A Genetic Programming Hyper-Heuristic Approach to Design High-Level Heuristics for Dynamic Workflow Scheduling in Cloud

Kirita-Rose Escott, Hui Ma and Gang Chen
School of Engineering and Computer Science, Victoria University of Wellington
Wellington, New Zealand
{kirita-rose.escott, hui.ma, aaron.chen}@ecs.vuw.ac.nz

***Abstract*—Workflow scheduling in the cloud is the process of allocating tasks to scarce cloud resources, with an optimal goal. This is often achieved by adopting an effective scheduling heuristic. Workflow scheduling in cloud is challenging due to the dynamic nature of the cloud, often existing works focus on static workflows, ignoring this challenge. Existing heuristic such as MINMIN focus mainly on one specific aspect of the scheduling problem. High-level heuristics are heuristics constructed from existing man-made heuristics. In this paper, we introduce a new and more realistic workflow scheduling problem that considers different kinds of workflows, cloud resources and high-level heuristics. We propose a High-Level Heuristic Dynamic Workflow Scheduling Genetic Programming (HLH-DSGP) algorithm to automatically design high-level heuristics for workflow scheduling to minimise the response time of dynamically arriving task in a workflow. Our proposed HLH-DSGP can work consistently well regardless of the size and pattern of workflows, or number of available cloud resources. It is evaluated using a popular benchmark dataset using the well-know WorkflowSim simulator. Our experiments show that with high-level scheduling heuristics, designed by HLH-DSGP, we can jointly use several well-known heuristics to achieve a desirable balance among multiple aspects of the scheduling problem collectively, hence improving the scheduling performance.**

***Index Terms*—Cloud Computing, Dynamic Workflow Scheduling, Genetic Programming**

I. Introduction

Cloud computing is a large-scale heterogeneous and distributed computing infrastructure, which provides high quality and low cost services with minimal hardware investments [1]. A workflow is a loosely coupled coarse-grained parallel application that consists of a set of computational tasks linked through control and data dependencies [2]. Scheduling workflow tasks with an optimal goal, such as minimizing operating costs while maintaining or even improving quality of service is an important task for cloud providers.

Workflow scheduling aims to allocate workflow tasks to cloud resources so that the performance, e.g., the average response time involved in executing dynamically arriving tasks in workflows, can be minimized. Such a workflow scheduling problem is widely known as NP-complete [3]. As no algorithms have ever been developed to solve this problem in an efficient and scalable manner, heuristics are needed to find near-optimal solutions efficiently.

There are two workflow scheduling models, static and dynamic. The static model is represented when characteristics of a workflow, including execution times of tasks, the data size of communication between tasks, and the task dependencies are known a priori [4]. In the dynamic model, cloud resources are allocated to ready tasks at runtime [5]. The arrival of new requests for workflow execution make runtime resource allocation in cloud inevitable.

There are many factors that affect the workflow scheduling problem, such as the pattern of a workflow, the number of available cloud resources, the size of workflow tasks, and task dependencies [6]. Several man-made heuristics have been designed to address the dynamic workflow scheduling problem, such as *Heterogeneous Earliest Finish Time (HEFT)* [4], *MINMIN* [7] and *MAXMIN* [8]. These approaches only rely on a small number of factors, such as task size, to make scheduling decisions. Neglecting other impacting factors such as virtual machine speed and expected completion time of a task. Thus, these approaches cannot achieve promising results for a variety of different workflows and cloud settings. Therefore, it is highly desirable to design new heuristics with consistently good performance across many workflow scheduling scenarios.

As there are several impacting factors, such as task size, virtual machine speed and task execution time, there are several properties which must be jointly considered when designing scheduling heuristics. Numerous combinations of these properties are possible in practice to construct the scheduling heuristic. With high-level heuristics, we can jointly use several well-known heuristics, such as *MINMIN* and *MAXMIN*, to achieve a desirable balance among multiple properties of the workflow scheduling problem collectively. However, it is challenging to manually design an effective heuristic that is capable of solving many complex scheduling problems consistently well in the cloud.

GP based hyper-heuristics (GPHHs) have become increasingly popular because of their ability to evolve a wide range of program structures corresponding to different types of scheduling rules and heuristics [9]. Designing scheduling heuristics manually in the cloud is time-consuming and requires substantial domain knowledge. GP has successfully been employed to automatically design scheduling heuristics in the cloud

[6]. However, these approaches only considered the selection of virtual machines during allocation and did not consider exploiting man-made heuristics.

In this paper, we have the goal to develop a new GP-based algorithm for automatically designing high-level workflow scheduling heuristics in the cloud. Driven by this goal, this paper has three main research objectives:

1) Develop a new *High-Level Heuristic Dynamic Workflow Scheduling Genetic Programming* (HLH-DSGP) algorithm to automatically design high-level scheduling heuristics for the dynamic workflow scheduling problem.
2) Conduct an extensive experimental evaluation of HLH-DSGP by using the Workflowsim simulator and a popular benchmark dataset with 12 different workflow applications. Scheduling heuristics designed by HLH-DSGP are further compared to several existing scheduling heuristics, including GRP-HEFT [10], RR, and MCT [11].
3) Analyse the experimental results to investigate why designed heuristics did or didn't work.

This paper adopts as simulated evaluation approach using the WorkflowSim [11], a workflow extension of CloudSim [12].

The rest of the paper is organised as follows: Section II presents previous work related to workflow scheduling in the cloud environment. Section III gives an overview of the workflow scheduling problem. Section IV outlines the proposed HLH-DSGP approach. Section V presents the experimental evaluation results and analysis. The conclusions and future work are outlined in Section VI.

II. Related Work

Workflow scheduling is an important and challenging area of research in cloud computing. Numerous state-of-the-art workflow scheduling schemes have been proposed in the literature for scheduling scientific workflows in cloud computing [13]. Both static and dynamic approaches have been studied in the literature.

Shishido et al. [14] examine the effects of different meta-heuristics on workflow scheduling in cloud. The major objective was to investigate the optimization performance of Particle Swarm Optimization (PSO), the Genetic Algorithm (GA), and the Multi-Population Genetic Algorithm (MPGA) on static scheduling problems.

There are many existing workflow scheduling algorithms that consider different constraints such as budget, deadline, and security level. Faragardi et al. [10] proposed a novel resource provisioning mechanism and a workflow scheduling algorithm, named Greedy Resource Provisioning and modified HEFT (GRP-HEFT), for minimizing makespan of a given workflow subject to a budget constraint for the hourly-based cost model of modern IaaS clouds. Arabnejad et al. [15] propose a heuristic scheduling algorithm, Budget Deadline Aware Scheduling (BDAS), that addresses eScience workflow scheduling under budget and deadline constraints in Infrastructure as a Service. Ghafouri et al. [16] propose Constrained Budget-Decreased Time (CBDT), a scheduling algorithm which aims to create a schedule that reduces the makespan while satisfying the budget constraint of the workflow application. These approaches are also limited to static scheduling problems.

Saraswathi et al. [17] propose a dynamic resource allocation scheme where VMs are allocated based on the characteristics of the job. Jobs are either low or high priority and each have a lease type associated with them. Sahni et al. [2] propose a dynamic cost-effective deadline-constrained heuristic algorithm for scheduling a scientific workflow in a public cloud. Their Just-in-time (JIT-C) algorithm is able to address VM performance variation, resource acquisition delays and heterogeneous nature of cloud resources. Both approaches ignore other impacting factors such as virtual machine speed, size of tasks, and expected completion time of tasks.

Yu et al. [6] propose a Flexible Scheduling Genetic Programming (FSGP) approach that employs Genetic Programming to automatically design scheduling heuristics for the static workflow scheduling problem. Escott et al. [18] propose a Dynamic Workflow Scheduling Genetic Programming (DSGP) algorithm to automatically design scheduling heuristics for the dynamic workflow scheduling problem. This paper is an extension to our proposed *Dynamic Workflow Scheduling Genetic Programming* (DSGP) [18]. Both approaches design a heuristic rule to select the best virtual machine for a task, however there are multiple tasks to be scheduled at a given time, the order of tasks to be scheduled have a big impact on the overall performance. Therefore, there is a need of heuristics that can be used to not only select virtual machines but also tasks to be allocated, i.e., the best task-to-virtual machine pair during scheduling. Furthermore, both FSGP and DSGP are compared with state-of-the-art heuristics such as Heterogeneous Earliest Finish Time (HEFT) [4], MINMIN [7] and MAXMIN [8]. These heuristics have demonstrated effectiveness in some related studies [19], hence it can be assumed that they can be effectively applied to the dynamic workflow scheduling problem. Given this and the fact that GP has been successfully used in similar problems [20], [21], in this paper we aim to design a GPHH approach that can evolve heuristics capable of producing a near-optimal resource allocation.

Many discussed approaches, such as [14], [15] and [16] are static and cannot be applied to the dynamic workflow scheduling problem. We experimentally compare with GRP-HEFT [10], MINMIN [7], MAXMIN [8], as well as First Come First Serve (FCFS), Minimum Completion Time (MCT) and Round Robin (RR) [11].

III. Preliminaries

In this section, first the overview of the problem is defined. Then, the formal definition of the dynamic workflow scheduling problem is formulated.

A. Problem Overview

In the workflow scheduling problem, a workflow w is commonly depicted as a directed acyclic graphs $DAG([n],[e])$, as

demonstrated in Fig 1, where n is a set of nodes representing n dependent tasks, $[e]$ indicates the data flow dependencies between tasks. Each workflow has a task with no predecessors named $entry$ task and a task with no successors names $exit$ task. Considering workflows with different patterns, and tasks arriving from time to time, *dynamic workflow scheduling* allocates tasks to virtual machines with the objective of minimizing the overall response time of execution.

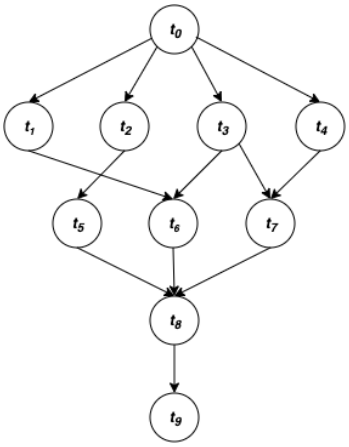

Fig. 1. A DAG with 10 Tasks in the Workflow

In the process of allocation, the following constraints must be satisfied.

- A task can only be allocated once all its predecessors have been executed.
- Each task can be allocated to any virtual machine, which will process the task.
- Each virtual machine can only process at most one task at any given time.

B. Formal Definition

The following equations present the basic properties of tasks and virtual machines that will be used to formulate the dynamic workflow scheduling problem.

The execution time of task t_i on virtual machine v_k is given by Equation (1).

$$ET_{ik} = \frac{s_i}{m_k} \tag{1}$$

ET_{ik} is obtained by dividing the size s_i of task t_i divided by the speed m_k of virtual machine v_k.

A task t_i becomes ready for execution whenever all of its parents $aParent(t_i)$ have completed processing. A pair p_{ik} is a mapping of ready task t_i to virtual machine v_k in set of pairs i.e. $p_{ik} \in P$, denoted by Equation (2).

$$p_{ik} = (t_i, v_k) \tag{2}$$

The rank $r(p_{ik})$ of a pair p_{ik} is determined by the heuristic rule r. The pairs in P are ordered according to rank r_p.

TABLE I
NOTATIONS FOR DYNAMIC WORKFLOW SCHEDULING PROBLEM

Notation	Description
M	Total number of virtual machines
VM	Set of of virtual machines
v_k	A virtual machine $v_k \in VM$
m_k	Speed of virtual machine v_k
b_k	Bandwidth of virtual machine v_k
N	Total number of tasks in a workflow
T	Set of tasks in a workflow
t_i	A task $t_i \in T$
s_i	Size of task t_i
$aParent(t_i)$	All immediate predecessors of task t_i
$aChild(t_i)$	All immediate successors of task t_i
ET_{ik}	The execution time of task t_i on virtual machine v_k
AT_i	Arrival time of task t_i
ST_i	Start time of task t_i
WT_i	Wait time of task t_i
ECT_i	Expected Completion time of task t_i and it's children $aChild(t_i)$
RT_i	Actual Response time of task t_i
P	Set of pairs
p_{ik}	A mapping of task t_i to virtual machine v_k pair $p_{ik} \in P$
r_p	The rank of a pair $p_{ik} \in P$
Num	Total number of workflows
W	Set of workflows
w_j	A workflow $w_j \in W$
ART_j	Average response time of a workflow
ART	Average response time of all workflows

Each ready task t_i can be allocated to a virtual machine v_k. The allocation time, denoted by AT_i, is given by Equation (3).

$$AT_i = max_{p \in aParent(t_i)} FT_p \tag{3}$$

The actual start time of each task t_i, denoted by ST_i, is decided either by the time that virtual machine v_k becomes idle, or when t_i becomes a ready task t_i, whichever is later. Virtual machine v_k becomes idle when the previous task t_{prev} being executed on v_k completes processing. The actual start time is given by Equation (4).

$$ST_i = \max\{FT_{t_{prev}}, AT_i\} \tag{4}$$

The waiting time of task t_i, denoted WT_i, is given by Equation (5), as the difference between the arrival time and the start time of a task t_i.

$$WT_i = ST_i - AT_i \tag{5}$$

The expected completion time ECT_i of task t_i and it's children $aChild(t_i)$ is the maximum expected completion time possible and is given by Equation (6).

$$ECT_i = total_{c \in aChild(t_i)} ET_{ik} \tag{6}$$

The actual response time of task t_i, denoted RT_i, is given by Equation (7).

$$RT_i = FT_i - AT_i \tag{7}$$

The response time of a workflow, denoted ART_j, is the total response time of all tasks t_i in the workflow T divided by the

total number of tasks N in the workflow. ART_j is given by Equation (8).

$$ART_j = \frac{\sum_{i=1}^{N} RT_i}{N} \tag{8}$$

The response time for all workflows, denoted ART, is the sum of the response time ART_T for all workflows W divided by the total number of workflows Num.

$$ART = \frac{\sum_{j=1}^{N} ART_j}{Num} \tag{9}$$

The objective of the dynamic workflow scheduling problem is to minimize the overall response time of all the workflows. i.e., min ART. We aim to find a rule r, which can produce a schedule with low ART.

IV. Designing High-Level Heuristics for Dynamic Workflow Scheduling using Genetic Programming Approach

In this section we present our new HLH-DSGP algorithm to solve the workflow scheduling problem. We describe the outline of the algorithm. We then describe the Genetic Programming Hyper Heuristics (GPHH) aspect of our approach which outlines the representation of a scheduling heuristic, the terminal and function sets, as well as the genetic operators crossover, mutation, and reproduction.

A. Outline of HLH-DSGP

HLH-DSGP aims to exploit man-made heuristics to design high-level heuristics to schedule cloud resources for workflow execution. We rely on the Workflow Scheduler in the WorkflowSim simulator to simulate the process of dynamically arriving tasks of a workflow over an extensive period of time [11]. In WorkflowSim, the Workflow Scheduler obtains *ready tasks* from the Workflow Engine periodically. A task is considered a ready task if and only if all of its parent tasks have finished executing on cloud resources.

HLH-DSGP aims to produce a schedule of task to VM pairs that minimizes the average response time involved in executing a set of workflows using a fixed set of cloud resources. Unlike previous approach [18], HLH-DSGP considers both task and VM jointly to form each task-VM pair for the schedule. This joint consideration has potential to provide more flexible scheduling decisions and room for improved performance.

HLH-DSGP is a hyper-heuristic approach that consists of a *training phase* and a *testing phase*. During the training phase, HLH-DSGP evolves a high-level scheduling heuristic, driven by the goal to minimize average response time involved in executing a set of workflows using a fixed set of cloud resources. The best heuristic discovered by HLH-DSGP is subsequently utilized in the testing phase to schedule various workflows. Figure 2 demonstrates an overview of the training process.

As discussed, HLH-DSGP is designed to search for effective high-level heuristics to solve the dynamic workflow scheduling problem in cloud. The pseudo-code in ALGORITHM 1 outlines the HLH-DSGP algorithm.

Algorithm 1: GPHH Algorithm

```
Input: A set of workflows
Output: Heuristic Rule
Initialize a population of rules R;
while does not meet stop criteria do
    for a rule r in R do
        for a workflow w in W do
            Initialise the simulator;
            for a ready task t in T do
                for a virtual machine vm in VM do
                    if vm is idle then
                        p = createVmTaskPair(vm, t)
                    end
                end
            end
            rankVmTaskPairs(P, r);
            for a pair p in P do
                if vm is idle then
                    allocate(vm, rt);
                end
            end
            ART += calculateAverageResponseTime(w);
        end
        Evaluate fitness of r;
    end
    TournamentSelection;
    Crossover;
    Mutation;
    Reproduction;
end
return the best rule
```

In our experiments, the training set consists of 12 different workflow applications and a fixed collection of cloud resources. Our approach follows a standard GP framework with an embedded simulation as the evaluation process. As demonstrated, the output of the training phase is the best scheduling high-level heuristic rule discovered by HLH-DSGP. This heuristics is further evaluated in the testing phase.

B. Genetic Programming Hyper Heuristic (GPHH)

Genetic Programming (GP) is a prominent Evolutionary Computation (EC) technology for designing heuristics for various difficult problems [22]. GP contains following phases: initialization, selection, crossover, mutation. The aim of GP is to find the tree with the best fitness value.

We represent a scheduling heuristic rule in the form of a binary tree. In GP, a feasible solution is represented by a binary tree which consists of function set and terminal set [23]. The function set F contains the operations of values, including arithmetic operator, mathematical function, boolean operator and conditional expression, etc. The terminal set T is composed of variables. The novelty of this work is that it

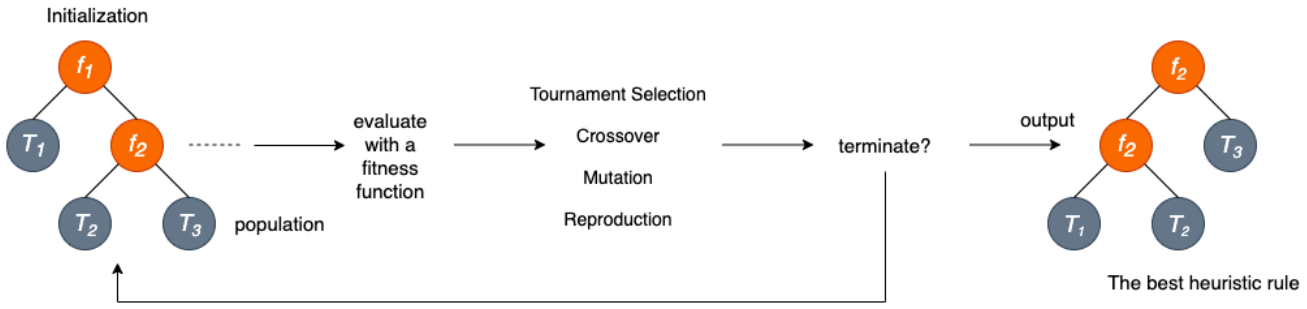

Fig. 2. Overview of the GPHH training and testing

exploits man-made heuristics by utilizing them as terminals in conjunction with aspects of the workflow scheduling problem such as task size, virtual machine size and expected completion time of a task. The terminal and function sets are described in Table II.

TABLE II
THE TERMINAL AND FUNCTION SET FOR THE PROPOSED GPHH APPROACH

Terminal Name	**Definition**
TS	The total size of a task t_i
VS	The speed of a virtual machine v_k
UC	The unit cost of a virtual machine v_k
ET	Execution time of a task t_i
WT	Waiting time of a task t_i
BW	Bandwidth of a virtual machine v_k
ECT	Expected completion time of a task t_i and it's children
$MINMIN$	Rank of pair p from MINMIN heuristic
$MAXMIN$	Rank of pair p from MAXMIN heuristic
RR	Rank of pair p from RR heuristic
$FCFS$	Rank of pair p from FCFS heuristic
MCT	Rank of pair p from MCT heuristic
Function Name	**Definition**
Add, Subtract, Multiply, Minimum, Maximum	Basic arithmetic operations (+, -, $\times$, min, max)
Protected Division	Protected division, return 1 if the denominator is 0 (%)

Crossover, mutation, and reproduction are commonly used techniques in GP. We aim to use these techniques to generate scheduling heuristics which will then be evaluated using a determined fitness function, discussed in this section.

During *Initialization*, the initial population is created by generating binary trees randomly with maximum depth. When the binary tree started, an element will be selected randomly from the union set composed by function set F and terminal set T to generate a child node [23].

Crossover refers to the genetic operator which recombines the genetic information of two parents to create a new offspring. For example, when the left branch of Parent A is combined with the right branch of Parent B to become offspring A'. Additionally, the left branch of Parent B is combined with the right branch of Parent A to become offspring B'.

Mutation refers to the genetic operator which changes the sub-tree of a tree to maintain genetic diversity from one generation to the next. For example, when the sub-tree of the parent is replaced by a new sub-tree in the child. Mutation is used to prevent instances from becoming too similar to each other, which, can lead to poor performance [6].

Reproduction refers to the genetic operator which selects individuals from the current generation and passes the best of those individuals to the next generation as offspring.

The *Fitness* of a heuristic rule r is evaluated by the overall average response time ART, of all workflows scheduled using the rule. Fitness is defined in Equation (10).

$$fitness(r) = AMS(r) \tag{10}$$

Based on the above, the goal is to find r^* such that it minimizes $ART(r^*)$ over all possible rules.

V. EXPERIMENTAL EVALUATION

In this sections, we describe the experiments we conducted to evaluate the HLH-DSGP algorithm. We perform this evaluation using the WorkflowSim simulator to simulate a real cloud environment and execute workflows using our generated heuristic rule.

A. Dataset

Four well-known workflow patterns, CyberShake, Inspiral, Montage, and Sipht, are used in our experiments. We use three different sizes, small, medium and large, as described in Table III. Thus, we tested 12 workflow patterns, with the number of tasks ranging between 25 and 100.

TABLE III
NUMBER OF TASKS IN WORKFLOW APPLICATIONS

Application Size	**CyberShake**	**Inspiral**	**Montage**	**SIPHT**
Small	30	30	25	30
Medium	50	50	50	60
Large	100	100	100	100

Our sets of virtual machines consisted of VMs of small, medium, large and extra large size. We have a small set of 4, which has 1 each of the different sized VMs, a medium set of 16, which has 4 of each VM and a large set of 32, which has

8 of each VM. We have 12 workflows, and 3 sets of VMs to test each workflow, therefore we have 36 test scenarios.

B. Baseline Algorithms

WorkflowSim has dynamic implementations of traditional workflow scheduling algorithms. We examined those implementations and give a summary of each below [10] [8] [7] [11].

- GRP-HEFT is a modified version of HEFT that considers cost while always selects the fastest available virtual machine to schedule a task to
- MINMIN takes the task with the minimum size and allocates it to the first idle virtual machine
- MAXMIN takes the task with the largest size and allocates it to the first idle virtual machine
- FCFS allocates the first ready task to the first available virtual machine
- MCT allocates a task to the virtual machine with the minimum expected completion time for that task
- RR selects the virtual machine a task is to be allocated to in a circular order

C. Parameter Settings

In our experiments we set the population size to 512, and the number of generations to 51. The crossover, mutation and reproduction rates are 85%, 10% and 5% respectively [20]. The tournament size for tournament selection is 5 and the maximum depth of a tree is set to 10. Tournament selection is used to encourage the survival of effective heuristics in the population. We run the our experiments 30 times to verify our results.

D. Results

The overall results of our experiments show that HLH-DSGP is able to design a high-level scheduling heuristic that can produce a schedule with the lowest overall response time. Table IV shows the overall response time of each of the algorithms for all workflow scenarios. The smaller the value, the better the performance of the algorithm. From Table IV we see that FCFS and Round Robin are the poorest performing algorithms.

Tables V, VI, and VII show the response time for individual workflow scenarios. As discussed, small workflows are those with 25-30 tasks, mediums workflows have 50-60 tasks and large workflows have 100 tasks. All workflows are run on all three sets of cloud resources. In addition to the results for individual workflows, we calculate the *average* of each algorithm for each size of cloud resources.

The results in Table V show that, GRP-HEFT is a strong competitor for small workflows, particularly for the medium and large sets of virtual machines. However, HLH-DSGP outpeforms GRP-HEFT for the small set of virtual machine. Additionally, as seen in Table IV, GRP-HEFT is unable to produce a schedule with a low overall response time.

The results in Table VI show that GRP-HEFT is a strong competitor for medium workflows, particularly for the medium and large sets of virtual machines. However, again HLH-DSGP outpeforms GRP-HEFT for the small set of virtual machine and overall.

The results in Table VII show that HLH-DSGP clearly outperforms the competing algorithms for large workflows. On average HLH-DSGP outperforms competing algorithms for the small and medium sets of virtual machines. Whereas, MAXMIN performs best for the large set of virtual machines.

Of all the approaches, overall MINMIN is secondary to our HLH-DSGP approach. On average, in the different workflows GRP-HEFT performs secondary to HLH-DSGP. We see that RR and FCFS consistently perform the worst, both overall and on average in the different workflow scenarios.

E. Further Analysis

We analysed the terminals of the best heuristic rules generated in each independent run to determine which terminals affected the performance of high-level heuristics generated by HLH-DSGP. We counted the frequency of terminal appearances, show in Table VIII. We can observe from Table VIII that the most frequently occurring terminals are the rank of task-VM pair according to the MINMIN heuristic ($MINMIN$) and the execution time (ET). In the following, we show one of the high-level heuristic rules generated by HLH-DSGP to analyse the relationship between the terminals and the heuristic rules. In this paper we schedule tasks to virtual machines according to the value calculated by the heuristic. The smaller the value, the faster it will be executed. From $Heuristic_1$, we see that ET and $MINMIN$ occur the most frequently. As MINMIN performed secondary to our HLH-DSGP algorithm overall, it is not surprising that the $MINMIN$ terminal occurred most frequently.

$Heuristic_1$ *(+ (/ (MIN (* (/ TS ET) (+ TS (+ MINMIN RR))) (MIN (* (+ MCT MINMIN) ET) (* (- FCFS MINMIN) MAXMIN))) MINMIN) (+ (* ECT ET) (+ (+ (- (MAX MAXMIN ET) (MAX UC BW)) ET) (+ (/ TS ET) (* ECT ET)))))*

As TS, $MAXMIN$, VS and $FCFS$ are the next most frequently occurring terminals, we know they have also played a role in my of the high-level heuristic rules generated by HLH-DSGP. Unlike other heuristics, high level heuristics generated by HLH-DSGP, jointly use well-known man-made heuristics such as MINMIN and MAXMIN, with other aspects of workflow scheduling that affect the response time. This can be confirmed by observing other heuristics generated by our approach.

$Heuristic_2$ *(+ (- (- (- (+ (+ ET MCT) (+ FCFS FCFS)) (MIN TS UC)) (/ MCT RT)) (- (+ (/ ECT (+ MCT ET)) (+ (/ ECT UC) (/ MCT RT))) (* ET ECT))) (- (* (MAX (MAX (MIN RR WT) (/ MCT RT)) MINMIN) (/ (+ MAXMIN BW) (* ET RT))) (- (+ (/ FCFS ET) (/ MCT RT)) (* TS ECT))))*

TABLE IV
OVERALL RESPONSE TIME OF ALL WORKFLOWS

	Scheduling Heuristic						
Workflow Application	**HLH-DSGP**	**GRP-HEFT**	**MINMIN**	**MAXMIN**	**MCT**	**FCFS**	**RR**
All Workflow Applications	**202.5749**±203	367.1283±663	238.3170±235	362.3809±655	261.3514±270	824.1127±1006	814.1152±1011

TABLE V
RESPONSE TIME OF SMALL WORKFLOWS

Virtual Machine Sets	**Workflows**	**HLH-DSGP**	**GRP-HEFT**	**MINMIN**	**MAXMIN**	**MCT**	**FCFS**	**RR**
Small	CyberShake	66.7304	98.4602	**63.8145**	98.4602	94.5456	87.6096	82.3248
	Inspiral	**494.2895**	544.0091	513.3909	520.0045	536.8247	523.0451	523.6858
	Montage	18.4727	17.7278	18.7283	**17.3355**	17.7742	20.3121	20.3121
	Sipht	**246.9994**	278.3650	724.6357	278.9296	567.3873	423.7029	423.7029
	Average	***206.6230***	234.6405	330.1424	228.6825	304.1330	263.6674 16	262.5064
Medium	CyberShake	35.7399	34.4604	35.1482	34.4604	**33.9127**	229.5193	229.5193
	Inspiral	275.4234	**260.0341**	278.3237	269.2729	265.8143	1526.6984	1526.6984
	Montage	13.1722	**11.1773**	11.2201	11.1787	11.1890	68.8822	68.8822
	Sipht	204.3352	**189.9652**	235.2857	190.1366	204.0337	448.7293	448.7293
	Average	132.1677	***123.9093***	139.9944	126.2621	128.7374	568.4573	568.4573
Large	CyberShake	31.7505	29.9080	30.1332	29.9080	**29.8882**	532.9814	532.9814
	Inspiral	240.5714	**235.5537**	240.1651	238.0728	236.9016	4562.5785	4562.5785
	Montage	11.9655	**9.7197**	9.7301	9.7210	9.7246	195.0576	195.0576
	Sipht	206.8612	190.2374	349.3161	**190.0496**	336.7262	3320.8919	3320.8919
	Average	122.7872	***116.3547***	157.3361	116.9379	153.3102	2152.8773	2152.8773

TABLE VI
RESPONSE TIME OF MEDIUM WORKFLOWS

Virtual Machine Sets	**Workflows**	**HLH-DSGP**	**GRP-HEFT**	**MINMIN**	**MAXMIN**	**MCT**	**FCFS**	**RR**
Small	CyberShake	**96.0438**	195.9446	98.9682	196.0306	122.7546	141.5640	117.3037
	Inspiral	**729.2235**	841.4161	753.1528	767.7413	800.8968	787.2573	791.4566
	Montage	31.9591	46.9481	**30.9770**	46.9456	48.8614	50.4574	50.5397
	Sipht	**396.3700**	2195.4448	543.6559	2197.1684	571.6696	554.3105	490.9751
	Average	***313.3991***	819.9385	356.6885	801.9715	386.0456	383.3973	362.5688
Medium	CyberShake	74.7184	76.6792	83.1832	76.6792	**63.0717**	246.6415	246.6415
	Inspiral	418.6810	**374.7427**	400.2263	425.4149	408.7063	1073.3435	1073.3435
	Montage	24.7563	**24.3939**	24.8250	**24.3939**	25.1723	49.5456	49.5456
	Sipht	235.2719	**218.3755**	346.4156	218.7968	266.7682	689.4385	689.4385
	Average	188.3569	***173.5478***	213.6625	186.3212	190.9296	514.7423	514.7423
Large	CyberShake	40.3718	**38.7641**	38.9426	**38.7641**	38.8718	685.2342	685.2342
	Inspiral	274.6592	**266.9070**	270.2236	274.0907	272.5656	2807.4500	2807.4500
	Montage	19.1336	17.4841	17.5649	**17.4839**	17.5384	128.8636	128.8636
	Sipht	225.9681	213.2453	238.8598	**213.2355**	311.9708	2023.9704	2023.9704
	Average	140.0332	***134.1001***	141.3977	135.8936	160.2367	1411.3795	1411.3795

We can see that $Heuristic_2$ supports claims that man-made heuristics such as MINMIN, MAXMIN, and FCFS can be exploited to design effective scheduling heuristics in conjunction with characteristics of workflow tasks and cloud resources.

VI. CONCLUSION

In this paper, we present a genetic programming hyper-heuristic approach for designing high-level heuristics for dynamic workflow scheduling in the cloud. The novelty of our paper is that we jointly use several well-known man-made heuristics to achieve a desirable balance among multiple aspects of the scheduling problem collectively, addressing the scheduling of workflows in a dynamic cloud environment where tasks arrive periodically. The experimental results show that our HLH-DSGP approach outperforms competing algorithms in terms of overall average response time and in majority of test scenarios. In future our proposed algorithm can be extended to consider multiple factors such as makespan, cost, and deadline as well as response time.

REFERENCES

[1] G. Ismayilov and H. R. Topcuoglu, "Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in

TABLE VII
RESPONSE TIME OF LARGE WORKFLOWS

Virtual Machine Sets	Workflows	HLH-DSGP	GRP-HEFT	MINMIN	MAXMIN	MCT	FCFS	RR
Small	CyberShake	**153.5627**	450.3095	160.7169	445.3601	438.5640	424.1226	174.3898
	Inspiral	**881.2027**	1302.5987	921.2085	1164.5218	1026.7099	1111.6516	1222.2140
	Montage	58.1716	111.2658	**57.5125**	116.8590	113.8773	114.8245	114.8277
	Sipht	**375.6201**	3447.8238	414.0185	3432.7381	1014.3447	995.5160	914.0878
	Average	***367.1393***	1327.9995	388.3641	1289.8698	648.3740	661.5287	606.3798
Medium	CyberShake	74.9315	124.9829	**72.2316**	125.9456	78.6619	149.5920	146.9610
	Inspiral	**471.5110**	489.5769	524.4360	506.2618	497.8213	811.8021	774.0954
	Montage	29.3360	**28.1836**	30.4831	28.7290	28.4786	35.4549	35.4477
	Sipht	**239.9618**	241.8290	363.1002	249.0398	269.4986	664.7077	664.6078
	Average	***203.9351***	221.1432	247.5627	227.4941	218.6151	415.3892	405.2780
Large	CyberShake	54.6094	58.8284	**47.6512**	58.8410	47.9414	383.3629	383.3629
	Inspiral	297.8330	318.8669	340.2255	301.1212	**289.3946**	1854.1305	1854.1305
	Montage	**33.0447**	34.3129	34.0599	33.9076	34.0083	78.4705	78.4705
	Sipht	209.4437	**198.0767**	256.8829	198.1116	275.7803	1866.3357	1855.4268
	Average	148.7327	152.5213	169.7049	***147.9954***	161.7812	1045.5749	1042.8477

TABLE VIII
FREQUENCY OF EACH TERMINAL

Terminal	TS	VS	UC	ET	WT	BW	ECT	$MAXMIN$	$MINMIN$	RR	$FCFS$	MCT
Frequency	2.6	1.9	0.93	2.8	2.0	1.6	2.1	2.3	2.9	1.2	1.8	1.5

cloud computing," *Future Generation Computer Systems*, vol. 102, pp. 307–322, 2020.

[2] J. Sahni and D. P. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 2–18, 2015.

[3] D. Jakobović, L. Jelenković, and L. Budin, "Genetic programming heuristics for multiple machine scheduling," in *European Conference on Genetic Programming*. Springer, 2007, pp. 321–330.

[4] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.

[5] D. M. Abdelkader and F. Omara, "Dynamic task scheduling algorithm with load balancing for heterogeneous computing system," *Egyptian Informatics Journal*, vol. 13, no. 2, pp. 135–145, 2012.

[6] Y. Yu, Y. Feng, H. Ma, A. Chen, and C. Wang, "Achieving flexible scheduling of heterogeneous workflows in cloud through a genetic programming based approach," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 3102–3109.

[7] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy, "Task scheduling strategies for workflow-based applications in grids," in *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, vol. 2. IEEE, 2005, pp. 759–767.

[8] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[9] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Genetic programming for job shop scheduling," in *Evolutionary and Swarm Intelligence Algorithms*. Springer, 2019, pp. 143–167.

[10] H. R. Faragardi, M. R. S. Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, "Grp-heft: A budget-constrained resource provisioning scheme for workflow scheduling in iaas clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1239–1254, 2019.

[11] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *2012 IEEE 8th International Conference on E-Science*. IEEE, 2012, pp. 1–8.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[13] M. Masdari, S. ValiKardan, Z. Shahi, and S. I. Azar, "Towards workflow scheduling in cloud computing: a comprehensive analysis," *Journal of Network and Computer Applications*, vol. 66, pp. 64–82, 2016.

[14] H. Y. Shishido, J. C. Estrella, C. F. M. Toledo, and M. S. Arantes, "Genetic-based algorithms applied to a workflow scheduling algorithm with security and deadline constraints in clouds," *Computers & Electrical Engineering*, vol. 69, pp. 378–394, 2018.

[15] V. Arabnejad, K. Bubendorfer, and B. Ng, "Budget and deadline aware e-science workflow scheduling in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 29–44, 2018.

[16] R. Ghafouri, A. Movaghar, and M. Mohsenzadeh, "A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds," *Peer-to-Peer Networking and Applications*, vol. 12, no. 1, pp. 241–268, 2019.

[17] A. Saraswathi, Y. R. Kalaashri, and S. Padmavathi, "Dynamic resource allocation scheme in cloud computing," *Procedia Computer Science*, vol. 47, pp. 30–36, 2015.

[18] K.-R. Escott, H. Ma, and A. Chen, "Genetic programming based hyper heuristic approach for dynamic workflow scheduling in the cloud," in *The 31st International Conference on Database and Expert Systems Applications - DEXA2020*. Springer, 2020.

[19] M. Rahman, R. Hassan, R. Ranjan, and R. Buyya, "Adaptive workflow scheduling for dynamic grid and cloud computing environment," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 13, pp. 1816–1842, 2013.

[20] A. Masood, Y. Mei, G. Chen, and M. Zhang, "Many-objective genetic programming for job-shop scheduling," in *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2016, pp. 209–216.

[21] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008.

[22] J. Xie, Y. Mei, A. T. Ernst, X. Li, and A. Song, "A genetic programming-based hyper-heuristic approach for storage location assignment problem," in *2014 IEEE congress on evolutionary computation (CEC)*. IEEE, 2014, pp. 3000–3007.

[23] J. Lin, L. Zhu, and K. Gao, "A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem," *Expert Systems with Applications*, vol. 140, p. 112915, 2020.