Achieving Flexible Scheduling of Heterogeneous Workflows in Cloud through a Genetic Programming Based Approach

Yongbo Yu, Yalian Feng, Hui Ma, Aaron Chen, Chen Wang
School of Engineering and Computer Science,Victoria University of Wellington, New Zealand
Email: yuyong1 | fengyali | hui.ma | aaron.chen | chen.wang@ecs.vuw.ac.nz

***Abstract*—Cloud computing enables enormous computational resources to be scheduled as parallel workflow applications. Most traditional heuristics can only solve one particular scheduling problem. For example, Heterogeneous Earliest Finish Time (HEFT) and Greedy algorithms allocate resources to given ordered list of tasks using a specific single heuristic, which only caters for a specific scheduling problem, e.g. the fixed number of tasks in a workflow and available resources. Many researchers considered the heterogeneous workflows and cloud resources in scheduling in order to minimize the cost and makespan, but the solutions provided are only for specific workflow pattern. In this paper, we demonstrate a workflow scheduling problem which considers the combination of heterogeneous workflows as well as heterogeneous computing resources. We proposed Flexible Scheduling using Genetic Programming (FSGP) approach to minimize the total cost and makespan of heterogeneous workflows in the cloud. The performance of our proposed FSGP is regardless of the number of tasks in the workflow, available resources and workflow patterns. We evaluated our proposed approach using a benchmark dataset. Performance evaluation of some well-known algorithms such as HEFT and greedy algorithms exhibit that our FSGP approach perform better than other competing algorithms.**

I. Introduction

Cloud computing provides various resources as services including Infrastructure as a Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) [1]. Users can rent cloud resources on pay-per-use basis [2]. The success of cloud computing requires cloud providers to carefully schedule workflow tasks and allocate scarce resources with an ultimate goal such as reducing the operating costs to maximize revenue while maintaining or even improving the service quality.

Such a workflow scheduling problem is widely known as NP-complete [3]. There are many factors that affect the performance of workflow scheduling especially in the cloud environment, such as the pattern of a workflow, the number of available resources, the size of workflow tasks and task dependencies. In particular, a variety of workflow tasks may pose a significantly varied demand for computing resources. Any scheduling heuristics designed for specific workflows may cause unbalanced load in the cloud which in turn, affects the operation cost. Additionally, the cloud infrastructure must often support a diverse collection of workflows simultaneously [4]. Inevitably, the processing of one workflow in the cloud will affect the processing (or execution) of other workflows in an intricate manner, potentially affecting the efficiency of the entire cloud infrastructure. Clearly, it is challenging to design heuristics manually that are suitable for solving complex scheduling problems in the cloud environment.

Several techniques have been applied to solve workflow scheduling problem, such as Heterogeneous Earliest Finish Time (HEFT) [5], Particle Swarm Optimization (PSO) [6] Genetic Algorithm (GA) and greedy algorithm [1]. These approaches can produce promising results, but a great majority of these techniques only handles one objective like minimizing makespan or minimizing total cost. Further, most of these algorithms do not consider different types of virtual machines. Also, these techniques can only provide exact scheduling solution for a particular workflow pattern. Thus these existing algorithms can not handle heterogeneous workflow effectively. Flexible heuristics are needed to effectively and efficiently solve the workflow scheduling problem which considers a diverse collection of heterogeneous workflows as well as heterogeneous available resources in the cloud. Genetic Programming has been used to automatically generate heuristics.

Genetic Programming (GP) is a global search technique that is widely used for scheduling problems [7], [8] to find near-optimal dispatching rules as solutions. For example, researchers applied GP to generate scheduling policies in the job shop scheduling problems [9], [10]. Though these GP-based approaches for job shop scheduling show promising results, we cannot apply them to solve workflow scheduling problems. The workflows of the scheduling problem are more complex than the job patterns, because workflow patterns are more complex than job shop scheduling, and the number of VMs is much larger than the number of machines considered in the job shop scheduling problem. A workflow pattern is a blueprint for building workflows, which refers to a structured flow of activities to be performed explicitly by using resources. Any two workflows derived from the same workflow pattern will share the same internal structure.

In this paper, we propose a Flexible Scheduling Genetic Programming (FSGP) approach that employs GP to generate scheduling heuristics for the workflow scheduling problem. As a hyper-heuristic approach, FSGP is made up of two phases, i.e., training and testing. During training, a training set of workflows is first constructed based on multiple different workflow patterns. These workflows are further utilized to create a series of scheduling scenarios for training. Using these scenarios, FSGP will evolve a group of heuristics that can produce varied trade-offs between operation cost and makespan when applying them to schedule workflow execution in all training scenarios. Finally, the most effective heuristic evolved

by FSGP will be identified. This heuristic will be further used in the testing phase to schedule workflow execution in any testing scenarios. Every testing scenario is constructed in the same way as a training scenario. However, FSGP does not have access to any testing scenario while evolving scheduling heuristics during the training phase.

In this paper, we aim to automatically design workflow schedule heuristics, which can be subsequently utilized to solve the workflow scheduling problem. Our FSGP approach is suitable to solve such a complicated scenario which considers a diverse collection of heterogeneous workflows as well as heterogeneous available resources. There are three main objectives of this paper:

1) Develop FSGP to automatically design heuristics for the heterogeneous workflow scheduling problem.
2) Conduct experimental evaluation of the proposed FSGP approach, using datasets consisting workflows of five benchmark workflow patterns with various sizes, in comparison with many existing scheduling techniques such as HEFT [5] and RR in terms of both operating cost and makespan.
3) Analysis the results to investigate the essential properties of workflow and virtual machines that contribute to the heuristics generated by FSGP to further understand the heuristics.

The rest of paper is organized as follows: Section 2 presents the background of workflow scheduling in a cloud and some related works. Section 3 presents a description of the novel workflow scheduling problem. Section 4 depicts the proposed FSGP approach. Section 5 reports on the experimental evaluation results, as well as analysis of the results. Finally, conclusions and future work are outlined in Section 6.

II. Background & Related Work

The workflow scheduling problems in a cloud are known as NP complete [3]. Traditional algorithms such as HEFT [11], [5] and greedy algorithms [1] have been proposed to solve the workflow scheduling problems. These algorithms generate a single heuristic by making the best choice in view of the current status of the tasks and available resources. Unfortunately, these algorithms are found to lead to problems such as load unbalancing and poor scalability in clouds [5], [8], [12]. Mahmood et al. [1] proposed an Adaptive GA that focuses on minimizing the total processing and communication costs with hard deadlines of tasks. They also proposed a greedy algorithm which was used to compare with the Adaptive GA. Greedy algorithms are best known for simple implementation and relatively fast to provide a solution. However, greedy algorithms may not always give good solutions [1]. Liu et al. [10] proposed a co-evolutionary GA with deadline constraint. Dai et al. [13] proposed a workflow scheduling algorithm called MQoS-GAAC, which integrates the ant colony optimization (ACO) and GA. They assumed tasks are independent rather than belonging to some workflows that give them dependencies. The Fussy-PSO [12] and NSPSO[14], two PSO based approach achieve the balance of execution time and cost in the workflow scheduling. However, these approaches simplify the problem by ignoring the dependencies between "tasks" in workflows. The dependencies of tasks occur in most real problems on workflow scheduling problem in cloud computing and should be considered.

There are many research study heterogeneous workflows in cloud [15], [16], [17], [18]. Heterogeneous workflow is a workflow contains different structures of basic workflow patterns such as CyberShake, Montage and Epigenomics etc. However, the experimental results only show the difference between competing algorithms on specific workflow pattern (e.g. Montage) rather than the combination of different workflow patterns (e.g. a combination of CyberShake and Montage). Meanwhile, though different types of resources are considered, only one instance of each virtual machine type is used. In the real-world cloud environment, different workflows must be jointly supported by different types of virtual machines. Situations frequently change in the cloud environment, such as the number of processing tasks and available computing resources. Due to the changes in the complex cloud environment, manually generate scheduling heuristics requires a high level of expertise and is also time-consuming. Therefore, it is more desirable to seek ways of designing heuristics automatically. One prominent algorithm that can help us achieve this goal is GP, which is often used as a hyper-heuristic to generate heuristics for combinatorial optimization problems[19], [20]. For example, GP has shown its effectiveness and efficiency in generating heuristics for job shop scheduling problem[21], [22], [23]. Mei et al. [24] employed GP to show competitive performance for Traveling Thief Problem (TTP) which combines Traveling Salesman Problem (TSP) and Knapsack Problem (KP). In addition, heuristics generated by GP in Storage Location Assignment Problem (SLAP) have good reusability in slightly different scenarios [7].

GP has never been applied to the cloud workflow scheduling problem domain in the past but the algorithm has great potential due to its previous success on similar problems. We believe GP can evolve heuristics suitable for the workflow scheduling problem with the consideration of several new factors such as the complexity of workflow patterns and computing resources. Our proposed FSGP will generate scheduling heuristics for the diverse collection of heterogeneous workflows in a cloud with the aim to minimize a weighted sum of cost and makespan.

III. Cloud Workflow Scheduling Problem

A. Problem Overview

Generally, a workflow can be depicted with a directed acyclic graph (DAG) as shown in Fig. 1. A DAG is defined as $G = (T, E)$, where T is a set of tasks and E is a set of directed edges. In the workflow $e_{ij} \in E$ denotes execution precedence constraints, i.e., the child task t_j cannot be executed before the completion of its parent task t_i. On the other hand, parallel tasks can be executed concurrently whenever there are idle virtual machines. As can be observed in Fig. 1, task t_2 can only be executed after task t_0 has been executed; task t_9 can only be executed, when all of its parents (t_6, t_7, t_8) have been

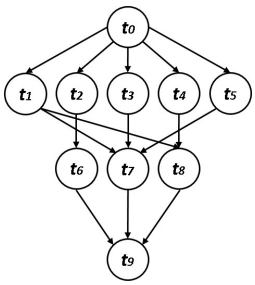

Fig. 1. A DAG with 10 Tasks in the Workflow.

executed. After t_0 has been executed, t_1, t_2, t_3, t_4 and t_5 can be executed at the same time if there are more than 5 idle virtual machines.

Tasks without parents are called *entry tasks* and tasks without children are *exit tasks*. For example in Fig. 1 task t_0 is the entry task and task t_9 is the exit task of the workflow.

The process of workflow scheduling is to allocate each pending task of one or multiple workflows to idle virtual machines in the cloud.

There are several key $constraints$ that are vital to our workflow scheduling problem and its solution.

- Each virtual machine can only process at most one task at any given time.
- Each task can be assigned to any virtual machine. Once a task starts its processing on a virtual machine, the selected virtual machine will continue to process it until completion. During this period, the selected virtual machine cannot be interrupted by other tasks.
- Only idle machines can be deployed to process new tasks.

In this paper we consider the workflow scheduling problem with three heterogeneous factors, the types and number of VMs, the patterns of workflows and the ways of combination to conduct the workflows which are used as training set and testing datasets.

B. Problem Formulation

We first present in Table I the basic properties of tasks and virtual machines that will be used to formulate the problem.

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

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

Each task t_i can be allocated to a virtual machine m_k once all its parent tasks are finished. The allocation/deployment 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 is decided either by the time that virtual machine m_k becomes idle, or when all the parent tasks of t_i are completed, whichever is later. The actual start time, denoted by ST_i, is given by Equation (3).

$$ST_i = max\{FT_{Pre(t_i)}, AT_i\} \tag{3}$$

TABLE I
NOTATIONS FOR HETEROGENEOUS WORKFLOW SCHEDULING PROBLEM

Notation	Description
M	Total number of virtual machines
VM	Set of of virtual machines
m_k	A virtual machine. $m_k \in VM$
v_k	Speed of virtual machine m_k
c_k	Cost of virtual machine m_k per unit of time
N	Total number of tasks
T	Set of of tasks
t_i	A task. $t_i \in T$
s_i	Size of task t_i
$Pre(t_i)$	Previous task of t_i executed on the same virtual machine
$aParent(t_i)$	All immediate predecessors of task t_i
ET_{ik}	The execution time of task t_i on virtual machine m_k
EC_k	The execution cost of all tasks on virtual machine m_k
EC_p	The execution cost of all virtual machines in the p^{th} scenario
ST_i	Start time of task t_i
FT_i	Finish time of task t_i
RFT_i	Relative time of task t_i
AT_i	Allocation time of task t_i assigned to virtual machine m_k
WT_i	Waiting time of task t_i to be executed on virtual machine m_k
Num	Total number of experimental scenarios
ATC	Average total cost of experimental scenarios
AMS	Average makespan of experimental scenarios

The task waiting time, denoted by WT_i as given by Equation (4), is the time difference between the actual start time and the allocation time.

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

The relative finish time of task t_i is given by Equation (5) and the finish time of task t_i is calculated by Equation (6).

$$RFT_i = WT_i + ET_i \tag{5}$$

$$FT_i = ST_i + ET_i \tag{6}$$

If a cloud service provider charges the user on actual usage, the cost on a particular virtual machine m_k is given by Equation (7). RFT_{total} is the difference between the finish time of the last task and the start time of the first task executed in m_k.

$$EC_k = RFT_{total} * c_k \tag{7}$$

where

$$RFT_{total} = FT_N - ST_1 \tag{8}$$

The average total cost of experimental scenarios is then given by Equation (9).

$$ATC = \frac{\sum_{p=1}^{Num} \sum_{k=1}^{M} EC_{kp}}{Num} \tag{9}$$

The makespan of a workflow is the total completion time to process all tasks in the workflow. Accordingly, the average makespan of all workflows is given by Equation (10).

$$AMS = \frac{max\{FT_1, \ldots, FT_N\} - ST_1}{Num} \tag{10}$$

Based on the above definitions, we can formulate the workflow scheduling problem as:

$$\min fitness = w \times \overline{ATC} + (1 - w) \times \overline{AMS} \tag{11}$$

where w is the weight vector that balances the importance in between $\overline{ATC}$ and $\overline{AMS}$. That is, the objective of cloud workflow scheduling is to minimize both $\overline{ATC}$ and $\overline{AMS}$. where

$$\overline{ATC} = \frac{ATC_{max} - ATC}{ATC_{max} - ATC_{min}} \tag{12}$$

and

$$\overline{AMS} = \frac{AMS_{max} - AMS}{AMS_{max} - AMS_{min}} \tag{13}$$

ATC_{max} and ATC_{min} are iteratively updated in each generation, they are the minimal and maximum value found so far in every generation.

IV. Flexible Scheduling using Genetic Programming Approach

In this section, we describe the representation of our heuristics generated by our FSGP. The function set, the terminal set, and operators which we design for our FSGP such as $crossover$, $mutation$ and $reproduction$.

A. Representation

In our approach, we represent a scheduling heuristic as a GP tree. The commonly considered properties of workflow and virtual machines are represented as terminal nodes in GP tree, while intermediate nodes are arithmetic functions which are described in the next section. Some example trees are presented in Fig. 2

B. Function Set and Terminal Set

TABLE II
The Terminal and Function Set for the Proposed FSGP Approach

Terminal name	**Definition**
TS	The size of task t_i
V	The speed of virtual machine m_k
UC	Unit cost of virtual machine m_k
ET	Execution time of task t_i
WT	Waiting time of task t_i
RFT	Relative finish time of task t_i
$Const$	Constant value
Function name	**Definition**
Add, Substract, Multiply	Basic arithmetic operations (+, -, $\times$)
Protected Division	Protected division. Return 1 if the denominator is 0.(/)

As shown in Table II, the function set consists of the operators, such as add, substract, multiply and protected division. The protected division will return one if divided by 0.

The properties of workflow and virtual machines are used as terminals which have shown in Table. II. All the terminals are representing the properties of workflows and virtual machines, which have an impact on either the total cost or makespan.

C. Crossover, Mutation, Reproduction and Fitness

Crossover and mutation are the most commonly used techniques in GP [3], [7]. $Crossover$ is often called recombination and is a genetic operator used to recombine the genetic information of two parents to generate a new offspring. As shown in Fig. 2, the left branch of Parent A recombined with the right branch of Parent B and become A'. As the same, the right branch of A recombined with the left branch of B and became A'.

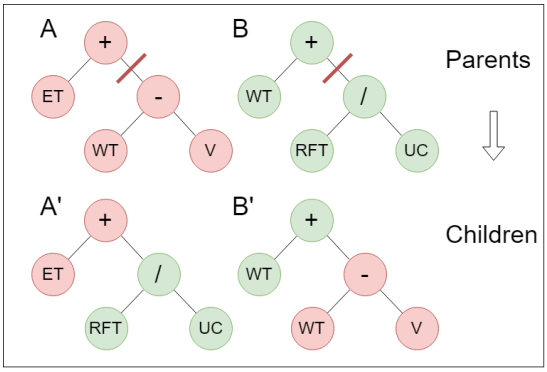

Fig. 2. An Example of Crossover Process

$Mutation$ is a genetic operator used to maintain genetic diversity from one generation to the next generation by changing the subtree of a tree. As demonstrated in Fig. 3 the origin subtree on the right-hand of the tree is replaced by a newly created subtree after mutation process. Mutation is used to avoid local minima by preventing the instances in the population from becoming too similar to each other, which may cause the program to stop evolving, leading to poor performance.

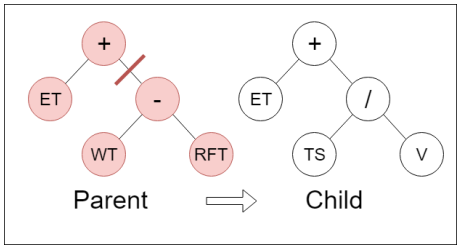

Fig. 3. An Example of Mutation Process

$Reproduction$ selects several individuals in the current generation and will pass the best individuals to the next generation as offspring.

We aim to generate heuristic that can be used to schedule workflows so that ATC and AMS are minimized. To evaluate the fitness of our heuristic we use Equation (11) in Section III. That is, the fitness of a heuristic is computed by the normalized average total cost (i.e. $\overline{ATC}$) and normalized average makespan (i.e. $\overline{AMS}$) of all the workflows scheduled by the heuristic generated by FSGP.

As illustrated in Fig. 4, the FSGP scheduling process starts with a training set of unscheduled tasks and standby virtual machines.

D. Outline of our FSGP approach

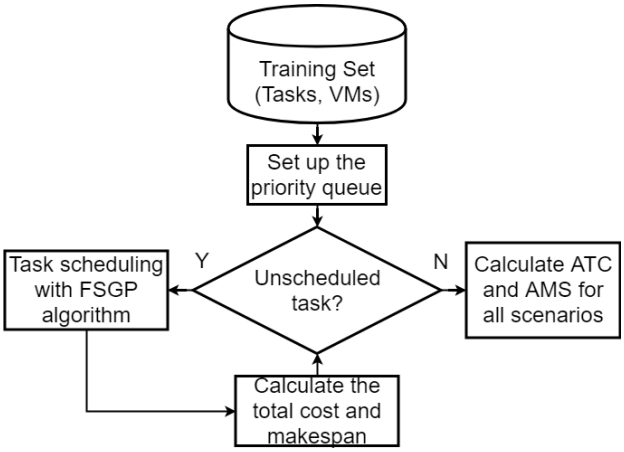

Fig. 4. Constructing Scheduling Process with FSGP Approach

Before scheduling, all workflow tasks are added to a priority queue based on their number of immediate child tasks. A task with more child tasks gets higher priority. Due to our preference of executing such tasks first since workflow execution can be easily halted without completing these tasks as early as possible.

Our proposed FSGP approach employs GP to evolve scheduling heuristics. As described in ALGORITHM 1, the evaluation of each heuristic depends on how well the specific heuristic performs at scheduling workflows using cloud resources. In other words, every evolved heuristic in the form of a GP tree is used to scheduling all the tasks in a given set of workflows. Once all the tasks in all workflows have been scheduled, the ATC and AMS can then be calculated using Equation (9) and (10.) In this paper we initialize ATC and AMS with the maximum value.

The input training set of this algorithm contains a list of tasks and a list of available virtual machines, the total number of generations in each run, and the size of population. The output of our proposed algorithm is a single heuristic rule which will be used to determine the priority of each task in a testing scenario and provide a near-optimal scheduling solution.

V. EXPERIMENTS

In this section we present experimental evaluation of our proposed approach in comparisons with some existing algorithms. We will first describe the dataset we constructed and used in the experiments. We will then describe the settings and comparisons between competing algorithms and our FSGP approach. [1] In the following, we will describe the dataset used in the experiments and competing algorithms. The results and analysis will be presented subsequently.

[1]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

Algorithm 1: FSGP Algorithm

Input: T_{train}, number of generation G, population evolve process $evolve$
Output: Single Heuristic Rule

```
initialize population pop;
while G has not yet passed do
    for each individual(i) in pop do
        for each training sample T in T_train do
            /* solve T using i in pop as a heuristic */;
            while tasks in queue is not empty do
                if Virtual Machine is ready then
                    select which task to serve using
                     heuristics (h)
                end
            end
            get the total cost of T using h get the total
             makespan of T using h
        end
        Evaluate individual i using Eq. (11)
    end
    evolve(pop)
end
Return best individual
```

A. Dataset

To evaluate the performance of our proposed FSGP approach, we conducted the experimental evaluation with the diverse collection of heterogeneous workflows and virtual machines as shown in Table III. We design a test set with 243 workflow scenarios. In each scenario, heterogeneous workflows are created by including five benchmark basic workflow patterns. The dataset includes three sizes of each workflow type in our experiments, which are small (approx. 30 tasks), medium (approx. 50 tasks) and large (approx. 100 tasks). In term of VMs to account for the resource heterogeneity, 15 types of VMs (see Table IV) described in Amazon EC2[2] are considered. The EC2 Compute Unit (ECU) provides the relative measure of the processing power, and it is equivalent to CPU capacity of a 1.0-1.2 GHz Opteron or Xeon processor.

TABLE III
SIZE OF TASKS AND VIRTUAL MACHINES IN EACH EXPERIMENTAL SCENARIO

Scenarios	CyberShake	Epigenomics	Inspiral	Montage	SIGHT	VM
1	small	small	small	small	small	small
2	small	small	small	small	medium	medium
3	small	small	small	small	large	large
4	small	small	small	medium	small	small
5	small	small	small	medium	medium	medium
...	...	...	...	...	...	...
243	large	large	large	large	large	large

Three groups of virtual machine combinations are designed to test the impact of different VMs in the workflow scheduling problem. In the first group, we only use one instance of each

[2]https://aws.amazon.com/ec2/instance-types/

TABLE IV
DEFINITION OF VMS AS DESCRIBED BY AMAZON EC2

Type	vCPU	ECU	Memory (GiB)	Cost ($/hr)
m4.large	2	6.5	8	0.1
m4.xlarge	4	13	16	0.2
m4.2xlarge	8	26	32	0.4
m4.4xlarge	16	53.5	64	0.8
m4.10xlarge	40	124.5	160	2
c4.xlarge	4	16	7.5	0.199
c4.2xlarge	8	31	15	0.398
c4.4xlarge	16	62	30	0.796
c4.8xlarge	36	132	60	1.591
g3.4xlarge	16	47	122	1.14
g3.8xlarge	32	94	244	2.28
g3.16xlarge	64	188	488	4.56
r3.4xlarge	16	52	122	1.33
r3.8xlarge	32	104	244	2.66
r4.large	2	7	15.25	0.133

VM type, in which case 15 VMs (i.e. small) are always occupied with new tasks and the level of parallel task processing is low. For the second group, we provide 10 VMs for each type, hence a total of 150 VM (i.e. medium) instances. During workflow scheduling, tasks can be relatively evenly distributed to different resources for parallel processing to reduce the makespans. In the third group, there are 25 VMs available for each type, making a total of 375 VM (i.e. large) instances. In this group, we can support the highest possible level of parallelism in the cloud.

The 243 experimental scenarios (i.e. each scenario has a unique ID from 1 to 243), are further divided into a training set and a test set. Each of them has approximately the same number of experimental scenarios. In particular, scenarios with odd IDs are selected as the training set, while scenarios with even IDs are treated as the test set. The test dataset is used to evaluate the proposed FSGP approach as well as other competing algorithms.

B. Competing Algorithms

We have examined several typically used scheduling heuristics as the competing algorithms. A brief summary of each is given below [6], [8], [9], [25], [26].

- PSO, as known as Particle Swarm Optimization which will produce a near optimal solution; its goal is to minimize the total makespan and total cost.
- HEFT, as known as Heterogeneous Earliest Finish Time which always selects the fastest available resource; its goal is to minimize the total execution time.
- CHP, a greedy algorithm that always selects the cheapest available resource; its goal is to minimize the total computation cost.
- RR, as known as Round Robin algorithm which selects each resource in a circular order.
- WRR, also known as weighted Round Robin algorithm which selects each resource in circular order and skips the one with maximum Relative Finish Time (RFT).
- Random, a random algorithm that randomly picks up an available resource.

C. Experimental Settings

In our FSGP, we set the population size as 500, and the number of generations as 100. The crossover, mutation and reproduction rate are 85%, 10% and 5%, respectively [19]. The maximal depth of a tree is set to 17 and the tournament selection method with size 7 is used to encourage the survival of effective heuristics in the population. For PSO we set the population size as 20 and the number of generation as 200 which is used in [6] and achieved a good performance. We set C_1 = 0.1 and C_2 = 0.2 that are slightly smaller than the settings in [6], to have a bigger searching space. PSO converges before 200 generation, therefore we set the maximum generation to 200.

D. Comparison of Fitness, ATC and AMS

In Fig.5, we compare the overall fitness with all the competing algorithms across all the scenarios. The result shows our FSGP has consistently better performance, comparing with other competing algorithms.

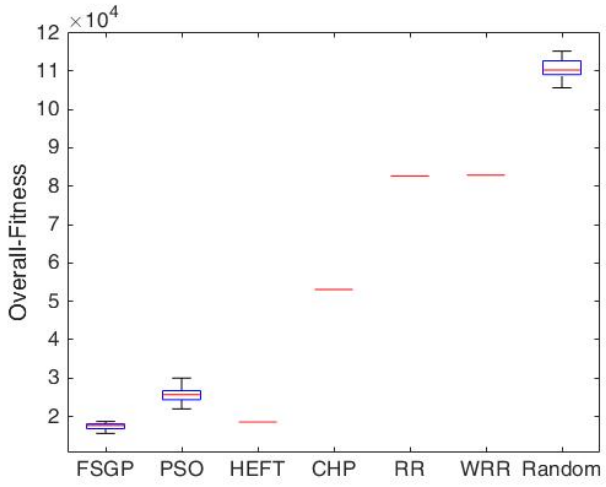

Fig. 5. Boxplot of overall-finess on competing algorithms

Table V shows the mean of total cost and makespan over 30 independent runs of FSGP, PSO and HEFT algorithms. The smaller the value, the better performance the algorithm is.

In Fig. 6 and 7, we compare all competing algorithms across all test scenarios. The box-and-whisker diagrams show the minimum and maximum values of different heuristics. The red line in the box indicates the median value and the outliers are plotted as '+' symbols for each heuristic. A red line without surrounding box indicates the value is unchanged across 30 runs.

In Fig. 6 and Fig. 7, we can observe that our FSGP achieved consistently the lowest makespan across different scenarios. FSGP also achieved the secondary best overall performance on total cost across different scenarios while PSO produced the lowest cost. On the other hand, the Random algorithm performs worst across the diverse workflow scenarios.

In summary our FSGP approach performed consistently better than CHP, RR, WRR and Random algorithms across

TABLE V
RESULTS FOR THE FSGP AND HEFT ALGORITHMS
(FOR BOTH ATC AND AMS THE SMALLER THE VALUE THE BETTER)

FSGP		PSO		HEFT	
ATC($)	AMS(hrs)	ATC($)	AMS(hrs)	ATC($)	AMS(hrs)
$138,061 \pm 8,285$	$4,066 \pm 55$	$72,158 \pm 1,421$	$2,0165 \pm 3506$	$148,159$	$4,140$

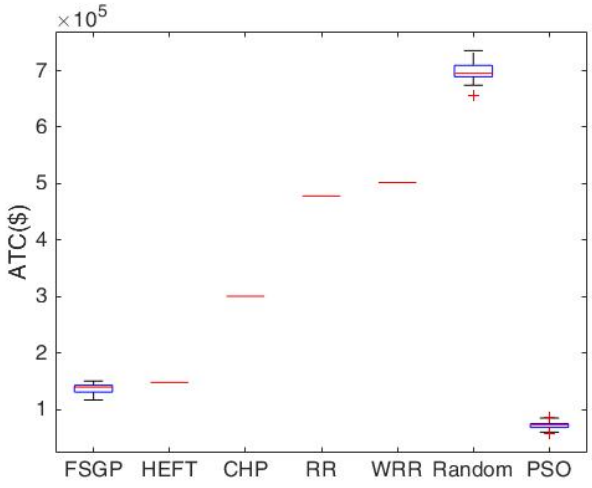

Fig. 6. Boxplot of Average total cost on competing algorithms

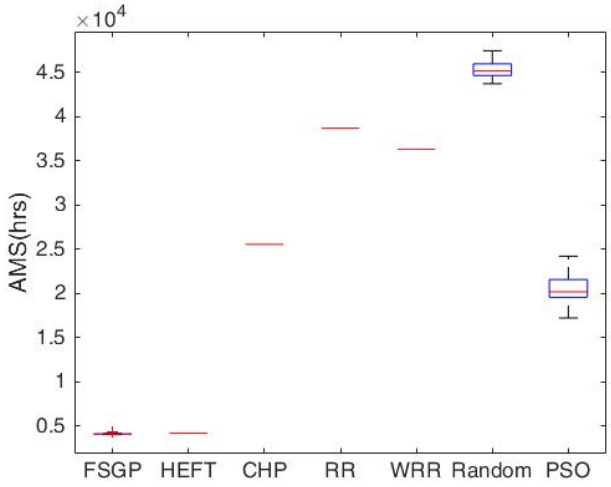

Fig. 7. Boxplot of Average Makespan on competing algorithms

all experimental scenarios. The median value of our FSGP is 1.1% better than HEFT, and all the values in the upper and lower quartile are less than the corresponding quantities obtained by HEFT. Hence FSGP clearly outperformed HEFT. Comparing with PSO, FSGP had a much better performance on total makespan, but it produced a slightly higher total cost. However, FSGP achieved consistently better fitness values than all the competing algorithms and across all experimental scenarios in term of fitness. In particular, FSGP represents an average of 45% of CHP, 28% of RR, 11% of WRR and 19% of Random in the total operation cost, and an 84%, 91%, 89% and 89% reduction of CHP, RR, WRR and Random in makespan, respectively.

We then perform Wilcoxon test to compare the results. For the average total cost, the result for comparison between PSO and FSGP is $p\text{-}value = 3.016 \times e^{-}11$, the result for HEFT and FSGP is $p\text{-}value = 1.3327 \times e^{-}8$. For the average makespan, the result for comparison between PSO and FSGP is $p\text{-}value = 2.9729 \times e^{-}11$, the result for HEFT and FSGP is $p\text{-}value = 7.3572 \times e^{-}10$. All the h values obtain from Wilcoxon are equal to 0, which indicates a failure to reject the null hypothesis at the 5% significance level. Therefore our FSGP is better than PSO and HEFT across all scenarios.

In addition, in terms of workflow scheduling, HEFT prefers to choose VMs with higher processor speed even though the cost is relatively high. In this case, some low-cost VMs might always remain idle without contributing any resources to workflow processing, which causes another problem of load unbalancing in the cloud. While using FSGP, the scheduling heuristics consider jointly multiple scheduling related properties made available through terminal nodes, not simply the processor speed. Hence tasks are more likely to be distributed evenly over VMs.

E. Further Analysis

To investigate which features/terminals affect the performance of the heuristics generated by FSGP, we collected the best heuristic rules generated in each run. There are some useless feature combinations like "$(WT - WT)$" and "(ET/ET)", which will not have any impact and will be removed from the trees.. After removing these useless combinations, we counted the frequency of terminal appearances in the heuristic rules, which is shown in Table VI.

TABLE VI
FREQUENCY OF EACH TERMINAL

Terminal	TS	V	UC	ET	WT	RFT
Frequency	45	86	30	36	47	127

We can observe from Table VI that the most frequently occurred terminals are relative finish time (RFT) and the speed of virtual machine (V). In the following, we show one of the heuristic rules generated by our FSGP to analyze the relationship between the heuristic rules and terminals.

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

In this paper, tasks are scheduled according to the value calculated by the heuristic; the smaller the value is, the earlier

it will scheduled. From $Heuristic_1$ above, we observe that smaller RFT and greater V can lead to a smaller value calculated from this heuristic. Also, TS and WT are the third and fourth most frequently occurred terminals. Hence they also played an essential rule in many heuristic rules generated by FSGP. Therefore, the heuristic rules containing WT has good performance. Comparing with existing heuristics, heuristics generated by our approach include more features that have an impact on cost and makespan.

The above observation is confirmed with other heuristics generated by our approach, for example

$Heuristic_2$ = $((RFT \times RFT) + RFT)$ + $((RFT$ - $V)$ + $((((WT$ + $(V$ / $((RFT - V) \times RFT)))$ + $RFT) \times RFT)$ $\times$ $(RFT - V)))$

To help us find out the most impacted features we can simplify it as the following:

$Heuristic_2 = RFT^2 + (1 + WT) \times C + RFT \times C$ with $C = RFT - V$.

$Heuristic_2$ also confirm that smaller RFT and greater V can leads to higher priority of a task to be scheduled. Further, the smaller $(1+WT$) is, the smaller value calculated from the heuristic, the higher priority a tasks to be assigned. That is, the tasks with shorter waiting time should be processed earlier. Again, our heuristic considers more features than existing heuristics.

VI. Conclusions and Future Work

This paper presented a genetic programming based approach to automatically generate heuristics for scheduling heterogeneous workflows in a cloud. The novelty of our paper is that it addresses two factors simultaneously: the makespan and total cost, with the consideration of tasks dependencies and variety types of virtual machines. The experimental results show our FSGP's flexibility to tackle workflow scheduling problem in the complex cloud environment, and our FSGP approach outperforms other competitive algorithms across the various workflow patterns and varied types of resources in terms of total cost and marksman. In the future, our proposed algorithm can also be extended to solve dynamic workflow scheduling problems, which included adding/removing tasks from the workflow while the workflow is being processed. Because our proposed algorithm aims to provide a heuristic rule to calculate the priority of each task when dealing with a dynamic environment, it can generate a scheduling solution very fast. More importantly, it can also keep evolving along the changes of workflow occurring.

References

[1] A. Mahmood and S. A. Khan, "Hard real-time task scheduling in cloud computing using an adaptive genetic algorithm," *Computers*, vol. 6, no. 2, p. 15, 2017.

[2] J.-Z. Luo, J.-H. Jin, A.-b. Song, and F. Dong, "Cloud computing: architecture and key technologies," *Journal of China Institute of Communications*, vol. 32, no. 7, pp. 3–21, 2011.

[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] D. de Oliveira, E. Ogasawara, F. Baio, and M. Mattoso, "Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows," *2010 IEEE 3rd International Conference on Cloud Computing*, no. 8, 2010.

[5] N. Chopra and S. Singh, "Heft based workflow scheduling algorithm for cost optimization within deadline in hybrid clouds," in *Computing, Communications and Networking Technologies (ICCCNT), 2013 Fourth International Conference on*. IEEE, 2013, pp. 1–6.

[6] V. G. Shubham, Rishabh Gupta and P. K. Jana, "An effective multi-objective workflow scheduling in cloud computing: A PSO based approach," *2016 Ninth International Conference on Contemporary Computing (IC3)*, no. 6, 2016.

[7] 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 *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 3000–3007.

[8] T. Estrada, M. Wyatt, and M. Taufer, "A genetic programming approach to design resource allocation policies for heterogeneous workflows in the cloud," in *Parallel and Distributed Systems (ICPADS), 2015 IEEE 21st International Conference on*. IEEE, 2015, pp. 372–379.

[9] L.-j. Jin, F. Casati, M. Sayal, and M.-C. Shan, "Load balancing in distributed workflow management system," in *Proceedings of the 2001 ACM symposium on Applied computing*. ACM, 2001, pp. 522–530.

[10] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 5, 2017.

[11] H. Topcuoglu, S. Hariri, and M. 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.

[12] R. Garg and A. K. Singh, "Multi-objective workflow grid scheduling using\varepsilon-fuzzy dominance sort based discrete particle swarm optimization," *The Journal of Supercomputing*, vol. 68, no. 2, pp. 709–732, 2014.

[13] Y. Dai, Y. Lou, and X. Lu, "A task scheduling algorithm based on genetic algorithm and ant colony optimization algorithm with multi-qos constraints in cloud computing," in *Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2015 7th International Conference on*, vol. 2. IEEE, 2015, pp. 428–431.

[14] K. Padmaveni and D. J. Aravindhar, "Hybrid memetic and particle swarm optimization for multi objective scientific workflows in cloud," in *Cloud Computing in Emerging Markets (CCEM), 2016 IEEE International Conference on*. IEEE, 2016, pp. 66–72.

[15] J. Meena, M. Kumar, and M. Vardhan, "Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint," *IEEE Access*, vol. 4, pp. 5065–5082, 2016.

[16] J. Sahni and D. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," *IEEE Transactions on Cloud Computing*, 2015.

[17] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, and J. Wen, "Deadline-constrained cost optimization approaches for workflow scheduling in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3401–3412, 2017.

[18] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multi-objective workflow scheduling in cloud," *IEEE Transactions on parallel and distributed Systems*, vol. 27, no. 5, pp. 1344–1357, 2016.

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

[20] 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.

[21] R. Hunt, M. Johnston, and M. Zhang, "Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 618–625.

[22] Y. Mei, S. Nguyen, and M. Zhang, "Evolving time-invariant dispatching rules in job shop scheduling with genetic programming," in *European Conference on Genetic Programming*. Springer, 2017, pp. 147–163.

[23] D. Karunakaran, Y. Mei, G. Chen, and M. Zhang, "Dynamic job shop scheduling under uncertainty using genetic programming," in *Intelligent and Evolutionary Systems*. Springer, 2017, pp. 195–210.

[24] Y. Mei, X. Li, F. Salim, and X. Yao, "Heuristic evolution with genetic programming for traveling thief problem," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 2015, pp. 2753–2760.

[25] M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of scientific workflows in the askalon grid environment," *ACM SIGMOD Record*, vol. 34, no. 3, pp. 56–62, 2005.

[26] D. C. Devi and V. R. Uthariaraj, "Load balancing in cloud computing environment using improved weighted round robin algorithm for nonpreemptive dependent tasks," *The scientific world journal*, vol. 2016, 2016.