of *LLM evolution* is shown in Algorithm 1. The inputs of *LLM evolution* include an *archive*, the best individual in current generation ind_{best}, the number of heuristics chosen for prompt *prompt_ size*, and the number of heuristics generated by LLM *num*. Then *LLM evolution* outputs new individuals.

Firstly, an *archive* update strategy is designed in this paper (line 1). Specifically, when there is no individual in the *archive* that is identical to the ind_{best} in the tree structure, heuristics in ind_{best} will be stored in the archive; otherwise, the ind_{best} will not be stored to maintain the diversity in the *archive* [18]. When



$p_{vm} = \frac{CC}{LVC} + (LVM + COV)$

Candidates	CC	LVC	LVM	COV	p_{vm}	Selected VM
VM_1	21	10	0.7	1.2	0.8	VM_3
VM_2	14	11	0.9	0.2	0.17	
VM_3	15	1.8	3.1	4.4	0.83	

$p_{pm} = (VMC + VMM) + PMC$

Candidates	VMC	VMM	PMC	p_{pm}	Selected PM
PM_1	1.1	2	28	31.1	PM_1
PM_2	8.8	5.4	15	29.2	
PM_3	12	4.4	7.8	24.2	

Fig. 2: Example of using *VM selection heuristic* and *PM selection heuristic* for the microservice deployment

the *archive* reaches a fixed capacity (e.g., 5), the earliest stored individual within the *archive* is removed due to the earliest individual lacks several generations of evolution, which is most likely to perform poorly. Then, a prompt is generated by sampling promising individuals in the *archive* (lines 3-4). An example of a generated prompt is shown in Fig. 3, which includes three parts: *task description*, *promising heuristics* and *output format*. To be specific, the *task description* part gives details of the problem to be solved (i.e., resource allocation problem in container-based cloud for microservice applications). Meanwhile, the format of promising heuristics and the expected outputs from LLMs are detailed and described in the *task description*. In the *promising heuristics* part, *prompt_size* promising individuals from the *achive* are sampled with an equal probability for each individual (line 3).

The response of LLM includes *num* new heuristics (Line 5). A response with three new heuristics is presented in Fig. 3 (b). Then, new heuristics will be checked for their validity (Line 6). For example, $LVM - COV+$ is not a correct heuristic since the absence of the second operand of $+$. Correct heuristics will be converted into tree-represented individuals (Line 7). Individuals generated by *LLM evolution* are combined with the individuals generated by traditional GP evolution as a new population for the next generation.

5 Experimental Evaluation

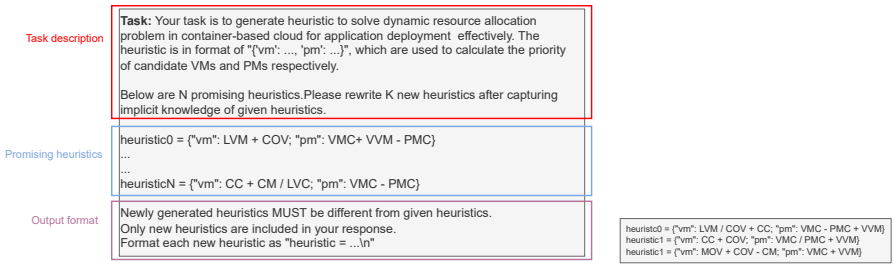
We compare LLM-GPHH with two heuristic algorithms, i.e., KP_HP [7], and ECSched [8], as well as one state-of-the-art hyper-heuristic algorithm, Hybrid-Evo [17]. The average performance results obtained under 30 independent runs are verified and compared based on the Wilcoxon rank-sum test with a significance level of 0.05.

Algorithm 1 The process of *LLM evolution*

Input: An *archive*; the best individual in current generation: ind_{best}; the number of heuristics chosen for prompt: $prompt\_size$; the number of heuristics generated by LLM: num;

Output: new individuals generated by LLM;

```
archive ← update_archive(ind_best)
new_indivdiuals ← []
heuristics ← sample(archive, prompt_size)
prompt ← generate_prompt(prompt_size, num, heuristics)
new_heuristics ← LLM(prompt)
valid_heuristics ← Check(new_heuristics)
new_individuals.add(Generate(valid_heuristics))
```



(a) Example of prompt (b) Example of response

Fig. 3: Examples of prompt and the response of LLM

Simulation. We simulate the state of a cloud data center by using a real-world dataset [12,17] that records the arrivals of a large number of applications in cloud data centers during a long time period. Each application belongs to one of the popular application structures studied in [13]. Six scenarios are used to simulate different workload patterns of new arrival applications and the number of different OS types, following [17].

Parameter Settings. The parameter settings of GPHH are based on the existing state-of-the-art GPHH methods [16,17]. Specifically, the population size is 512. The tournament selection size is 7. The rates of crossover, mutation and reproduction are 0.8, 0.1 and 0.1, respectively. The archive size of LLM-GPHH is 10, while the prompt_size is 5. The number of LLM generated individuals is 1. GPT-3.5-turbo-0613 [4] API is used in this paper under the consideration of cost and performance.

Experiment Results. Table 2 compares the test performance results across all scenarios. We can observe from Table 2 that LLM-GPHH significantly outperforms the other three algorithms (i.e., KP-HP, ECSched and Hybrid-Evo) in

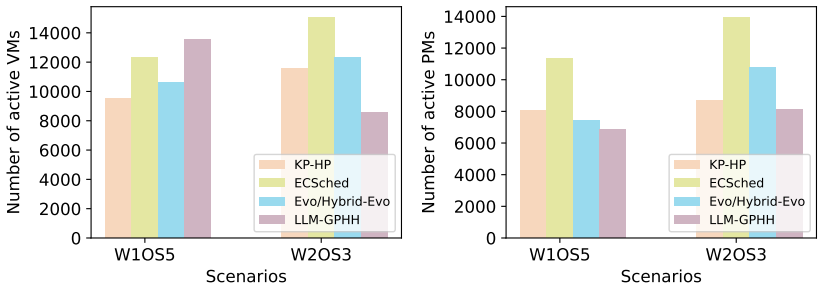
[4] https://openai.com/pricing

terms of CO in all six scenarios. As for EC, LLM-GPHH consumes less EC in most scenarios when compared to KP-HP, ECSched and Hybrid-Evo.

Table 2: The comparison results of test instances in 6 scenarios in terms of mean and standard deviation of EC (kWh) and CO (Mbps). (*Notice that smaller values in the table mean better performance; "+", "-" and "=" indicate significantly better, significantly worse and no significant difference respectively*)

Scenarios	Objectives	KP-HP [7]	ECSched [8]	Hybrid-Evo [17]	LLM-GPHH
W1OS3	**EC**	86123.45 ± 352.58	119882.72 ± 423.55	87849.80 ± 8610.68	83392.60 ± 3443.20 (+)(+)(+)
	CO	152102.27 ± 2302.83	181091.05 ± 3993.89	895782.48 ± 132158.76	84528.65 ± 9254.59 (+)(+)(+)
W1OS4	**EC**	79900.73 ± 536.62	119128.81 ± 384.50	85797.29 ± 9164.63	80519.16 ± 9230.01 (=)(+)(+)
	CO	140231.02 ± 2487.36	161973.62 ± 3545.41	688004.95 ± 235592.43	73013.34 ± 8491.053 (+)(+)(+)
W1OS5	**EC**	74794.61 ± 1140.74	113037.15 ± 343.56	78659.62 ± 7192.01	74105.31 ± 3926.38 (=)(+)(=)
	CO	127200.59 ± 2544.78	155613.84 ± 3663.14	664116.27 ± 225881.42	74340.36 ± 11513.97 (+)(+)(+)
W2OS3	**EC**	77425.84 ± 768.97	132472.29 ± 528.86	90405.21 ± 12136.97	79318.25 ± 13292.66 (+)(+)(+)
	CO	130645.07 ± 1926.53	171362.58 ± 3361.51	668220.39 ± 204766.39	84928.44 ± 6608.78 (+)(+)(+)
W2OS4	**EC**	75623.89 ± 863.04	125253.16 ± 550.92	85107.43 ± 48059.55	77736.15 ± 22240.93 (+)(=)(+)
	CO	121785.46 ± 2618.48	159210.25 ± 4696.93	929877.51 ± 140681.28	78142.90 ± 9769.10 (+)(+)(+)
W2OS5	**EC**	66919.23 ± 1006.71	117324.06 ± 452.58	69191.39 ± 8164.00	73121.75 ± 10912.93 **(-)**(+)(=)
	CO	108456.06 ± 2543.06	146288.07 ± 3167.12	760304.42 ± 229171.43	76007.46 ± 2219.03 (+)(+)(+)

Number of active VMs and PMs. We see from Fig. 4 that LLM-GPHH requires more active VMs compared to other algorithms, but it minimizes the number of active PMs in W1OS5. Meanwhile, both the active VMs and PMs of LLM-GPHH are minimized in W2OS3. As a result, LLM-GPHH can save computational resources by reducing the overhead of VMs and PMs [19].



(a) Number of VMs (b) Number of PMs

Fig. 4: The number of active VMs and PMs

Effectiveness of LLM evolution. We compare LLM-GPHH with the GPHH algorithm designed in this paper but without the assistance of LLM (GPHH-noLLM) in two scenarios. The results in Table 3 show the effectiveness of *LLM evolution* as LLM-GPHH outperforms GPHH-noLLM in terms of EC in both W1OS3 and W2OS3. Meanwhile, LLM-GPHH consumes less CO than GPHH-noLLM in W2OS3, while has no significant difference with GPHH-noLLM in W1OS3. The results show the effectiveness of *LLM evolution* in some scenarios. We also test different settings of *LLM evolution*, e.g., the number of promising heuristics in the prompt and the number of response heuristics, the most effective settings are described in the parameters setting part.

Table 3: The comparison results of ablation studies. (*Notice that smaller values in the table mean better performance; "+", "-" and "=" indicate significantly better, significantly worse and no significant difference respectively*)

Scenarios	Objectives	GPHH-noLLM	LLM-GPHH
W1OS3	**EC**	88235.35 ± 9292.04	83392.60 ± 3443.20 (+)
	CO	80215.42 ± 10028.89	84528.65 ± 9254.59 (=)
W2OS3	**EC**	85933.33 ± 10005.24	79318.25 ± 13292.66 (+)
	CO	88168.45 ± 3303.04	84928.44 ± 6608.78 (+)

6 Conclusions and Future Work

In this paper, to solve the dynamic MDC problems, we proposed *LLM evolution* to enhance the ability of GPHH to generate effective heuristics. Experimental results showed that LLM-GPHH outperforms existing heuristics and hyper-heuristics methods in terms of minimizing energy consumption and communication overhead. Meanwhile, ablation studies also showed the effectiveness of *LLM evolution*. This suggests that LLM indeed has the potential to enhance GPHH's capabilities in searching effective heuristics. In the future, we will continue to explore the ability of LLM and try to further improve the capability of LLM in addressing complex real-world problems.

References

1. Alzahrani, A., Tang, M.: A microservice-based saas deployment in a data center considering computational server and network energy consumption. In: 2023 IEEE 16th International Conference on Cloud Computing (CLOUD). pp. 505–515. IEEE (2023)
2. Buyya, R., Ilager, S., Arroba, P.: Energy-efficiency and sustainability in new generation cloud computing: A vision and directions for integrated management of data centre resources and workloads. Software: Practice and Experience **54**(1), 24–38 (2024)
3. Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Communications surveys & tutorials **18**(1), 732–794 (2015)
4. Fang, Z., Ma, H., Chen, G., Hartmann, S.: Energy-efficient and communication-aware resource allocation in container-based cloud with group genetic algorithm. In: International Conference on Service-Oriented Computing. pp. 212–226. Springer (2023)
5. Fang, Z., Ma, H., Chen, G., Hartmann, S.: A group genetic algorithm for energy-efficient resource allocation in container-based clouds with heterogeneous physical machines. In: Australasian Joint Conference on Artificial Intelligence. pp. 453–465. Springer (2023)
6. Gajera, V., Gupta, R., Jana, P.K., et al.: An effective multi-objective task scheduling algorithm using min-max normalization in cloud computing. In: 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT). pp. 812–816. IEEE (2016)

7. Hu, Y., de Laat, C., Zhao, Z.: Optimizing service placement for microservice architecture in clouds. Applied Sciences **9**(21), 4663 (2019)
8. Hu, Y., Zhou, H., de Laat, C., Zhao, Z.: Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. Future Generation Computer Systems **102**, 562–573 (2020)
9. Meyerson, E., Nelson, M.J., Bradley, H., Gaier, A., Moradi, A., Hoover, A.K., Lehman, J.: Language model crossover: Variation through few-shot prompting. arXiv preprint arXiv:2302.12170 (2023)
10. Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J., Ellenberg, J.S., Wang, P., Fawzi, O., et al.: Mathematical discoveries from program search with large language models. Nature **625**(7995), 468–475 (2024)
11. Sharma, H.C., Bisht, M.: Best fit resource allocation in cloud computing. International Journal of Computer Sciences and Engineering, E-ISSN pp. 2347–2693 (2019)
12. Shen, S., Van Beek, V., Iosup, A.: Statistical characterization of business-critical workloads hosted in cloud datacenters. In: 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 465–474. IEEE (2015)
13. Shi, T., Ma, H., Chen, G., Hartmann, S.: Location-aware and budget-constrained application replication and deployment in multi-cloud environment. In: 2020 IEEE International Conference on Web Services (ICWS). pp. 110–117. IEEE (2020)
14. Sorgalla, J., Sachweh, S., Zündorf, A.: Exploring the microservice development process in small and medium-sized organizations. In: Product-Focused Software Process Improvement: 21st International Conference, PROFES 2020, Turin, Italy, November 25–27, 2020, Proceedings 21. pp. 453–460. Springer (2020)
15. Tan, B., Ma, H., Mei, Y.: A NSGA-II-based approach for multi-objective microservice allocation in container-based clouds. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid). pp. 282–289. IEEE (2020)
16. Tan, B., Ma, H., Mei, Y., Zhang, M.: A cooperative coevolution genetic programming hyper-heuristics approach for on-line resource allocation in container-based clouds. IEEE Transactions on Cloud Computing **10**(3), 1500–1514 (2020)
17. Wang, C., Ma, H., Chen, G., Huang, V., Yu, Y., Christopher, K.: 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). pp. 539–555. Springer (2023)
18. Wang, S., Mei, Y., Zhang, M.: Explaining genetic programming-evolved routing policies for uncertain capacitated arc routing problems. IEEE Transactions on Evolutionary Computation (2023)
19. Xu, F., Liu, F., Jin, H., Vasilakos, A.V.: Managing performance overhead of virtual machines in cloud computing: A survey, state of the art, and future directions. Proceedings of the IEEE **102**(1), 11–31 (2013)
20. Yang, Y., Chen, G., Ma, H., Zhang, M.: Dual-tree genetic programming for deadline-constrained dynamic workflow scheduling in cloud. In: International Conference on Service-Oriented Computing. pp. 433–448. Springer (2022)
21. Yuan, J., Bae, E., Tai, X.C.: A study on continuous max-flow and min-cut approaches. In: 2010 ieee computer society conference on computer vision and pattern recognition. pp. 2217–2224. IEEE (2010)