Genetic Programming Based Hyper Heuristic Approach for Dynamic Workflow Scheduling in the Cloud

Kirita-Rose Escott[1][0000−0001−5574−1023], Hui Ma[1][0000−0002−6232−4436], and Gang Chen[1]

Victoria University of Wellington, New Zealand
{kirita.escott, hui.ma, aaron.chen}@ecs.vuw.ac.nz

Abstract. Workflow scheduling in the cloud is the process of allocating tasks to limited cloud resources to maximise resource utilization and minimise makespan. This is often achieved by adopting an effective scheduling heuristic. Most existing heuristics rely on a small number of features when making scheduling decisions, ignoring many impacting factors that are important to workflow scheduling. For example, the MINMIN algorithm only considers the size of the tasks when making scheduling decisions. Meanwhile, many existing works focused on scheduling a static set of workflow tasks, neglecting the dynamic nature of cloud computing. In this paper, we introduce a new and more realistic workflow scheduling problem that considers different kinds of workflows, cloud resources, and impacting features. We propose a Dynamic Workflow Scheduling Genetic Programming (DSGP) algorithm to automatically design scheduling heuristics for workflow scheduling to minimise the overall makespan of executing a long sequence of dynamically arriving workflows. Our proposed DSGP algorithm can work consistently well regardless of the size of workflows, the number of available resources, or the pattern of workflows. It is evaluated on a well-known benchmark dataset by using the popular WorkflowSim simulator. Our experiments show that scheduling heuristics designed by DSGP can significantly outperform several manually designed and widely used workflow scheduling heuristics.

Keywords: Cloud Computing · Dynamic Workflow Scheduling · Genetic Programming.

1 Introduction

Cloud computing is a distributed computing paradigm, which enables the delivery of IT resources over the Internet and follows the pay-as-you-go billing method [13]. These resources are provided by cloud providers for cloud users to execute their application-specific *workflows*. Workflows are widely used models for modelling and managing complex distributed computations [16]. To maximise revenue, cloud users aim to minimise their operation costs by effectively using cloud resources to process their workflows and, in the meantime, achieve the best possible performance in workflow execution.

Workflow scheduling aims to allocate workflow tasks to cloud resources so that the overall performance, e.g., the makespan involved in executing a long sequence of dynamically arriving workflows, can be minimised. The heterogeneity of the workflow tasks and the wide range of available cloud resources make this an NP-hard optimisation problem[18]. No algorithms have ever been developed to satisfactorily solve this problem in an efficient and scalable manner. Therefore, heuristics are needed to find near-optimal solutions efficiently.

While designing any new heuristics, it is important to distinguish static workflow scheduling from its dynamic counterpart. In particular, static working scheduling requires the scheduling decision to be made *a priori* over all workflows pending for execution. On the other hand, dynamic working scheduling requires cloud resources to be allocated to ready tasks at runtime [1]. This is because new requests for workflow execution arrive at the cloud progressively making runtime resource allocation inevitable.

Several heuristics have been designed to address the dynamic workflow scheduling problem, such as *Heterogeneous Earliest Finish Time (HEFT)*, *MINMIN* [3] and *MAXMIN* [4]. All these heuristics have demonstrated effectiveness in some related studies [12]. However, they only rely on a very small number of features, such as task size, to make scheduling decisions. Hence important information that is vital for workflow scheduling, e.g., virtual machine speed, the waiting time, and the expected completion time of every pending task, have been completely ignored. Due to this reason, these heuristics only work well on specific scheduling problem instances and cannot be reliably applied to a variety of different workflows and cloud settings. It is hence highly desirable to design new heuristics with consistently good performance across many workflow scheduling scenarios.

When designing scheduling heuristics for the dynamic workflow scheduling problem, several properties must be jointly considered, including the size of tasks, the dependencies of tasks, the amount of available resources and the specifications of those resources [14]. Numerous combined use of these properties is possible in practice to construct the scheduling heuristic. It is hence difficult to manually design an effective heuristic that is capable of solving many complex scheduling problems consistently well in the cloud.

Genetic Programming (GP) is a prominent Evolutionary Computation (EC) technology for designing heuristics for various difficult problems [17]. Designing scheduling heuristics manually in the cloud is time-consuming and requires substantial domain knowledge. Hence, GP possesses a clear advantage in designing these heuristics automatically. Its effectiveness has already been demonstrated in many combinatorial optimization problems, including job shop scheduling and other resource allocation problems [9, 10, 15].

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

1. Develop a new *Dynamic Workflow Scheduling Genetic Programming* (DSGP) algorithm to automatically design scheduling heuristics for the dynamic workflow scheduling problem.
2. Conduct an extensive experimental evaluation of DSGP by using the Workflowsim simulator and a popular benchmark dataset with 12 different workflow applications. Scheduling heuristics designed by DSGP is further compared to several existing scheduling heuristics, including HEFT, RR, and MCT [7].
3. Analyse the experiment results to investigate essential properties of workflows and cloud resources with a substantial impact on the performance of designed scheduling heuristics.

It is infeasible to conduct a real-world evaluation of any scheduling heuristic designed by DSGP due to restrictions on hardware and computation cost. To cope with this challenge, this paper adopts a simulated approach based on the combined use of multiple simulation technologies, including GridSim [5], CloudSim [6], and WorkflowSim [7]. With the help of these simulators, we can accurately and quickly evaluate any scheduling heuristic, ensuring the efficient and effective operation of DSGP [7].

The rest of the paper is organised as follows: Section 2 presents previous work related to workflow scheduling in the cloud environment. Section 3 gives an overview of the workflow scheduling problem. Section 4 outlines the proposed DSGP approach. Section 5 presents the experimental evaluation results and analysis. The conclusions and future work are outlined in Section 6.

2 Related Work

Scheduling cloud resources in the application layer to serve the cloud user is a major topic in cloud resource management and scheduling research [19]. Thus there are many existing works on this topic.

Some research studies workflow scheduling as a static problem, which assumes that all the workflows are known at the time of scheduling. Some meta-heuristic approaches propose to make the given workflow to resources (VMs) in clouds.

Greedy algorithms have previously been proposed to solve the workflow scheduling problem in the cloud. Typically, these algorithms generate a singular heuristic based on the current state of the tasks and resources. Mahmood et al. [8] propose an adaptive GA approach that focuses on minimising processing and communication costs of tasks with hard deadlines, as well as a greedy algorithm to be used for comparison. Greedy algorithms are known as simple and relatively quick, however, they do not always produce the best solution [8].

There are many existing workflow scheduling algorithms that consider different constraints with the aim to optimise cloud objectives such as makespan, cost, and application performance. Arabnejad et al. [2] 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 (IaaS). Yu et al. [18] propose a Flexible Scheduling Genetic Programming (FSGP) that employs GP to generate scheduling heuristics for the workflow scheduling problem. While these algorithms produce promising results, they are only able to solve static workflow scheduling problems. In the cloud services can be dynamically added or removed at any given time. This means that services can be added on the go and while working with workflows a number of resources can be defined during runtime [11]. This means that the aforementioned algorithms may not perform well, if at all, in a dynamic cloud environment.

As far as we know, GP has never been applied to the dynamic workflow scheduling problem. However, due to the success of GP being used in similar problems, we are confident that GP can evolve heuristics capable of producing a near-optimal resource allocation. Our proposed DSGP will generate scheduling heuristics for the diverse collection of workflows in a cloud with the aim to minimise the overall makespan.

3 Preliminaries

3.1 Problem Overview

In the workflow scheduling problem, a workflow 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. Assuming 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 makespan of the execution.

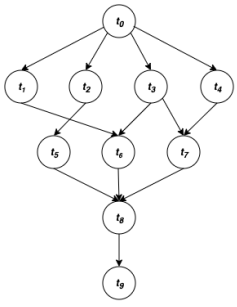

Fig. 1. A DAG with 10 Tasks in the Workflow

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.

3.2 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).

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

The size s_i of task t_i divided by the speed m_k of virtual machine v_k.
Each task t_i can be allocated to a virtual machine v_k once all of the parents $aParent(t_i)$ have completed processing. The allocation time, denoted by AT_i, is given by Equation (2).

$$AT_i = max_{p \in aParent(t_i)} FT_p \tag{2}$$

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 all the parent tasks of t_i are completed, 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 (3).

$$ST_i = \max\{FT_{t_{prev}}, AT_i\} \tag{3}$$

The waiting time of task t_i, denoted WT_i, is given by Equation (4), as the difference between the arrival time and the start time of a task.

$$WT_i = ST_i - AT_i \tag{4}$$

The relative finish time of task t_i is given by Equation (5).

$$RFT_i = WT_i + ET_{ik} \tag{5}$$

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 makespan of a workflow MS_T with tasks is the time taken to process all tasks in workflow T. The time taken is calculated by subtracting the earliest start time ST from the latest finish time FT, as shown in Equation (7).

$$MS_T = \max FT - \min ST \tag{7}$$

The total number of tasks TNS defined in Equation (8) sums the number of tasks N in workflow j of all Num workflows. TMS is the sum of makespan MS_T, of workflow j of all Num workflows, as shown in Equation (9).

$$TNS = \sum_{j=1}^{Num} Nj \tag{8}$$

$$TMS = \sum_{j=1}^{Num} MS_T j \tag{9}$$

The overall average makespan of all test workflows AMS is obtained by dividing TMS by TNS, as presented in Equation (10).

$$AMS = \frac{TMS}{TNS} \tag{10}$$

The objective of the dynamic workflow scheduling problem is to minimize the overall average makespan of all the workflows. i.e., min AMS. We aim to find a rule r, which produces a schedule with minimal AMS. The fitness of a rule r is evaluated by the AMS, of all workflows scheduled using the rule. Fitness is defined in Equation (11).

$$fitness(r) = AMS(r) \tag{11}$$

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

4 Design Scheduling Heuristic through Genetic Programming

In this section, we present our new DSGP algorithm to solve the workflow scheduling problem. We will first describe the outline of the algorithm. We then describe the representation of the heuristics generated by DSGP, as well as the function set and the terminal set to be used by DSGP. Finally, we introduce the genetic operators to evolve new scheduling heuristics, including *crossover, mutation*, and *reproduction*.

4.1 Outline of DSGP

DSGP aims to evolve scheduling heuristics to help the Workflow Scheduler to schedule cloud resources for workflow execution. As mentioned in the introduction, we rely on the Workflowsim simulator to simulate the processing of dynamically arriving workflows over an extensive period of time [7]. In Workflowsim, the Workflow Scheduler obtains all *ready tasks* regularly from the Workflow engine. A task is ready if and only if all of its parent tasks in a workflow have been processed by virtual machines in the cloud. Given this, we decide to prioritise

those tasks with a higher number of children in the corresponding workflows. This is because, once they are completed, a larger number of child tasks will become ready for processing by the Workflow Scheduler, ensuring the cloud to make continued progress in executing these workflows. In comparison, processing a large (or time-consuming) task with no children can often cause delays in the execution of all pending workflow, therefore increasing the overall makespan [18].

Being a hyper-heuristic approach, DSGP consists of a *training phase* and a *testing phase*. During the training phase, DSGP is designed to evolve a scheduling heuristic, driven by the goal to minimize the total makespan involved in executing a group of workflows by using a fixed collection of cloud resources (i.e., VMs in cloud data centers). The best heuristic discovered by DSGP during training is subsequently utilized in the testing phase to schedule various workflows.

As discussed, DSGP is designed to search for effective scheduling heuristics to solve the dynamic workflow scheduling problem. The pseudo-code in Algorithm 1 details the algorithmic design of DSGP.

In our experimental studies, the training set for DSGP consists of 12 different workflow types and a fixed collection of cloud resources. DSGP follows a standard GP framework with an embedded simulation as the evaluation process. The output of the training stage is the best scheduling heuristic/rule discovered by DSGP. This heuristic will be further evaluated in the testing stage to determine its true usefulness for scheduling various workflows in simulated cloud environments.

4.2 Representation

We represent a scheduling heuristic in the form of a GP tree. Example GP trees are illustrated in Fig 2 and 3. The terminal nodes of the GP tree capture a range of properties concerning workflows and cloud resources. Our study of relevant research shows that those are the most important properties to be considered by a scheduling heuristic to minimize the makespan of workflow execution. The intermediate nodes of the GP tree are arithmetic functions, as summarized in Table 1.

4.3 Terminal Set and Functional Set

Demonstrated in Table 1, the terminals used to represent properties of workflows and cloud resources, in this case, virtual machines, that have effects on the total makespan of a workflow. Properties such as the size of a task, the execution time of a task and, the speed of a virtual machine. The function set, also described in Table 1, denotes the operators used; addition, subtraction, multiplication, and protected division. Protected division means that it will return 1 if divided by 0.

Algorithm 1: DSGP 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
        TMS = 0;
        for a workflow in a set of workflows do
            Initialise the simulator;
            Prioritise tasks in set of ready tasks;
            for a task in a set of ready tasks do
                vm = vmSelection(vms, r);
                if vm is idle then
                    allocate(task, vm);
                end
            end
            TMS += calculateMakespan(workflow);
        end
        calculate AMS
        fit = fitness(AMS)
    end
    TournamentSelection;
    Crossover;
    Mutation;
    Reproduction;
end
return the best rule
```

4.4 Crossover, Mutation, and Reproduction

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

Crossover refers to the genetic operator which recombines the genetic information of two parents to create a new offspring. This is demonstrated in Fig 2, where the left branch of Parent A is combined with the right branch of Parent B to become offspring A'. Likewise, 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. This is demonstrated in Fig 3, where 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 [18].

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.

Table 1. The Terminal and Function Set for the Proposed FSGP Approach

Terminal Name	**Definition**
TS	The total size of a task t_i
VS	The speed of a virtual machine v_j
ET	Execution time of a task t_i
WT	Waiting time of a task t_i
RFT	Relative finish time of a task t_i
ECT	Expected completion time of a task t_i
Function Name	**Definition**
Add, Subtract, Multiply	Basic arithmetic operations (+, -, ×)
Protected Division	Protected division, return 1 if the denominator is 0 (%)

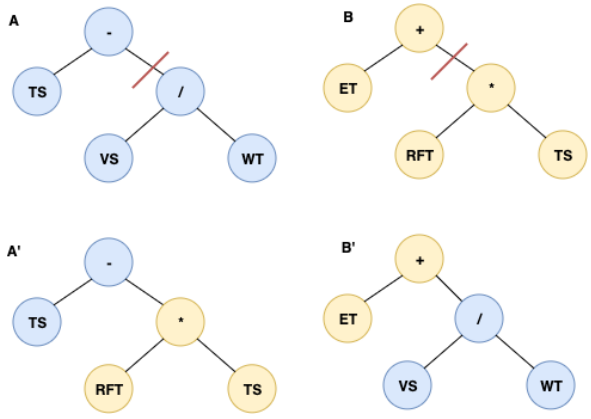

Fig. 2. Crossover Example

5 Experimental Evaluation

In this section, we describe the experiments we conducted to evaluate the DSGP algorithm. We can perform this evaluation by using the WorkflowSim simulator to simulate a real cloud environment and execute each of the workflows using the generated heuristic in that environment. As our approach aims to generate scheduling heuristics which minimise the average total makespan, the fitness of a generated heuristic is determined by the normalised average total makespan of all the workflows scheduled by the heuristic.

5.1 Dataset

Four benchmark workflow patterns are used in our experiments. Fig 4 demonstrates visualisations of the CyberShake, Inspiral, Montage and SIPHT workflows. We used three different sizes for each of the patterns, medium, large and,

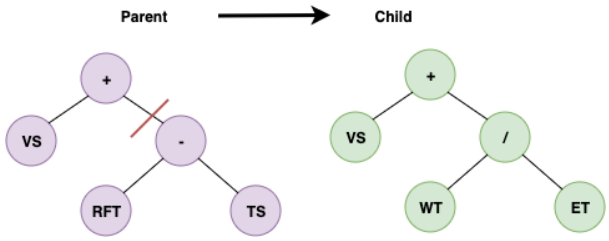

Fig. 3. Mutation Example

extra-large, described in Table 2. Thus, we tested 12 workflows in total, with the number of tasks ranging between 50 and 1,000.

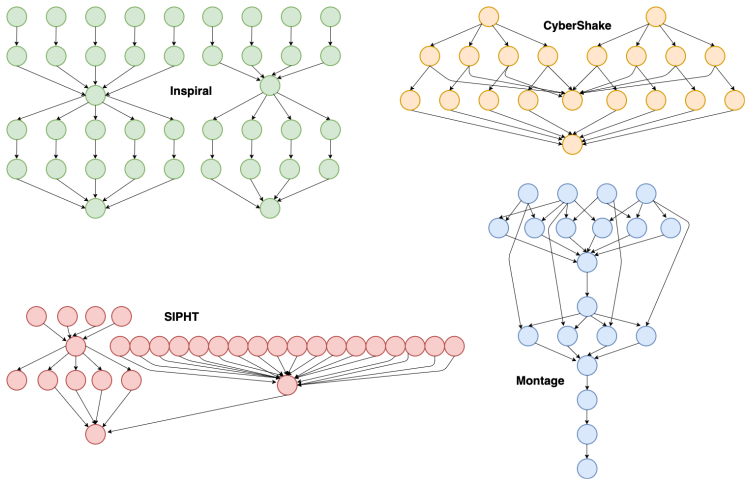

Fig. 4. Visual Representation of Workflow Applications

5.2 Simulator

We used the WorkflowSim simulator in our experiments to evaluate the effectiveness of the generated heuristics. The configuration settings are described in Table 3. CondorVM extends the Cloudsim VM, it has a local storage system and can indicate whether it is busy or not [7]. In WorkflowSim, VMs are created with the same parameters. We have VMs of small, medium, large and extra large size. We also use three different sets of VMs in our experiments. We have a small set of 4, which has 1 each of the different sized VMs, a set of 16, which has 4 of each VM and a 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.

The storage size is 10GB, the RAM is 512MB, Network Bandwidth is 10Mbit/s and we use one CPU. We use the local file system, as opposed to the shared file

Table 2. Number of Tasks in Workflow Applications

Application Size	CyberShake	Inspiral	Montage	SIPHT
Medium	50	50	50	60
Large	100	100	100	100
Extra Large	1,000	1,000	1,000	1,000

system which has only one storage for the data center. WorkflowSim also has a Clustering Engine which merges tasks into Jobs which are processed together, in our experiments we disable this feature.

Table 3. Configuration of Simulation Environment

Datacenter/VM Configuration	
Type	CondorVM
Storage Size/RAM	10GB/512MB
Network Bandwidth	10Mbit/s
# CPUs	1
VM Ratio Size	{Small, Medium, Large, X-Large}
# VMs	{8, 16, 32}
Simulator Configuration	
File System	Local
Clustering	Disabled
Overheads/Failures	Disabled

5.3 Baseline Algorithms

WorkflowSim has dynamic implementations of traditional workflow scheduling algorithms. We examined those implementations and give a summary of each below [1] [4][3][7].

- HEFT always selects the fastest available virtual machine to schedule a task to. We implemented our own version of HEFT in WorkflowSim for comparison
- 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

5.4 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[9]. The tournament size for tournament selection is 7 and the maximum depth of a tree is set to 17. 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.

5.5 Results

The results of our experiments show that DSGP outperforms competing algorithms. Table 4 shows the average makespan of each of the algorithms for all workflow application scenarios. The smaller the value, the better the performance of the algorithm. We see that FCFS has the worst performance, and DSGP has the best performance. We calculate the average makespan for DSGP as the mean average makespan over 30 independent runs.

Table 5 compares the results of all approaches for all of the testing scenarios. We can see that DSGP achieves the best results on more of the test scenarios than the competing approaches. We see that the RR and FCFS approaches consistently performed the worst.

Of all the approaches, HEFT performed secondary to our DSGP approach. In both the number of test scenarios that it achieved the best results and in the overall average makespan of all test scenarios. MINMIN, MAXMIN, and MCT, achieved the best results for 6 of the 36 test scenarios between them. Whereas, RR and FCFS did not achieve the best results for any of the test scenarios.

Table 4. Average Makespan of All Workflow Applications

	Scheduling Heuristic						
Workflow Application	**DSGP**	**HEFT**	**MINMIN**	**MAXMIN**	**RR**	**FCFS**	**MCT**
All Workflow Sizes {Medium, Large, X Large}	**36.90** ± 14.55	37.41 ± 13.48	57.84 ± 24.34	40.49 ± 13.75	207.01 ± 118.37	207.52 ± 118.36	48.97 ± 19.97

5.6 Discussion

We analysed the terminals of the best heuristic rules generated in each run to determine which terminals affected the performance of the heuristics generated by DSGP. We found that there were some purposeless combinations such as "$(ECT - ECT)$", which have no impact on the trees and were removed from the analysis. From here, we counted the frequency of terminal appearances, shown in Table 6.

Table 5. Average Makespan of Individual Workflow Application Scenarios

Workflow	Application Size	Num VMs	DSGP	HEFT	MINMIN	MAXMIN	RR	FCFS	MCT
		Small	15.9989	**14.1131**	23.211	15.3275	15.4486	16.7487	18.0751
		Medium	**14.0997**	14.2128	16.3827	14.2584	15.5911	15.3292	14.1279
	Medium	Large	**6.0264**	9.4706	9.4792	9.441	9.5293	9.4523	9.4436
		Small	13.9852	**7.9588**	25.4003	18.4432	87.3345	87.3345	13.9744
		Medium	**8.0987**	5.707	7.7164	7.0453	29.0844	29.0844	9.6625
	Large	Large	6.5287	**2.8697**	3.4996	2.9647	4.8269	4.9214	3.4289
		Small	9.135	8.0487	6.1173	**5.6271**	237.518	237.518	5.9525
		Medium	**3.0255**	4.8862	4.8939	5.9262	89.8922	89.8922	4.8771
Cybershake	X Large	Large	2.5018	**1.4865**	2.1634	3.5278	11.4675	11.4675	2.4932
		Small	**103.4092**	112.5431	162.6154	136.9769	129.1874	120.7618	118.9919
		Medium	**52.6484**	91.7377	102.0764	106.6661	88.306	90.5894	89.8665
	Medium	Large	**31.882**	94.566	94.3443	94.4015	96.2813	94.5207	95.4195
		Small	85.9817	70.4122	**56.6749**	69.0277	332.6658	332.6658	61.3014
		Medium	60.6875	**41.4328**	112.5836	63.0693	166.9795	193.6208	89.5669
	Large	Large	**21.7495**	28.4321	36.2024	32.1388	33.4726	33.5923	32.7614
		Small	91.244	77.2779	32.2015	33.1887	1063.3361	1063.3361	**31.882**
		Medium	28.584	40.2824	29.4118	22.712	743.3518	743.3518	**22.9864**
Inspiral	X Large	Large	30.5142	**14.6363**	67.3942	53.9798	58.3216	55.1827	68.513
		Small	**4.5023**	4.7461	4.9686	4.8121	6.3935	6.3795	4.8058
		Medium	**3.9144**	4.8263	5.7214	5.1894	6.5424	6.5337	5.1818
	Medium	Large	**1.9999**	4.9277	4.9664	4.9572	6.2118	6.2125	4.9568
		Small	5.183	**2.3159**	5.1986	5.0767	15.8117	15.8117	5.1466
		Medium	5.0404	**2.0527**	5.2787	5.5415	8.9132	8.9131	5.1955
	Large	Large	4.7851	**1.7169**	2.0262	1.8659	3.788	3.7858	1.7486
		Small	4.8342	**2.0076**	2.7838	2.6907	44.1525	44.1525	2.7691
		Medium	**1.3619**	1.4293	7.2686	7.2607	23.2355	23.2355	7.1617
Montage	X Large	Large	1.8457	**1.0318**	4.9664	1.8851	3.7543	3.7543	1.8539
		Small	204.9747	**132.0703**	362.3385	131.4414	301.155	301.155	298.8751
		Medium	**84.9519**	98.2268	165.3731	109.2362	180.0472	180.0423	127.7043
	Medium	Large	84.3457	**75.4759**	84.6456	75.8628	76.365	77.6026	76.7125
		Small	114.3366	106.0498	178.111	**78.6599**	345.7508	345.7508	120.9216
		Medium	59.5024	61.0234	138.9487	**45.6752**	224.5103	224.5103	62.1764
	Large	Large	48.0613	**23.4687**	47.6233	43.0831	46.4149	50.6955	37.3639
		Small	**72.3466**	107.3811	93.8539	79.6473	1541.926	1541.926	144.3428
		Medium	**24.4861**	61.7895	90.9403	46.2486	1220.3527	1220.352	82.0233
SIPHT	X Large	Large	**16.1024**	16.1154	84.7539	113.6452	184.5964	180.3763	80.6106

Table 6. Frequency of Each Terminal

Terminal	VS	ET	ECT	RFT	WT	TS
Frequency	364	487	544	486	479	507

We can observe from Table 6 that the most frequently occurring terminals are expected completion time (ECT) and task size (TS). In the following, we show one of the heuristic rules generated by our DSGP to analyse the relationship between the terminals and the heuristics rules.

$$Heuristic_1 = (((WT - TS) + (ECT * RFT)) - ((ET - VS) - (ET * WT)))$$

In this paper, we schedule tasks according to the value calculated by the heuristic. The smaller the value is, the faster it will be executed. From the $Heuristic_1$, we observe that smaller ECT and a higher VS can lead to a smaller value calculated by this heuristic. As RFT and ET are the third and fourth most frequently occurring terminals, we know they have also played a role in many of the heuristic rules generated by DSGP. This means that a heuristic rule containing RFT will have good performance. Unlike other heuristics, the heuristics generated by DSGP contain more features that affect the makespan. This can be confirmed by observing other heuristics generated by our approach.

$$Heuristic_2 = (((RFT * WT) - (ET - ECT)) + ((VS + VS) + (ET + ET)))$$

We can see that $Heuristic_2$ supports the claim that a smaller ECT and a higher VS can lead to a smaller value calculated by the heuristic, therefore, a quicker execution for a task.

6 Conclusion

In this paper, we present a genetic programming hyper-heuristic approach for dynamic workflow scheduling in the cloud. The novelty of our paper is that our approach addresses the scheduling of workflows in a dynamic cloud environment where tasks arrive periodically. The experimental results show that our DSGP approach addresses the workflow scheduling problem for multiple workflow patterns with varied types of resources in a dynamic environment. Our DSGP approach outperforms competing algorithms in a majority of test scenarios and in terms of the overall average makespan for all test scenarios. In the future, our proposed algorithm can be extended to consider multiple factors, such as cost, security levels, and deadline as well as makespan.

References

1. Abdelkader, D.M., Omara, F.: Dynamic task scheduling algorithm with load balancing for heterogeneous computing system. Egyptian Informatics Journal **13**(2), 135–145 (2012)
2. Arabnejad, V., Bubendorfer, K., Ng, B.: Budget and deadline aware e-science workflow scheduling in clouds. IEEE Transactions on Parallel and Distributed Systems **30**(1), 29–44 (2018)
3. Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., Kennedy, K.: Task scheduling strategies for workflow-based applications in grids. In: CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005. vol. 2, pp. 759–767. IEEE (2005)
4. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., 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 **61**(6), 810–837 (2001)

5. Buyya, R., Murshed, M.: Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and computation: practice and experience **14**(13-15), 1175–1220 (2002)
6. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience **41**(1), 23–50 (2011)
7. Chen, W., Deelman, E.: Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In: 2012 IEEE 8th International Conference on E-Science. pp. 1–8. IEEE (2012)
8. Mahmood, A., Khan, S.A., Bahlool, R.A.: Hard real-time task scheduling in cloud computing using an adaptive genetic algorithm. Computers **6**(2), 15 (2017)
9. Masood, A., Mei, Y., Chen, G., Zhang, M.: Many-objective genetic programming for job-shop scheduling. In: 2016 IEEE Congress on Evolutionary Computation (CEC). pp. 209–216. IEEE (2016)
10. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. IEEE Transactions on Evolutionary Computation **17**(5), 621–639 (2012)
11. Raghavan, S., Sarwesh, P., Marimuthu, C., Chandrasekaran, K.: Bat algorithm for scheduling workflow applications in cloud. In: 2015 International Conference on Electronic Design, Computer Networks & Automated Verification (EDCAV). pp. 139–144. IEEE (2015)
12. Rahman, M., Hassan, R., Ranjan, R., Buyya, R.: Adaptive workflow scheduling for dynamic grid and cloud computing environment. Concurrency and Computation: Practice and Experience **25**(13), 1816–1842 (2013)
13. Sahni, J., Vidyarthi, D.P.: A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. IEEE Transactions on Cloud Computing **6**(1), 2–18 (2015)
14. Sonmez, O., Yigitbasi, N., Abrishami, S., Iosup, A., Epema, D.: Performance analysis of dynamic workflow scheduling in multicluster grids. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. pp. 49–60 (2010)
15. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers & Industrial Engineering **54**(3), 453–473 (2008)
16. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M., et al.: Workflows for e-Science: scientific workflows for grids, vol. 1. Springer (2007)
17. Xie, J., Mei, Y., Ernst, A.T., Li, X., Song, A.: A genetic programming-based hyper-heuristic approach for storage location assignment problem. In: 2014 IEEE congress on evolutionary computation (CEC). pp. 3000–3007. IEEE (2014)
18. Yu, Y., Feng, Y., Ma, H., Chen, A., Wang, C.: Achieving flexible scheduling of heterogeneous workflows in cloud through a genetic programming based approach. In: 2019 IEEE Congress on Evolutionary Computation (CEC). pp. 3102–3109. IEEE (2019)
19. Zhan, Z.H., Liu, X.F., Gong, Y.J., Zhang, J., Chung, H.S.H., Li, Y.: Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Computing Surveys (CSUR) **47**(4), 63 (2015)