A Cooperative Coevolution Hyper-Heuristic Framework for Workflow Scheduling Problem

Qin-zhe Xiao, Jinghui Zhong, Liang Feng, Linbo Luo, and Jianming Lv

Abstract—Workflow scheduling problem (WSP) is a well-known combinatorial optimization problem, which is defined to assign a series of interconnected tasks to the available resources to meet user defined Quality of Service (QoS). The guided random search methods and heuristic based methods are two most common methods for solving WSP. However, these methods either require expensive computational cost or heavily rely on human's empirical knowledge, which makes them inconvenient for practical applications. Keeping this in mind, this paper proposes a cooperative coevolution hyper-heuristic framework to solve WSP with an objective of minimizing the completed time of workflow. In particular, in the proposed framework, two heuristic rules, namely, the task selection rule (TSR) and the resource selection rule (RSR), are learned automatically by a cooperative coevolution genetic programming (CCGP) algorithm. The TSR is used to select a ready task for scheduling, while the RSR is used to allocate resources to perform the selected task. To improve the search efficiency, a set of low-level heuristics are defined and used as building blocks to construct the TSR and RSR. Further, to validate the effectiveness of the proposed framework, randomly generated workflow instances and four real-world workflows are used as test cases in the experimental study. Compared with several state-of-the-art methods, e.g., the Heterogeneous Earliest Finish Time (HEFT) and the Predict Earliest Finish Time (PEFT), the high-level heuristics found by our proposed framework demonstrate superior performance on all the test cases in terms of several metrics including the schedule length ratio, speedup and efficiency.

Index Terms—Workflow scheduling, genetic programming, hyper-heuristic, cooperative coevolution, DAG scheduling

1 Introduction

In real world, many practical applications in distributed computing platform could be modeled as workflows, such as the Montage workflow [1] and the Epigenomic workflow [2]. The workflow usually consists of a number of interdependent tasks that can be represented by a Directed Acyclic Graph (DAG). In this DAG, the nodes represent tasks, while the edges denote dependencies between the tasks. The workflow scheduling problem (WSP) requires properly assigning a set of resources to the interconnected tasks of a workflow to satisfy user defined QoS, such as minimizing the completed time of the workflow and minimizing the rental cost in cloud.

Over the past decades, extensive research efforts [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20] have been conducted for solving WSP. Among others, the guided random search methods and the heuristic-based methods are two categories of the most common methods for solving WSP. The guided random search methods are usually based on population-based evolutionary algorithms (EAs) [3], [4], [5] to optimize solutions for the WSP. In these algorithms, each individual in the population represents a candidate solution and the whole population is evolved iteratively by nature-inspired operators such as mutation, crossover, and selection. These algorithms have demonstrated strong search capabilities in locating the global optimal or near-global optimal solutions. However, they often require expensive computational cost due to the iterative search mechanism. Furthermore, if the given workflow varies, they should be re-executed to search for suitable solutions for the new WSP, which makes them inconvenient in practice.

On the other hand, the heuristic-based methods have also been proposed to solve the WSP. In contrast to the guided random search methods, the heuristic-based methods schedule workflow via heuristic rules, which can be further classified into three major categories [1]: 1) list-based heuristics, 2) clustering-based heuristics, and 3) duplication-based heuristics. Among them, the most well-known and popular methods are the list-based heuristics [8], [9], [10], [11], [12], [13], [14], [15], [16], which commonly comprise two phases, i.e., the task prioritizing phase and the resource selection phase. In the task prioritizing phase, each task possesses a priority and the task with higher priority will be scheduled first. In the resource selection phase, the selected task will be scheduled on the resource with highest priority. Further, in the clustering heuristics [17], [18], the key idea is to divide tasks into a set of clusters and then map the clusters to resources. Lastly, in the duplication-based heuristics [19], [20], the predecessors of a task may be copied to the resource on which the task will be executed. In this manner, the predecessors do not need to transfer data to the task and the communication time between them thus can be saved. Generally, existing heuristic-based methods for the WSP are manually

- Q. Xiao, J. Zhong, and J. Lv, are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China. E-mail: xiaoqinzhe@qq.com, {jinghuizhong, jmlv}@scut.edu.cn.
- L. Feng is with the College of Computer Science, Chongqing University, Chongqing, China. E-mail: liangf@cqu.edu.cn.
- L. Luo is with the School of Cyber Engineering, Xidian University, Xi'an, China. E-mail: lbluo@xidian.edu.cn.

Manuscript received 20 May 2018; revised 8 May 2019; accepted 15 June 2019. Date of publication 24 June 2019; date of current version 4 Feb. 2022. (Corresponding author: J. Zhong.)
Digital Object Identifier no. 10.1109/TSC.2019.2923912

designed in an ad-hoc manner. The design process is time-consuming and heavily dependent on the designers' empirical knowledge.

Keeping the above in mind, this paper proposes a cooperative coevolution hyper-heuristic framework to solve the WSP. The proposed framework is capable of automatically learning high-level heuristics to effectively schedule workflows without expert intervention. In this paper, the hyper-heuristic is a methodology that automatically produces high-level heuristics constructed by several simple low-level heuristics, and thus the high-level heuristics are more powerful and effective. To learn the high-level heuristics automatically, the Genetic Programming (GP) algorithm [21] is utilized in this paper. GP is a kind of powerful hyper-heuristic algorithm for automatically evolving high-level heuristics to solve problems, and it has been successfully applied to automated design of scheduling heuristic [25], [26], such as the job shop scheduling problems [27], [28] and air traffic control problems [29]. However, as far as we know, little work has been reported in the literature on using GP for solving WSP. This paper presents an early attempt to fill this gap. Furthermore, for WSP, the high-level heuristics are comprised of two coadapted sub-heuristic, i.e., the task selection rule (TSR) that used to select a ready task, and the resource selection rule (RSR) that used to select a resource to execute the ready task. The Cooperative Coeveolution (CC) architecture [22] is a mechanism for effectively evolving coadapted subcomponents, which are the TSR and RSR in this paper. Therefore, the CC architecture is integrated into GP, forming a Cooperative Coevolution Genetic Programming (CCGP) algorithm, to efficiently learn high-quality coadapted sub-heuristics.

In particular, the proposed framework is composed of a Heuristic-based Workflow Scheduling (HBWS) algorithm and the CCGP algorithm mentioned above. The HBWS algorithm is designed to schedule workflow by using given heuristics, while the CCGP algorithm is used to learn effective high-level heuristics. To verify the effectiveness of the proposed framework, comprehensive empirical study has been conducted. In the experiment, the best high-level heuristics learned by the proposed framework is compared with the state-of-the-art methods (e.g., the HEFT [8], Lookahead [9], PEFT [10], and the Particle Swarm Optimization (PSO) [5]) on various randomly generated workflows and four real-world workflows. The experimental results have demonstrated that the learned heuristics can always achieve superior performance than the other methods in terms of metrics including the schedule length ratio, speedup and efficiency.

The rest of the paper is organized as follows: Section 2 presents the related work about heuristic-based and guided random search algorithms for the WSP. Section 3 presents the background and the problem definition of the WSP. The proposed framework for the WSP is given in Section 4, and the framework implementation is described in Section 5. The experiments and results are provided and discussed in Section 6. Finally, the conclusion is given in Section 7.

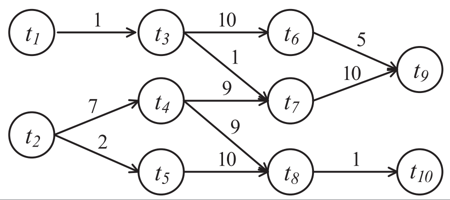
2 Related Work

In this section, we present a brief review of the related methods in the literature for solving the WSP. In particular, as aforementioned, two categories of algorithms, guided random search algorithms and heuristic-based algorithms, are discussed.

First of all, the guided random search algorithms search the solution space iteratively and use the historical information to guide the search. The population-based algorithms such as Generic Algorithm (GA) [3], [4], PSO [5] and Ant Colony Optimization [6], [7] have also been studied for solving WSP. Particularly, in [4], two GA variants, i.e., the Critical Path Genetic Algorithm and the Task Duplication Genetic Algorithm, have been proposed to utilize problem-specific heuristic principles to improve the search performance for WSP. Chen et al. [7] proposed a framework based on ACO to schedule workflows in grid with various QoS parameters, such as deadline constrain, cost constrain and makespan optimization. In these algorithms, solutions of WSP are encoded as chromosomes and nature-inspired operators such as mutation and crossover are utilized to evolve the population of chromosomes iteratively. The algorithms terminate when predefined stopping criteria are met, and finally the best individual with best objective value in the population will be decoded as the final solution. These algorithms are generally capable of generating better solutions than the heuristic-based algorithms, but they require much more computation time and they contain a number of control parameters which need to be set properly.

On the other hand, heuristic-based algorithms can be classified into three major categories: list-based heuristics, clustering-based heuristics, and duplication-based heuristics. The HEFT algorithm [8] is one of the most well-known list-based heuristic methods for WSP in heterogeneous environment. The algorithm has two major phases, i.e., the task prioritizing phase and the resource selection phase. In the task prioritizing phase, each task is assigned with an upward rank and tasks are scheduled according to their ranks. In the resource selection phase, each task will be performed on the resource that minimizing its earliest finish time (EFT). The authors also proposed the Critical-Path-on-a-Processor (CPOP) algorithm [8] where the rank of each task is equal to the summation of the upward rank and downward rank. The second phase is the same as HEFT except that the task on the critical path will be scheduled on the critical-path resource. In [11], a hybrid heuristic was proposed for DAG scheduling in heterogeneous system. The algorithm consists of three phases: ranking, group creation, and scheduling independent tasks within each group. In the ranking phase, each task is assigned with an upward rank. In the second phase, tasks are divided into different groups and the tasks in the same group are independent to each other. In the last phase, the Balanced Minimum Completion Time heuristic [11] is used to schedule independent tasks within each group. Similarly, other list-based heuristic algorithms such as the Lookahead [9], PEFT [10], PETS [12], and MSL [13] are usually comprised of two major phases. Generally, the PEFT performs better than the Lookahead, HEFT, CPOP and PETS algorithms on randomly generated graphs and several real-world applications. More recently, the Improved PEFT [14] and Heterogeneous Scheduling with Improved Task Priority [16] algorithms have been proposed to solve the WSP.

In contrast to the list-based algorithms, the clustering-based algorithms divide tasks into a set of clusters and tasks in a common cluster are scheduled to the same resource.



task	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
p_1	40	22	32	49	21	12	14	11	25	9
p_2	8	25	32	56	31	21	14	18	44	31
p_3	8	44	24	12	10	21	14	37	47	50

Fig. 1. A typical workflow DAG and the tasks execution time on three processors.

The heuristics are intended to cluster the tasks which spend a lot of time on communicating with each other. Then the tasks of a cluster will be scheduled on the same resource so as to reduce the communication overhead. However, these algorithms generally have higher time complexity compared to the list-based algorithms. Typical examples of the clustering algorithms include the Dominant Sequence Clustering [17] and the Clustering for Minimizing the Worst Scheduling Length algorithms [18].

Further, the task duplication algorithms, which are the third category of heuristic-based algorithms, reduce communication time by duplicating the predecessors of a task into the resource on which the task executes. However, these algorithms usually have a heavier resource workload than the other two types of heuristics due to the extra task duplication operation. Therefore, algorithms of this category are more suitable for cases where the communication time and the idle time of processors are extremely large. Typical examples of this category include the Task Duplication-based Scheduling algorithm [19] and the Task duplication-based scheduling Algorithm for Network of Heterogeneous systems [20].

However, existing heuristic-based methods are designed manually by experts. The rule designing procedure includes many trial-and-error steps, which is quite time-consuming and requires luxuriant domain knowledge.

GP has been successfully applied for scheduling problems, such as the job shop scheduling [27], [28]. The job shop scheduling is assigning jobs to resources for satisfying user QoS (e.g., minimizing complete time or mean tardiness), and the jobs are independent. However, in WSP, the scheduled targets are workflows and the tasks in a workflow have inter-task data dependencies. Therefore, GP for JSP cannot be applied directly to WSP. It is necessary to design task-specific low-level heuristics considering tasks' dependencies. Besides, the GP should be extended accordingly to efficiently learn high-level heuristics for WSP. In [32], the authors proposed a preliminary GP-based approach to find policies that can select resource to execute tasks in a cloud environment. However, their method only focused on selecting resource for a ready task, while the order for executing tasks was not considered.

To overcome the above drawbacks of the existing methods, we propose a hyper-heuristic framework for WSP in this paper. The major difference between the existing heuristic-based methods and the work reported in this paper is that the heuristics in our method are learned automatically rather than manually designed by experts. Unlike the guided random search methods, the heuristics offered by our method can be applied to different cases without re-executing the algorithm, which is more flexible and convenient for practical applications.

3 Problem Definition

A workflow can be represented by a DAG $G = (\boldsymbol{V}, \boldsymbol{E})$, where $\boldsymbol{V}$ is the set of v nodes and $\boldsymbol{E}$ is the set of e edges connecting nodes. A node $v_i \in \boldsymbol{V}$ represents a task t_i, while an edge $e(i,j) \in \boldsymbol{E}$ represents precedence dependency between task t_i and task t_j, i.e., task t_j can be executed once task t_i was finished. The weight of the edge represents the transfer time from task t_i to task t_j. The underlying resource computing system or infrastructure $\boldsymbol{P}$ for scheduling workflows is a set of heterogeneous, distributed and fully-interconnected (e.g., with network) computers such as grids and cloud computing. Fig. 1 illustrates a typical workflow DAG with 10 nodes and 11 edges and the table in Fig. 1 gives the estimated execution time of each task, if the entire task is running on one of the given resources (i.e., three computers p_1 to p_3).

In a DAG, a task without any predecessor is denoted as an entry task t_{entry} and a task without any successor is denoted as an exit task t_{exit}. We assume that task scheduling is non-preemptive, which means that a task will not stop running until it is finished or blocked. The execution time of tasks in all resources is known in advance and stored in an $v \times p$ matrix $\boldsymbol{W}$, where v is the number of tasks and p is the number of resources. Each element of $\boldsymbol{W}$ $(w_{i,j})$ represents the execution time of task t_i running on resource p_j. The average execution time of task t_i is calculated by

$$\overline{w}_i = \left(\sum_{j=1}^{p} w_{i,j}\right)/p. \tag{1}$$

The weight $c_{i,j}$ assigned to edge $e(i,j)$ represents the communication cost, i.e., communication time between task t_i and task t_j. The communication cost $c_{i,j}$ of task t_i (scheduled on resource p_m) and task t_j (scheduled on resource p_n) is calculated by

$$c_{i,j} = \begin{cases} L_m + \frac{data_{i,j}}{min(B_m,B_n)}, & m \neq n \\ 0, & m = n \end{cases}, \tag{2}$$

where L_m is the communication latency time of resource p_m, $data_{i,j}$ is the transfer data size from task t_i to task t_j and min (B_m, B_n) is the function to select the minimum value between the bandwidth B_m of resource p_m and the bandwidth B_n of resource p_n. Note that the communication cost equals to zero when two tasks are scheduled on the same resources.

The average communication cost $\overline{c_{i,j}}$ is calculated by

$$\overline{c_{i,j}} = \overline{L} + \frac{data_{i,j}}{\overline{B}}, \tag{3}$$

where $\overline{L}$ is the average latency time of all the resources and $\overline{B}$ is the average bandwidth of all the resources.

When a task is to be executed on a resource, two attributes, namely, the earliest start time (EST) and the earliest finished time (EFT), are associated to it. The EST of task t_i on resource p_j is denoted as $EST(t_i, p_j)$ which is calculated by

$$EST(t_i,p_j) = max\left\{ \begin{array}{l} avail(p_j), \\ \max_{t_k\in pred(t_i)}\{AFT(t_k)+c_{k,i}\} \end{array} \right\}, \quad (4)$$

where $avail(p_j)$ is the earliest available time of resource p_j and $pred(t_i)$ is the set of immediate predecessor tasks (also named as parent tasks) of task t_i. $AFT(t_k)$ is the actual finish time of task t_k and $AST(t_k)$ is the actual start time of task t_k. The inner max block of the formula returns the ready time of tasks t_i when all predecessor tasks (also named as child tasks) of the task have been finished and the required transfer data have arrived at resource p_j on which task t_i runs. The EST of task t_i on resource p_j is the maximum time among the earliest available time of resource p_j and the ready time of tasks t_i. Note that the ready time of entry tasks is equal to zero.

The EFT of task t_i on resource p_j is denoted as $EFT(t_i,p_j)$, which is calculated by

$$EFT(t_i,p_j) = EST(t_i,p_j) + w_{i,j}. \quad (5)$$

When the task t_i is to be executed on a resource p_j, the $AST(t_i)$ and $AFT(t_i)$ are equal to $EST(t_i,p_j)$ and $EFT(t_i,p_j)$, respectively. After all tasks of a workflow have been finished, the *makespan* of the workflow is set to the maximum time among the actual finish time of all tasks, i.e.,

$$makespan = \max\{AFT(t_{exit})\}. \quad (6)$$

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

$$\begin{array}{l} Minimize\ makespan \\ s.t.\ \ AST(t_i) \geq \max\limits_{t_j\in pred(t_i)} \{AFT(t_j)+c_{j,i}\}, \end{array} \quad (7)$$

where the constrain indicates the dependencies between task t_i and its predecessor tasks. Given a workflow, we can provide a schedule $S=(O,M,makespan)$ where O is a task sequence representing the scheduling order of tasks. And M is a task-resource mapping determining which resource is assigned to a task. The objective of the WSP is to properly assign resources to tasks so as to minimize the *makespan* of the workflow.

4 The CC Hyper-Heuristic Framework

The proposed cooperative coevolution hyper-heuristic framework for WSP includes two algorithms, i.e., the HBWS algorithm for scheduling workflow and the CCGP algorithm for learning the high-level heuristics used in the HBWS. In this section, the HBWS algorithm is introduced at first. Then, the CCGP algorithm is presented.

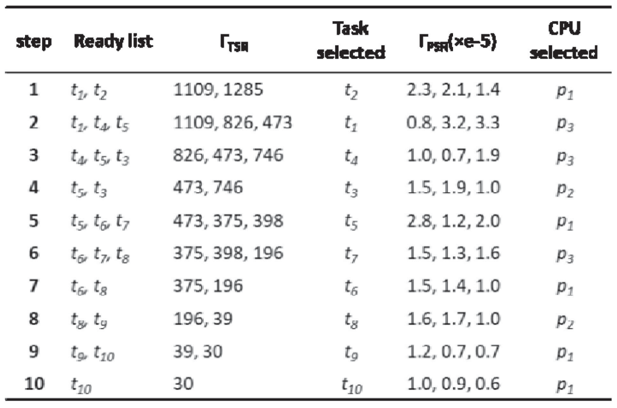
4.1 The HBWS Algorithm

Since the WSP is an NP-Complete problem [8], [33], the exhaust method requires a high time complexity to solve it. The heuristic-based methods are able to find the approximated best solutions with much less computational cost than the exhaust methods. Therefore, the HBWS utilizes heuristics to schedule a workflow. Specifically, the HBWS is designed as a kind of the list-based scheduling methods (e.g., HEFT, PEFT) which contains two phases: the task prioritizing phase and the resource selection phase. Here we use two sub-heuristics named TSR and RSR to represent the heuristics used in the two phases respectively. The TSR is utilized to select a ready task in the first phase, while the RSR is utilized to select a resource to execute the selected task in the second phase. Actually, the TSR and RSR are two priority functions (denoted as Γ_{TSR} and Γ_{RSR}), and the task or resource with the highest priority will be selected. The other list scheduling algorithms can be represented by this formula. For example, in HEFT [8], the Γ_{TSR} is the upward rank of tasks, and the Γ_{RSR} is equal to $EFT(t_i,p_j)$.

The pseudocode of HBWS is described in Algorithm 1. Given a workflow $G=(V,E)$, a resource set $P=\{p_1,p_2,\ldots,p_m\}$ and two heuristics (i.e., a pair of Γ_{TSR} and Γ_{RSR}) as the inputs, the HBWS will schedule all the tasks on the given resources and finally return the solution $S=(O,M,makespan)$. In the HBWS, a waiting queue *WQ* is used to record all the ready tasks. First of all, the waiting queue *WQ* is initialized with all entry tasks of the workflow. Then, the HBWS will repetitively select tasks and resources until all tasks are finished. At each iteration, the task t_i with the highest priority calculated by Γ_{TSR} is selected, and then the selected task t_i is executed on the resource p_j with the highest priority value calculated by Γ_{RSR}. When p_j is assigned to task t_i, the AST and AFT of task t_i are determined. The available time of resource p_j $avail(p_j)$ is determined by using the insert-based policy [8], and the *makespan* is updated. Next, task t_i is appended into task scheduling order O and the task t_i to resource p_j pair is added into the task-to-resource mapping M. After that, the child tasks of the selected task are added to the waiting queue if all of its parent tasks have been scheduled (i.e., parent citation count of task t_c equals to zero). Finally, the obtained schedule S is returned as the output.

Algorithm 1. Heuristic-based Scheduling Algorithm

```
Input: a workflow $G=(V,E)$, a processor set $P=\{p_1,p_2,\ldots,p_m\}$, two priority functions $\Gamma_{\text{TSR}}$ and $\Gamma_{\text{RSR}}$
Output: a schedule $S=(O,M,makespan)$
Procedure:
 1. $makespan = 0$
 2. add all entry tasks of the workflow into the waiting queue WQ
 3. while size of $WQ > 0$ do
 4.    calculate the priorities of tasks in $WQ$ by using $\Gamma_{\text{TSR}}$, and select the task $t_i$ with the first priority
 5.    select the processor $p_j$ with maximum priority calculated by $\Gamma_{\text{RSR}}(t_i,p_j)$ and schedule task $t_i$ on processor $p_j$
 6.    if task $t_i$ is an entry task
 7.       $AST_{t_i} = avail(p_j)$
 8.    else
 9.       $AST(t_i,p_j) = max\left\{avail(p_j), \max_{t_k\in pred(t_i)}\{AFT(t_k)+c_{k,i}\}\right\}$
10.    end if
11.    $AFT(t_i) = AST(t_i) + w_{i,j}$
12.    $makespan = \max(makespan, AFT(t_i))$
13.    update the order $O$ and the mapping $M$
14.    remove the selected task $t_i$ from $WQ$
15.    for each child task $t_c$ of task $t_i$
16.       if $t_c$'s parent tasks are all finished
17.          add task $t_c$ into $WQ$
18.       end if
19.    end for each
20. end while
21. return the obtained schedule $S=(O,M,makespan)$
```



step	Ready list	Γ_{TSR}	Task selected	Γ_{PSR}(×e-5)	CPU selected
1	t_1, t_2	1109, 1285	t_2	2.3, 2.1, 1.4	p_1
2	t_1, t_4, t_5	1109, 826, 473	t_1	0.8, 3.2, 3.3	p_3
3	t_4, t_5, t_3	826, 473, 746	t_4	1.0, 0.7, 1.9	p_3
4	t_5, t_3	473, 746	t_3	1.5, 1.9, 1.0	p_2
5	t_5, t_6, t_7	473, 375, 398	t_5	2.8, 1.2, 2.0	p_1
6	t_6, t_7, t_8	375, 398, 196	t_7	1.5, 1.3, 1.6	p_3
7	t_6, t_8	375, 196	t_6	1.5, 1.4, 1.0	p_1
8	t_8, t_9	196, 39	t_8	1.6, 1.7, 1.0	p_2
9	t_9, t_{10}	39, 30	t_9	1.2, 0.7, 0.7	p_1
10	t_{10}	30	t_{10}	1.0, 0.9, 0.6	p_1

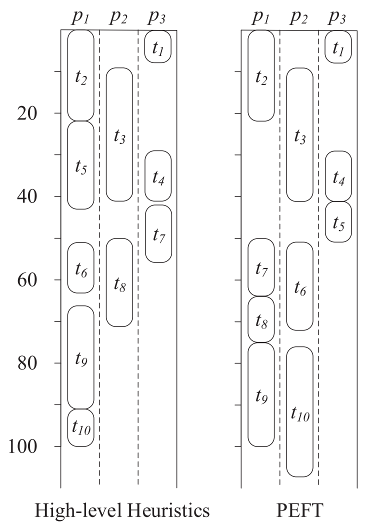


Fig. 2. The left table lists each schedule step in HBWS using the TSR in (25) and the RSR in (26) for performing the workflow example showed in Fig. 1. The two graphs on right present the schedules with the leaned heuristics $(makespan = 100)$ and PEFT $(makespan = 107)$.

For understanding the HBWS algorithm easier, Fig. 2 illustrates a scheduling example of the workflow. The left chart shows each scheduling step by using the HBWS algorithm with the best learned heuristics Γ_{TSR} and Γ_{RSR} in (25) and (26). At each step, the HBWS selects a task and a resource according to the Γ_{TSR} and Γ_{RSR}, respectively, until all tasks are scheduled. For example, at step 3, there are three ready tasks (t_4, t_5, t_3) in waiting queue and their priorities are 826, 473 and 746 calculated by Γ_{TSR}. Then the task t_4 which has the largest priority is selected. For performing the task t_4, all priorities of resources are calculated by Γ_{RSR} and the resource p_3 which has the best priority is chosen. In Fig. 2, the two graphs on right show the whole schedules using the best leaned heuristics and PEFT. It can be observed that the learned heuristics obtain better solution $(makespan = 100)$ than PEFT $(makespan = 107)$. Based on the scheduling algorithm, optimizing the *makespan* is equivalent to optimizing the two heuristics. Thus, the workflow scheduling problem can be converted to the following optimization problem: Given a workflow $G = (\boldsymbol{V}, \boldsymbol{E})$, find the best Γ_{TSR} and Γ_{RSR} that minimize the *makespan* of the workflow, i.e.,

$$\underset{\Gamma_{\mathrm{TSR}},\Gamma_{\mathrm{RSR}}}{\arg\min} \ \ makespan_{\Gamma_{\mathrm{TSR}},\Gamma_{\mathrm{RSR}}}. \tag{8}$$

4.2 The CCGP Algorithm

The CCGP algorithm is proposed to automatically learn effective high-level heuristics (i.e., TSR and RSR) for WSP. The CCGP algorithm is based on GP integrated with cooperative coevolution mechanism. GP is an EA that solves user defined problems by the evolution of computer programs (e.g., functions, policies or heuristics) [21], [23], [24]. As the high-level heuristics for WSP can be represented by mathematic formulas which combine the low-level heuristics together, GP is suitable to evolve the high-level heuristics. Traditionally, each individual of GP can represent a function formed by based functions (e.g., $+$, $/$, and log) and terminals or variables (e.g., x, y, and PI). In GP, each individual is decoded as a tree-structure or string-structure to represent a solution for the defined problem. For solving the WSP, each individual of GP represents high-level heuristics and the goal is to learn high-level heuristics to minimize the makespan of the given workflow. Furthermore, since the high-level heuristics consist of two interacting and co-adapted sub-heuristics (i.e., TSR and RSR), the CC mechanism is considered because it is suitable to solve the problems whose solutions comprise of multiple coadapted subcomponents [22], [30], [31].

To sum up, the general structure of the proposed hyper-heuristic framework is composed of the HBWS algorithm and the CCGP algorithm, as illustrated in Fig. 3. The function set and the terminal set are used as building blocks to construct the high-level heuristics TSR and RSR. Specifically, the terminals are the low-level heuristics and linked by the functions to construct the high-level heuristics. Subsequently, in the CCGP, two sub-populations (denoted as P_{TSR} and P_{RSR}) work cooperatively to evolve the TSR and RSR respectively, i.e., P_{TSR} focuses on evolving P_{TSR} and P_{RSR} focuses on evolving Γ_{RSR}. To evaluate a candidate solution (Γ_{i}) of one sub-population, it needs to be combined with the best individual of the other sub-population (denoted as Γ_{best}) to form a complete solution (the pair of TSR Γ_{TSR} and RSR Γ_{RSR}). Then the average performance of the complete solution on the training set $f(\Gamma_{\mathrm{i}}, \Gamma_{\mathrm{best}})$ (or $f(\Gamma_{\mathrm{best}}, \Gamma_{\mathrm{i}})$) are used as fitness value of Γ_{i}. In this way, the framework evolves the TSR and RSR via an iterative manner, and finally we can obtain the best learned high-level heuristics. Note that the HBWS algorithm is used to not only schedule the workflow, but also evaluate the performance of heuristics in CCGP. Since the performance of heuristics is evaluated on a training set including different workflows, the learned high-level heuristics could be general enough for scheduling different workflows if the training set is large enough.

5 The Framework Implementation

In this section, an implementation of the proposed CCGP is presented. First, the function and terminal sets are defined. Then, the CCGP algorithm based on a recently published GP variant (i.e., Self-learning Genetic Expression Programming (SL-GEP) [23]) is implemented.

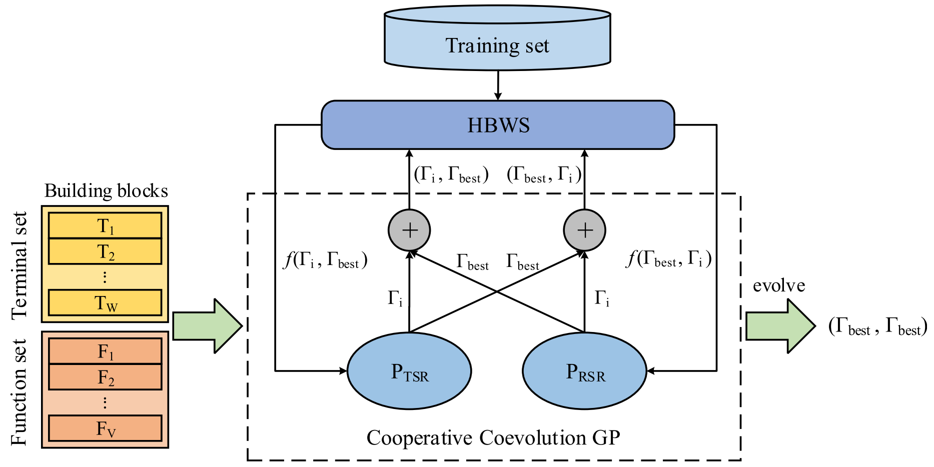


Fig. 3. The general structure of the proposed cooperative coevolution hyper-heuristic framework.

5.1 Function and Terminal Set

The learned high-level heuristics are constructed based on two kinds of elements: terminal and function. Terminals are crucial features that help to make decisions, while functions are used to link up terminals to form the high-level heuristics. The constructed heuristics actually map the terminals into priorities non-linearly. To construct good TSR and RSR, the function set and terminals set should be carefully designed. In this paper, we consider eight commonly used base functions (i.e., {ADD, SUB, MUL, DIV, SQRT, LOG, MIN, MAX}) to construct the high-level heuristics, as described in Table 1. Further, two different terminal sets are defined for the construction of TSR and RSR, respectively. This is because the features that influence the decision of selecting ready task at the scheduling step are different from those of selecting resource for executing the selected task. Specifically, five low-level heuristics are defined as terminals.

a) *The upward rank:* The upward rank $rank_u(t_i)$ of task t_i is the length of the critical path from t_i to the exit task, which is introduced in HEFT [8]. It can be calculated by

$$rank_u(t_i) = \overline{w_i} + \max_{t_j \in succ(t_i)} (\overline{c_{i,j}} + rank_u(t_j)), \tag{9}$$

where $succ(t_i)$ is the set of successors of task t_i. For the exit task, the upward value is $\overline{w_{exit}}$.

b) *The Optimistic Cost Table (OCT):* The OCT for forecasting the optimistic cost of each task on a resource was presented in PEFT [10]. The OCT of task t_i on resource p_k can be calculated by

$$OCT(t_i, p_k) = \max_{t_j \in succ(t_i)} \left\{ \min_{p_w \in P} \{OCT(t_j, p_w) + w_{j,w} + \overline{c_{i,j}}\} \right\}, \tag{10}$$

where $c_{i,j}$ is zero if p_k equals to p_w.

c) *The OCT rank:* In the PEFT, the priority $rank_{oct}(t_i)$ of task t_i is set to be the average OCT, which is defined as

$$rank_{oct}(t_i) = \frac{\sum_{j=1}^{p} OCT(t_i, p_j)}{p}. \tag{11}$$

d) *The Max Remaining Time (MRT):* The $MRT(t_i)$ estimates the maximum cost of paths from t_i children's tasks to the exit task, including the communication cost of t_i and its children's tasks. The $MRT(t_i)$ of task t_i can be calculated by

$$MRT(t_i) = \max_{t_j \in succ(t_i)} \{\overline{w_j} + \overline{c_{i,j}} + MRT(t_j)\}. \tag{12}$$

Note that the MRT of an exit task is equal to zero.

e) *The Real-time Optimistic Time (ROT):* To enhance the optimistic cost table (OCT) of PEFT with real time information, we define a new variable, *ROT*, which is calculated by

$$ROT(t_i, p_j) = \max_{t_k \in succ(t_i)} \left\{ \min_{p_l \in P} \left\{ \begin{array}{c} EST(t_k, p_l) + \\ w_{k,l} + OCT(t_k, p_l) \end{array} \right\} \right\}, \tag{13}$$

where task t_i is presumably scheduled on resource p_j. The $EST(t_k, p_l)$ is calculated according to the actual finish time

TABLE 1
Function Set

Function name	Definition
ADD, SUB, MUL	Binary addition, subtraction and multiplication operators
DIV	Protected division: $\mathrm{DIV}(a,b) = \begin{cases} 0, & \text{if } b < 1e-8 \\ a/b & \text{otherwise} \end{cases}$
SQRT	Protected square root: $\mathrm{SQRT}(a) = \begin{cases} \sqrt{a}, & \text{if } a \geq 0 \\ 0, & \text{otherwise} \end{cases}$
LOG	Protected logarithm: $\mathrm{LOG}(a) = \begin{cases} log a, & \text{if } a > 1e-8 \\ 0, & \text{otherwise} \end{cases}$
MIN, MAX	$\mathrm{MIN}(a,b)$ returns the minimum between a and b, whereas $\mathrm{MAX}(a,b)$ returns the maximum.

TABLE 2
Terminal Set

Terminal name	Definition
For the TSR (t_i is the considered task)	
CN	The number of child tasks of the t_i
MRT	The max remaining time $MRT(t_i)$ of t_i
$rank_u$	The upward rank $\mathrm{rank_u}(t_i)$ of t_i used by HEFT
$rank_{oct}$	The average OCT of t_i used by PEFT
RN	The number of ready tasks
RP	The proportion of the remaining tasks
For the RSR (t_i is the selected task from the TSR and p_j is the considered processor)	
$w_{i,j}$	The execution time of t_i on p_j
EST	The earliest start time $EST(t_i, p_j)$ of t_i on p_j
OCT	The optimistic cost table $OCT(t_i, p_j)$
ROT	The realtime optimistic time $ROT(t_i, p_j)$
AT	The available time $avail(p_j)$ of processor p_j

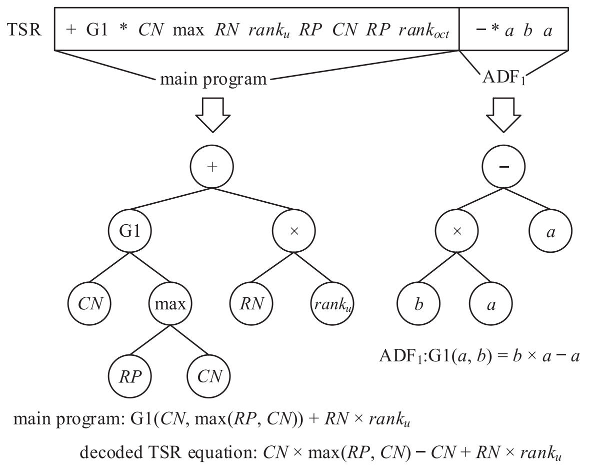

TSR
+ G1 * CN max RN rank_u RP CN RP rank_oct
− * a b a
main program
ADF1
+
G1
×
CN
max
RN
rank_u
RP
CN
−
×
a
b
a
ADF1:G1(a, b) = b × a − a
main program: G1(CN, max(RP, CN)) + RN × rank_u
decoded TSR equation: CN × max(RP, CN) − CN + RN × rank_u

Fig. 4. A sample gene expression TSR of an individual in CCGP using function and terminal sets in Tables 1 and 2.

of predecessors of task t_k. Therefore, we only consider the preceding tasks that have already been executed and the task t_i when the $EST(t_k, p_l)$ is calculated. The $ROT(t_i, p_j)$ forecasts the effect of t_i running on p_j to schedule succeeding tasks of t_i. In general, we prefer scheduling task t_i on the resource that can minimize the ROT of t_i. Note that for an exit task t_{exit}, the $ROT(t_{exit}, p_j)$ is equal to $EFT(t_{exit}, p_j)$.

Based on the above five features, we create two terminal sets for the construction of TSR and RSR, respectively. The description of the two terminal sets can be found in Table 2. The designed terminal sets include the variables that HEFT and PEFT used, i.e., the upward rank, the OCT and the OCT rank. Hence, the HEFT and PEFT heuristics are in the search space of CCGP algorithm. For example, the HEFT heuristic can be expressed as $\Gamma_{\mathrm{TSR}} = rank_u(t_i)$ and $\Gamma_{\mathrm{RSR}} = EFT(t_i, p_j) = EST(t_i, p_j) + w_{i,j}$, while the PEFT heuristic can be expressed as $\Gamma_{\mathrm{TSR}} = rank_{oct}(t_i)$ and $\Gamma_{\mathrm{RSR}} = OCT(t_i, p_j)$. Based on these low-level heuristics, the proposed CCGP algorithm tries to learn better high-level heuristics by non-linearly combining the low-level heuristics.

The terminals, i.e., the low-level heuristics, are designed based on the global information of workflows. Therefore, the learned high-level heuristics are suitable for solving the static workflow scheduling problem, like most of heuristic-based methods (e.g., HEFT and PEFT). However, if the terminals are designed by putting the dynamic changed information into consideration, the learned heuristics have potential to cope with dynamic workflows.

5.2 The CCGP Algorithm

Our CCGP algorithm is based on a recent published GP variant SL-GEP and the CC mechanism. In SL-GEP, each chromosome consists of a "main program" and several Automatically Defined Functions (ADFs). A chromosome can be converted to an equivalent expression tree by using the breadth first traversal scheme. Based on the expression tree, we can get an equivalent formula. For WSP, the main program is a priority function representing the TSR or the RSR. Fig. 4 demonstrates a typical chromosome example representing the TSR using the function and terminal sets described in Tables 1 and 2.

To learn general and effective high-level heuristics, a large number of training instances with different properties are used as training cases. Since the *makespan* of instances might vary significantly, it is necessary to normalize the *makespan* (also called the schedule length) to a lower bound. Thus, we use the schedule length ratio (SLR) as in [8], [10], which is defined as follow:

$$SLR = \frac{makespan}{\sum_{t_i \in CP_{MIN}} \min_{p_j \in P}\{w_{i,j}\}}, \tag{14}$$

where CP_{MIN} is the set of the critical path tasks and the execution time of each task takes the minimum computation cost. The denominator of SLR is the summation of execution time of the tasks on CP_{MIN}, i.e., the minimum makespan. The lower the SLR is, the better the solution is. Hence, the fitness of a candidate solution (i.e., a pair of Γ_{TSR} and Γ_{RSR}) is the average SLR over the set of training instances I, i.e.,

$$f(\Gamma_{\mathrm{TSR}}, \Gamma_{\mathrm{RSR}}) = \frac{\sum_{j=1}^{|I|} SLR_{I_j}}{|I|}. \tag{15}$$

In the proposed framework, two sub-populations (P_{TSR} and P_{RSR}) are used to evolve the TSR and RSR, respectively. To evaluate an individual of one sub-population, the individual will be combined with a representative individual of another sub-population to form a complete solution. The fitness of an individual is the fitness of the corresponding complete solution. We denote the representatives of P_{TSR} and P_{RSR} as P^r_{TSR} and P^r_{RSR} respectively. For example, the fitness of an individual P^i_{TSR} of P_{TSR} is calculated by $f(P^i_{TSR}) = f(P^i_{TSR}, P^r_{RSR})$. In the initialization phase, the representations of sub-populations are selected randomly, while in the evolution phase, the best individuals of sub-populations are selected as the representations. The pseudo code of the proposed CCGP is shown in Algorithm 2, which consists of the following four main steps.

1) Step 1 – Initialization: The first step is to generate individuals of two initial sub-populations P_{TSR} and P_{RSR}. Each dimension of chromosome is randomly assigned with functions, terminals or variables according to its type. The fitness value of an individual in each sub-population is determined by evaluating the whole heuristics formed by the individual and a randomly selected individual of another sub-population. After that, the two sub-populations are evolved by mutation, crossover and selection operations, which are described as below.

2) Step 2 – Mutation: In SL-GEP, the common "DE/current-to-best/1" mutation scheme is used to generate mutant vectors, i.e.,

$$y_i = x_i + F \cdot (x_{best} - x_i) + F \cdot (x_{r_1} - x_{r_2}), \tag{16}$$

where r_1 and r_2 are two distinct individual indices which are different from $i.\boldsymbol{x}_{best}$ is the best individual with minimum fitness value. F is the scaling factor of mutation. As described in [23], the mutation probability of each dimension $x_{i,j}$ of individual $\boldsymbol{x}_i$ is calculated by

$$\varphi = 1 - (1 - F \cdot \psi(x_{best,j}, x_{i,j})) \times (1 - F \cdot \psi(x_{r1,j}, x_{r2,j})), \tag{17}$$

where $\psi(a, b)$ is defined as

$$\psi(a,b) = \begin{cases} 1, & \text{if } a \neq b \\ 0, & \text{otherwise} \end{cases}. \tag{18}$$

When a dimension $x_{i,j}$ is to be mutated, a new value is generated to assign $y_{i,j}$ by using the "frequency-based assignment" described in [23] if $x_{i,j}$ is part of the main program. Otherwise, $y_{i,j}$ is assigned randomly. The key idea of the frequency-based assignment is to select a value based on the frequencies of building blocks (e.g., functions and terminals) in the population. Those with higher frequencies are more likely to be selected and assigned to $y_{i,j}$. The frequency calculations of the TSR and RSR are separated and independent, for the characters of the TSR are different from those of the RSR.

3) Step 3 – Crossover: In this step, a trail vector $\boldsymbol{u}$ will be created by:

$$u_j = \begin{cases} y_{i,j}, & \text{if } \mathrm{rand}(0,1) < CR \text{ or } j = k \\ x_{i,j}, & \text{otherwise} \end{cases}, \tag{19}$$

where *CR* is the crossover rate assigned from 0 to 1 randomly, *k* is a random value between 1 and *D*, and *j* is the *j*th dimension of the individual $\boldsymbol{x}_i$. Each individual $\boldsymbol{x}_i$ will get a new trail vector $\boldsymbol{u}$ and then $\boldsymbol{u}$ will compete with $\boldsymbol{x}_i$ in the selection operation.

4) Step 4 – Selection: In the selection operation, the final individual $\boldsymbol{x}_i$ will select the vector with better fitness between $\boldsymbol{x}_i$ and $\boldsymbol{u}$ as offspring:

$$x_i = \begin{cases} u, & if\ f(u) < f(x_i) \\ x_i, & \text{otherwise} \end{cases}. \tag{20}$$

The fitness of the $\boldsymbol{u}$ is evaluated by the complete solution formed by $\boldsymbol{u}$ and the representative of the other sub-population.

Step 2, Step 3 and Step 4 are repeated until reaching the termination conditions such as reaching the maximum number of generations. Finally, the best heuristics evolved by the CCGP are the combination of the best individual of each sub-population.

6 Experimental Studies

In this section, we investigate the effectiveness of the proposed framework for the WSP. The performance metrics are firstly presented and the experimental settings are then described. Lastly, the high-level heuristics learned by CCGP are compared with several well-known human designed heuristics and an EA algorithm.

Algorithm 2. CCGP

```
Input: training set $I$; population size $NP$, number of ADFs $K$,
head length of main program and ADFs $h, h'$
Output: the best TSR and RSR
Procedure:
 1. initialize two populations ($P_{TSR}$ and $P_{RSR}$), and evaluate fit-
    ness of individuals of each population
 2. while stopping criterion not met do
 3.   $P^r_{TSR}$, $P^r_{RSR}$ ← best individual of $P_{TSR}, P_{RSR}$
 4.   for each sub-population $P$ do
 5.     update frequencies of functions and terminals of the
        sub-population
 6.     for $i = 1$ to $NP$ do
 7.       $F = \mathrm{rand}(0,1)$;$CR = \mathrm{rand}(0,1)$; $k = \mathrm{rand}(1, D)$
 8.       randomly select two individuals $\boldsymbol{x}_{r1}$ and $\boldsymbol{x}_{r2}$ where $r_1$
          and $r_2$ are different and unequal to $i$
 9.       for $j = 1$ to $D$ do
10.         calculate the mutation probability $\varphi$ by (18)
11.         if (rand $(0,1) < CR$ or $j = k$) and (rand $(0,1) < \phi$)
            then
12.           $u_j$ ← "frequency-based assignment"
13.         else
14.           $u_j = x_{i,j}$
15.         end if
16.       end for
17.       if $f(\boldsymbol{u}) < f(P^i)$ then
18.         $P^i = \boldsymbol{u}$
19.       end if
20.     end for
21.     update the best individual indexes
22.   end for
23. end while
24. return $P^r_{TSR}$, $P^r_{RSR}$
```

6.1 Performace Metrics

In the experiment study, four commonly used performance metrics [8], [9], [10] are used for comparison. These performance metrics are described as follows:

1) *Schedule Length Ratio (SLR):* The makespan (schedule length) is most commonly used to evaluate a schedule of a DAG, but it is usually normalized to a lower bound to test graphs with different parameters. The average SLR of all test instances, which is calculated by equation (14), are used to evaluate the performance of different heuristics. Generally, the heuristic that yields lower average SLR performs better.
2) *Speedup:* The speedup *S* is defined as

$$S = \frac{\min_{p_j \in P}\{\sum_{t_i \in V} w_{i,j}\}}{makespan}, \tag{21}$$

where the numerator is the single-resource execution time computed by assigning all tasks to the fastest resource that minimizes the total execution time of all the tasks. The *makespan* is the execution time in the case that the tasks can be executed in parallel and it is usually less than the total execution time if the tasks can only be executed sequentially. The speedup metric indicates how much speedup we can gain by using multiple resources than a single resource.

3) *Efficiency:* For a heterogeneous environment, the efficiency [34] of a schedule is defined as the ratio of the speedup value to the number of resources, where each resource is weighted with the relative speedup compared with the fastest one:

$$E = \frac{S}{\sum_{p_j \in P} S_{p_j}}, \text{ where} : S_{p_j} = \frac{\min_{p_k \in P}\{\sum_{t_i \in V} w_{i,k}\}}{\sum_{t_i \in V} w_{i,j}}. \tag{22}$$

The efficiency formula then becomes

$$E = \frac{makespan^{-1}}{\sum_{p_j \in P} \left(\sum_{t_i \in V} w_{i,j}\right)^{-1}}. \tag{23}$$

The average efficiency is used as a performance metric for comparison.

4) *Number of occurrences of better schedules*: The percentage of better, equal, and worse schedules generated by the learned high-level heuristics and competitors are also investigated in our experiment.

6.2 Experimental Settings

In the CCGP, there are four hyper parameters: the population size *NP*, the number of ADFs *K*, the head length of main program and ADFs *h* and *h'*, respectively. For selecting the suitable parameters for our problem, we do some experiments on them. For the population size, we vary it in the set of values $\{10, 30, 50, 100, 200\}$ and found that the better parameters are from 30 to 100. If the size is too low or high, the result will get worse because small population will loss divergence and converge fast into a local optimal value, while large population needs much more time to converge. Hence, we set $NP = 50$. For parameters *h*, by changing it's value from 5 to 40, the experiment showed that *h* near 20 achieves the best results. The chromosome with a short head length can only expresses limited functions, while the chromosome with longer length can express more functions but result in a larger search space. Thus, for balancing the expression ability and search efficiency, we set $h = 20$. Similarly, we set $K = 2, h' = 3$ based on our empirical experiments.

In our experiment, we choose five well-known human designed heuristics and a recently published EA algorithm for comparison, i.e., the HEFT [8], Lookahead [9], PEFT [10], PETS [12], MSL [13], and PSO [5]. In [5], the PSO is used to minimize the monetary cost while meeting deadline constraints. We adjust the PSO by removing the constraints and modify the optimal objective to minimize the makespan for comparison analysis. We consider two categories of workflows used for performance evaluation: randomly generated workflows and four well-known real-world workflows. The random workflows are generated by random DAG generator with different parameters and the four real-world workflows are the Gaussian Elimination, Fast Fourier Transform (FFT) [8], [10], Montage [1] and Epigenomics [2].

6.3 Case Study 1: Randomly Generated Workflows

First, the randomly generated workflows are used to test the performance of the learned high-level heuristics. In [35], several random graph generation methods are provided, and the Erdős-Rényi method $G(v, \rho)$ is selected to generate random workflow graph, where v is the number of vertices and ρ is the link probability of vertices. In the generation method, a vertice v_i and each vertice $v_j\,(j < i)$ are connected to form an edge $e_{i,j}$ with the link probability ρ, which is set as fixed 0.1 in the experiment. Then, pn heterogeneous resources are allocated to schedule the generated workflow. After the topology structure of the workflow is determined and the resources are allocated, the computation costs and communication costs of tasks on the given resources are computed by using the following two parameters [8]:

- *CCR*: The communication-computation ratio *CCR* is the ratio of the average communication cost to the average computation cost in a DAG;
- *β*: The range percent of computation costs on resources *β* is the heterogeneity factor for resource speeds. The difference between the computation costs of a task on resources is large if the *β* value is high. Otherwise, the computation costs are nearly equal. The average computation cost $\overline{w_i}$ of task t_i is selected from a uniform distribution with range $[0, 2 \times \overline{w_{DAG}}]$ where $\overline{w_{DAG}}$ is the average computation cost of whole DAG and is randomly set from 1 to 10 in the experiment. The computation cost $w_{i,j}$ of task t_i on resource p_j is set within the range:

$$\overline{w_i} \times \left(1 - \frac{\beta}{2}\right) \leq w_{i,j} \leq \overline{w_i} \times \left(1 + \frac{\beta}{2}\right). \tag{24}$$

In the experiment, we use the following parameter settings for generating random instances: $v = \{30, 50, 100, 200\}$, $CCR = \{0.1, 1, 2\}, \beta = \{0.5, 1, 2\}, pn = \{4, 8, 16, 32\}$. There are 144 parameter combinations, each of which is used to generate five different instances consisting of a random workflow and some resources used to execute tasks of the workflow. In this way, we generate 720 training instances and 720 test instances for train and testing.

For the randomly generated workflows, the best high-level heuristics learned by the proposed framework, i.e., the best priority functions Γ_{TSR} and Γ_{RSR}, are

$$\begin{aligned}\Gamma_{\text{TSR}} = &\max(\min(CN \times rank_{oct}, 2 \times rank_u) - MRT \\ &+ \max(2 \times (rank_{oct} + \sqrt{rank_{oct}}), rank_{oct}), rank_u),\end{aligned} \tag{25}$$

$$\Gamma_{\text{RSR}} = 1/(ROT \times (ROT + \max(ROT, OCT)) \times \log(ROT)). \tag{26}$$

As can be seen from the formulas, some terminals are not used. For example, terminals *RN* and *RP* are not in the Γ_{TSR}, which indicates that they may be less important for constructing effective scheduling heuristics. To investigate the

TABLE 3
Comparison on Different Metrics Between the Learned High-Level Heuristics and Other Algorithms For Random Workflows Case

Algorithm	SLR		Speedup		Efficiency	
	mean	p-value	mean	p-value	mean	p-value
LH	**2.416**	-	**6.007**	-	**0.633**	-
PEFT	2.513	3.67E-18	5.860	1.21E-29	0.610	2.27E-29
Lookahead	2.690	4.36E-29	5.640	7.97E-64	0.597	2,84E-63
HEFT	2.686	3.53E-30	5.696	1.77E-59	0.605	1.12E-47
MSL	2.937	7.59E-69	5.219	2.64E-131	0.526	6.39E-97
PETS	2.849	2.45E-55	5.347	2.49E-106	0.547	3.76E-83
PSO	3.945	2.78E-59	3.690	2.76E-46	0.418	2.78E-82

importance of a terminal in the formulas, we add noise to the terminal while fixing the other terminals. The larger the SLR metric changes, the more importance the terminal is. The experimental results show that the terminal *ROT* has the greatest impact on SLR, and the other terminals only have a slight influence on SLR. These results indicate that the terminal *ROT* defined by us plays a more important role in scheduling workflows.

After learning the best high-level heuristics, we compare it with other algorithms on test set to investigate its effectiveness and robustness. The means on three metrics are presented on Table 3. Here (and in the rest of this paper) we use LH to represent the learned heuristics evolved by the CCGP algorithm. The LH performs better than other heuristics in terms of all metrics. To confirm whether the mean of LH is significantly less than the means of other algorithms, the paired student's *t*-test is used. The obtained *p*-values at level 0.01 are also given in Table 3. We can see that all *p*-values are less than 0.01, therefore, our LH is significantly better than other algorithms. The experimental results indicate that the CCGP algorithm is effective to utilize the low-level heuristics to construct and learn better high-level heuristics. As can be seen from Table 3, the PSO gets worse results than heuristic-based algorithms. This is because the method in [5] doesn't consider the execution order of tasks. However, there are many kinds of task execution orders in a workflow as long as the dependencies between tasks are met. Different orders could lead to significant different scheduling results. The PSO method in [5] executes tasks according to the task indexes which are fixed in advance. Since the order based on the task indexes usually is not the best execution order, the PSO method usually could not find the best results. However, in the heuristic based algorithms and our method, the order of tasks can be flexible decided by certain heuristics. Thus, they are capable of finding better results.

Table 4 shows the percentage of better, equal and worse schedules produced by the LH for all test instances compared with other heuristics. It can be seen that the LH achieves better SLR than PEFT by 66%, Lookahead by 82%, HEFT by 80%, MSL by 95%, PETS by 92% and PSO by 98%, respectively. The LH performs better than (or equal to) the second best algorithm PEFT on most of test instances (about 76%).

As for the time complexities of the heuristic-based methods, they usually contain the calculations of heuristic variables (e.g., the upward rank in HEFT [8], and the optimistic cost table in PEFT [10]) and the complexity of workflow scheduling algorithm HBWS. Firstly, the computation complexities of the heuristic variables are different. For example, the computation complexity of the upward rank in HEFT [8] is $O(v^2p)$ and the OCT table in PEFT [10] also needs $O(v^2p)$ for computation. Secondly, the time complexity of the HBWS algorithm is $O(v(v \cdot t_1 + p \cdot t_2))$ where t_1 is the time complexity of Γ_{RSR} and t_2 is the time complexity of Γ_{RSR}. In the HEFT and PEFT, the variables are ready in the preparation phrase, so both t_1 and t_2 are 1. Thus, their total complexities are $O(v^2p + v(v + p))$. Since the number of tasks is usually larger than the number of resources, the complexity can be simplified as $O(v^2p)$.

TABLE 4
Schedule Length Ratio Comparison Between the Learned High-Level Heuristics and Other Algorithms

		PEFT	Lookahead	HEFT	MSL	PETS	PSO
LH	Better	66%	82%	80%	95%	92%	98%
	Equal	10%	6%	5%	1%	2%	0%
	Worse	24%	12%	15%	4%	6%	2%

In this paper, the computation complexities of the heuristic variables designed in Section 5.1 are $O(v^2p)$. For the time complexity of HBWS, t_1 is 1 and t_2 is v because of the computation of real-time variable *ROT*. Therefore, the total complexity of our hyper-heuristic method is $O(v^2p + v(v+ vp)) = O(v^2p)$, which is of the same order as the HEFT and the PEFT. In the experiment, for the averaging running time of all algorithms to schedule a workflow in test set, the LH and Lookahead take about 0.023 seconds and the other heuristics take about 0.01 seconds, while PSO takes about 9 seconds, which means that the PSO requires much more computational cost than heuristic base methods.

Fig. 5a shows the average SLR with respect to the number of tasks for randomly generated workflows. It can be observed that as the number of tasks increases, the SLR of all algorithms increases. The LH outperforms other algorithms on four different number of tasks. The result of the average efficiencies of all heuristics as a function of resource number is shown in Fig. 5b. We can see that among all heuristics, the LH obtains the best efficiency on different number of resources, and with the increasing number of resources, the corresponding efficiency of each heuristic is gradually reduced. The reason is that the makespan does not decrease linearly as the number of resources increases, since it is limited by the dependencies between tasks. To explore the influence of DAG density on the metric SLR, we generate different density workflows for testing. Specifically, in the random workflow generator, we vary the link probability ρ of tasks from 0.1 to 1, and other parameters are set the same as before. The final result is given in Fig. 5c. It can be observed that the LH gets the best performance in all cases, and heuristic based algorithms can handle workflows with density 0.1, 0.8 or 1 better than the workflows with medium density 0.3 or 0.5.

To check whether there is a significant difference among median SLR obtained by all algorithms, the Friedman test [36], [37] is applied. The Friedman test is a non-parametric

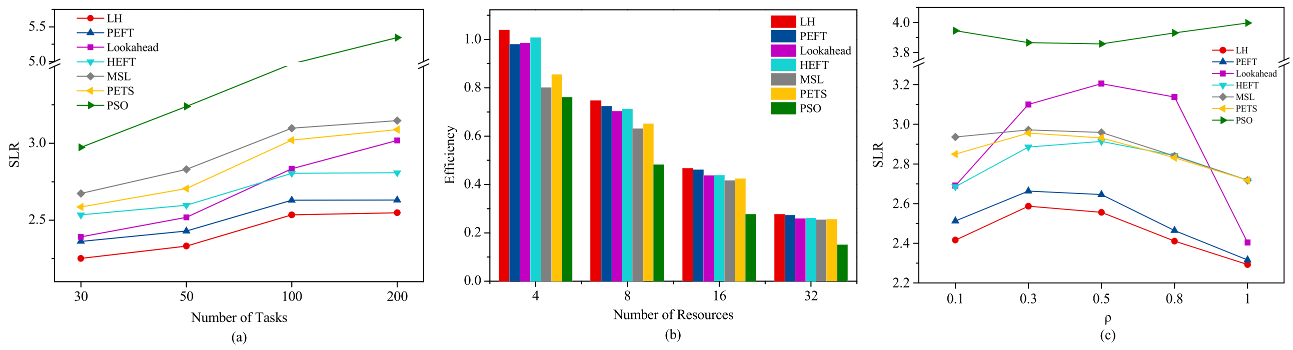


Fig. 5. (a) Average SLR as a function of the tasks number, (b) average efficiency as a function of the resource number, and (c) average SLR as a function of the link probability for randomly generated workflows.

statistical test for detecting differences in treatments across multiple test attempts. Specifically, according to the different parameter setting combinations, the test instances are divided into 144 groups. In each group, all heuristics are ranked in terms of the obtained average SLR. Then we calculate the average ranking of different heuristics and the Friedman value. As shown in Table 5, the obtained Friedman value is 2512.420, which is greater than the critical value $\chi^2_{.01} = 16.81$ at a level of $\alpha = 0.01$. This result indicates that the difference of the average SLR among the heuristics is significant with the error rate $p < 0.01$. We follow the Friedman test with Bonferroni–Dunn's test that is utilized as Post-hoc test to compare the difference between two heuristics. The critical difference (CD), which is the minimum required difference between the average ranks of any two heuristics, is 0.568 at level $\alpha = 0.01$. To compare the difference between LH and other algorithms, the threshold value is used, which is calculated by the average rank of the LH plus the CD (i.e., $1.299 + 0.568 = 1.867$, as shown in Table 5). If the average rank of a heuristic is larger than the threshold value, the heuristic is significantly worse than the LH. The results in Fig. 6 indicate that the LH learned by our framework performs significantly better than other human designed heuristics on the randomly generated workflows.

6.4 Case Study 2: Real-World Workflows

Next, we consider four real-world workflows for testing. The topology structures of these real-world workflows are shown in Fig. 7. The parameters CCR, β and pn are set the same as the randomly generated workflows when generating real-world workflow instances. The other settings of generating the real-world workflows are described as follows:

TABLE 5
Friedman Test for Random Workflows at $\alpha = 0.01$

Algorithm	LH	PEFT	Lookahead	HEFT	MSL	PETS	PSO
Average ranking	**1.299**	2.333	3.556	3.250	5.799	5.042	6.722
Friedman value	2512.420						
$\chi^2_{.01}$	16.81						
CD	0.568						
Threshold	1.867						

1) *Gaussian Elimination.* Gaussian Elimination is an algorithm used to solve systems of linear equations in linear algebra. The algorithm is parallelized and represented by a DAG [8]. The DAG consists of the pivot column operation tasks and update operation tasks. For a Gaussian Elimination DAG, the matrix size m determines the number of tasks which is equal to $(m^2 + m - 2)/2$. In our experiment, the range of m is set to be $\{5, 10, 15, 20\}$, i.e., the number of tasks is $\{14, 54, 119, 209\}$.
2) *Fast Fourier Transform.* The fast Fourier transform (FFT) algorithm computes the discrete Fourier transform of a sequence, or its inverse. As described in [8], [10], the FFT algorithm consists of recursive calls and the butterfly operation. If the size of input vector is m, there will be $2 \times m - 1$ recursive call tasks and $m \times \log_2 m$ butterfly operation tasks in an FFT graph. The parameter m varies from 4 to 32 incrementing by the power of 2 and the corresponding number of tasks varies from 15 to 223.
3) *Montage.* Montage [1], [2] has been designed as a toolkit for assembling Flexible Image Transport System image into custom mosaics. The DAG of Montage workflow in Fig. 7c is a sample graph with 20 tasks. In our experiment, the number of Montage DAGs is set to be 25 or 50, which follows the settings in PEFT [10].
4) *Epigenomics.* The Epigenomics workflow [2] is used to map the epigenetic state of human cells on a genome-wide scale. We consider the Epigenomics

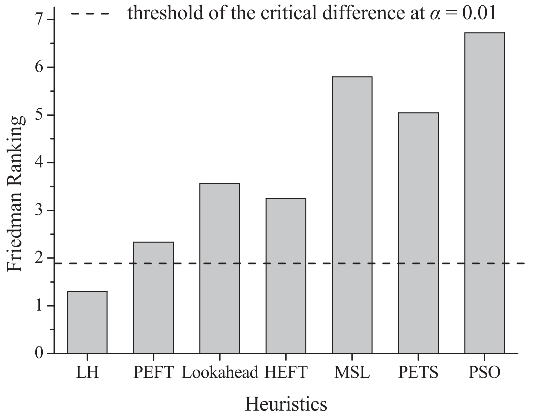


Fig. 6. Bonferroni-Dunn graphic for randomly generated workflows.

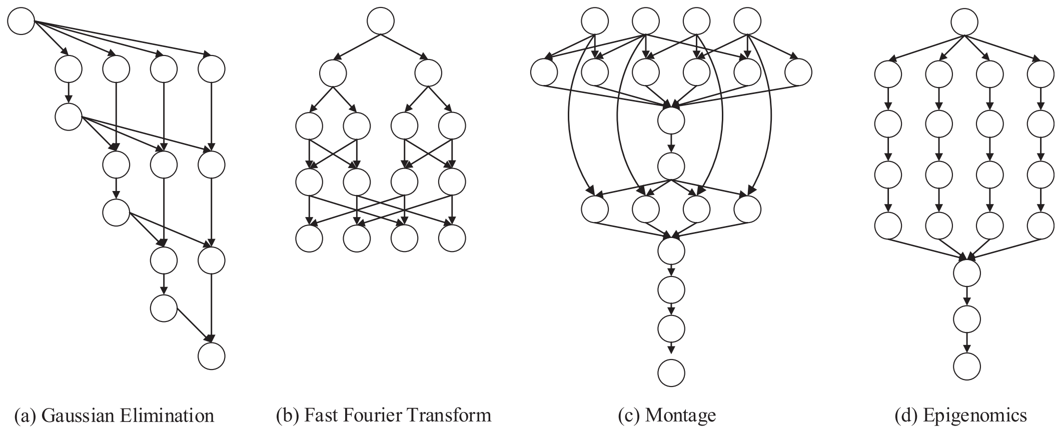


Fig. 7. Examples of real-world workflows.

TABLE 6
Comparison on Different Metrics for Real-World Workflows Case

Algorithm	Gaussian Elimination			FFT			Montage			Epigenomics		
	SLR	S	E	SLR	S	E	SLR	S	E	SLR	S	E
LH	**2.452**	**4.360**	**0.465**	**2.716**	**7.163**	**0.683**	**2.554**	**3.628**	**0.408**	**2.368**	**4.163**	**0.465**
PEFT	2.588	4.251	0.451	2.865	6.853	0.643	2.702	3.495	0.389	2.427	4.085	0.452
Lookahead	2.650	4.188	0.449	2.772	7.052	0.672	2.648	3.573	0.402	2.498	4.056	0.456
HEFT	2.874	4.057	0.437	2.953	6.926	0.665	2.856	3.468	0.392	2.750	3.898	0.443
MSL	3.197	3.601	0.370	3.479	5.547	0.511	2.920	3.384	0.377	2.791	3.740	0.411
PETS	3.170	3.608	0.370	3.419	5.561	0.517	2.860	3.419	0.384	2.832	3.699	0.407
PSO	3.972	2.693	0.314	4.232	4.041	0.445	3.204	2.831	0.335	3.292	3.112	0.375

workflows with 24 and 46 tasks in the experiment, following the setting in PEFT [10].

The experimental performance metrics of each algorithm on real-world workflows test set are listed in Table 6, where S and E represents speedup and efficiency respectively. The results show that the LH obtains the best results in all cases. The results indicate that the CCGP algorithm can learn better high-level heuristics than human-made heuristics for the real-world workflows. The PSO performs worst since it doesn't take the tasks' execution order into consideration as discussed in the random workflows case.

Similarly, the Friedman test is also applied to check whether the obtained average SLR of all heuristics is significantly different in the real-world workflow cases. The test instances of four real-world workflows are divided into total 432 groups according to the different workflows and parameter setting combinations. The average ranks of the heuristics are shown in Table 7. The obtained Friedman value is 2501.230, which is greater than the critical value $\chi^2_{.01} = 16.81$. Thus, the difference of the average SLR among the heuristics is significant at level $\alpha = 0.01$. Bonferroni–Dunn's test is also used and the critical difference (CD) is 0.568 at level $\alpha = 0.01$. Fig. 8 illustrates the average ranks of all heuristics and the results show that the LH is significantly better than other methods.

TABLE 7
Friedman Test for Real Workflows at $\alpha = 0.01$

Heuristic	LH	PEFT	Lookahead	HEFT	MSL	PETS	PSO
Average ranking	**1.396**	2.813	2.403	3.694	5.583	5.556	6.556
Friedman value	2501.230						
$\chi^2_{.01}$	16.81						
CD	0.568						
Threshold	1.964						

7 Conclusion

In this paper, we have proposed a cooperative coevolution hyper-heuristic framework to automatically learn high-level

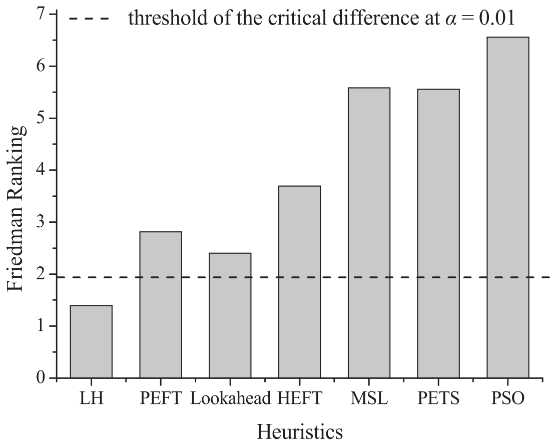


Fig. 8. Bonferroni-Dunn graphic for real-world workflows.

heuristics to solve the WSP. In the proposed framework, workflows are scheduled by the HBWS algorithm with the given heuristics (i.e., TSR and RSR). At each step, the TSR is used to select task, while the RSR is used to select resource to execute the selected task. Furthermore, a CCGP algorithm has been developed to automatically learn these two sub-heuristics. To improve the learning efficiency, effective low-level heuristics have been defined as building blocks to construct the high-level heuristics. In addition, the SL-GEP and the cooperative coevolution mechanism are adopted to implement the CCGP algorithm. In the experimental study, the randomly generated workflows and four real-world workflows are used as the training and test cases. Five well-known heuristics (i.e., the HEFT, Lookahead, PEFT, MSL and PETS) and a recently published EA-based method have been considered as the baselines for comparison. The experimental results showed that the high-level heuristics evolved by the proposed CCGP algorithm can offer very promising performance in terms of SLR, speedup and efficiency.

The advantage of the proposed cooperative coevolution hyper-heuristic framework is that the heuristics learned by the proposed algorithm are general enough to be applied to different workflows. However, the interpretability of the learned heuristics is not good enough. In the future, we plan to extend our method by utilizing multi-objective optimization techniques to optimize both accuracy and interpretability of the heuristics. Meanwhile, we would like to design more effective functions and terminals to further improve the search efficiency. Furthermore, applying the proposed framework to solve dynamic WSP is a promising research direction.

Acknowledgments

This work is supported under the National Natural Science Foundation of China (Grant No. 61602181, 61876025, 61872282, 61876065), Program for Guangdong Introducing Innovative and Entrepreneurial Teams (Grant No.: 2017ZT07X183), Guangdong Natural Science Foundation Research Team (Grant No. 2018B030312003), Natural Science Foundation of Guangdong Province, China (No.2018A0303130022), and Natural Science Basic Research Plan in Shaanxi Province of China (Program No. 2019JM-031).

References

[1] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M.-H. Su, “Montage: A Grid-enabled engine for delivering custom science-grade mosaics on demand,” *Proc. SPIE*, vol. 5493, pp. 221–234, 2004.

[2] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, “Characterization of scientific workflows,” in *Proc. IEEE 3rd Workshop Workflows Support Large-Scale Science*, 2008. pp. 1–10.

[3] R. C. Corrêa, A. Ferreira, and P. Rebreyend, “Scheduling multiprocessor tasks with genetic algorithms,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 8, pp. 825–837, Aug. 1999.

[4] F. A. Omara and M. M. Arafa, “Genetic algorithms for task scheduling problem,” *J. Parallel Distrib. Comput.*, vol. 70, no. 1, pp. 13–22, 2010.

[5] M. A. Rodriguez and B. Rajkumar, “Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds,” *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr.-Jun. 2014.

[6] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, “Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 29, no. 6, pp. 911–924, Jun. 2010.

[7] W.-N. Chen and J. Zhang, “An ant colony optimization approach to a grid workflow scheduling problem with various Qos requirements,” *IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.)*, vol. 39, no. 1, pp. 29–43, Jan. 2009.

[8] H. Topcuoglu, S. Hariri, and M.-Y. Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[9] L. F. Bittencourt, R. Sakellariou, and E. R. Madeira, “Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm,” in *Proc. IEEE 18th Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, 2010, pp. 27–34.

[10] H. Arabnejad and J. G. Barbosa, “List scheduling algorithm for heterogeneous systems by an optimistic cost table,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.

[11] G. C. Sih and E. A. Lee, “A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 175–187, Feb. 1993.

[12] E. Ilavarasan and P. Thambidurai, “Low complexity performance effective task scheduling algorithm for heterogeneous computing environments,” *J. Comput. Sci.*, vol. 3, no. 2, pp. 94–103, 2007.

[13] D. Sirisha and G. V. Kumari, “A new heuristic for minimizing schedule length in heterogeneous computing systems,” in *Proc. IEEE Int. Conf. IEEE Elect. Comput. Commun. Technol.*, 2015, pp. 1–7.

[14] N. Zhou, D. Qi, X. Wang, Z. Zheng, and W. Lin, “A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table,” *Concurrency Comput.: Practice Exp.*, vol. 29, 2016, Art. no. e3944.

[15] Z. Cai, X. Li, and J. Gupta, “Heuristics for provisioning services to workflows in Xaas clouds,” *IEEE Trans. Serv. Comput.*, vol. 9, no. 2, pp. 250–263, Mar./Apr. 2016.

[16] G. Wang, Y. Wang, H. Liu, and H. Guo, “HSIP,” *Sci. Program.*, vol. 2016, 2016, Art. no. 19.

[17] T. Yang and A. Gerasoulis, “DSC: Scheduling parallel tasks on an unbounded number of resources,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 9, pp. 951–967, Sep. 1994.

[18] H. Kanemitsu, M. Hanada, and H. Nakazato, “Clustering-Based Task Scheduling in a Large Number of Heterogeneous Resources,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3144–3157, 2016.

[19] S. Ranaweera and D. P. Agrawal, “A task duplication based scheduling algorithm for heterogeneous systems,” in *Proc. 14th Int. Parallel Distrib. Process. Symp.*, 2000, pp. 445–450.

[20] R. Bajaj and D. P. Agrawal, “Improving scheduling of tasks in a heterogeneous environment,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 2, pp. 107–118, Feb. 2004.

[21] J. R. Koza, *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[22] M. A. Potter and K. A. De Jong, “Cooperative coevolution: An architecture for evolving coadapted subcomponents,” *Evol. Comput.*, vol. 8, no. 1, pp. 1–29, 2000.

[23] J. Zhong, Y.-S. Ong, and W. Cai, “Self-learning gene expression programming,” *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 65–80, Feb. 2016.

[24] J. Zhong, L. Feng, and Y.-S. Ong, “Gene expression programming: A survey,” *IEEE Comput. Intell. Mag.*, vol. 12, no. 3, pp. 54–57, Aug. 2017.

[25] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, “Automated design of production scheduling heuristics: A review,” *IEEE Trans. Evol. Comput.*, vol. 20, no. 1, pp. 110–124, Feb. 2016.

[26] S. Nguyen, Y. Mei, and M. Zhang, “Genetic programming for production scheduling: A survey with a unified framework,” *Complex and Intelligent Systems*, Berlin, Germany: Springer, vol. 3, no. 1, pp. 41–66, 2017.

[27] J. C. Tay and N. B. Ho, “Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems,” *Comput. Ind. Eng.*, vol. 54, no. 3, pp. 453–473, 2008.

[28] D. Yska, Y. Mei, and M. Zhang, “Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling,” in *Proc. Eur. Conf. Genetic Program.*, vol. 10781, 2018, pp. 288–303.

[29] J. V. Hansen, “Genetic search methods in air traffic control,” *Comput. Operations Res.*, vol. 31, no. 3, pp. 445–459, 2004.

[30] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 210–224, Apr. 2012

[31] N. García-Pedrajas, C. Hervás-Martínez, and D. Ortiz-Boyer, "Cooperative coevolution of artificial neural network ensembles for pattern classification," *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 271–302, Jun. 2005.

[32] T. Estrada, M. Wyatt, and M. Taufer, "A genetic programming approach to design resource allocation policies for heterogeneous workflows in the cloud," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2015, pp. 372–379.

[33] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.

[34] R. Prodan and T. Fahringer, "Overhead analysis of scientific workflows in grid environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 3, pp. 378–393, Mar. 2008.

[35] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J. M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *Proc. 3rd Int. ICST Conf. Simul. Tools Techn.*, 2010, Art. no. 60.

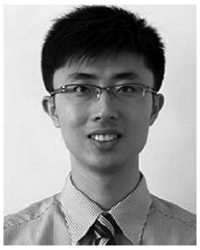
[36] M. S. Nash, *Handbook of Parametric and Nonparametric Statistical Procedures*. Boca Raton, FL, USA: CRC Press, 1997.

[37] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 special session on real parameter optimization," *J. Heuristics*, vol. 15, no. 6, pp. 617–644, 2009.

Qin-zhe Xiao is currently working toward the master's degree in the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His research interests include evolutionary computation and computer vision.

Jinghui Zhong received the PhD degree from the School of Information Science and Technology, Sun YAT-SEN University, China, in 2012. He is currently an associate professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. During 2013 to 2016, he worked as a postdoctoral research fellow at the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include evolutionary computation (e.g., genetic programming and differential evolution), machine learning, and agent-based modeling.

Liang Feng received the PhD degree from the School of Computer Engineering, Nanyang Technological University, Singapore, in 2014. He was a postdoctoral research fellow at the Computational Intelligence Graduate Lab, Nanyang Technological University, Singapore. He is currently an assistant professor with the College of Computer Science, Chongqing University, China. His research interests include computational and artificial intelligence, memetic computing, big data optimization and learning, as well as transfer learning.



Linbo Luo received the BEng(first class honours) and the PhD degrees in computer engineering from Nanyang Technological University (NTU), Singapore, in 2005 and 2011, respectively. After his PhD, he was a postdoctoral associate in the Singapore-MIT Alliance for Research and Technology Centre and then a research fellow in the Parallel and Distributed Computing Centre at NTU. He is currently an associate professor with Xidian University, China. His current research areas include complex system modeling and simulation, multi-agent systems and computational intelligence.

Jianming Lv received the BS degree in computer science from Sun YAT-SEN University, China, in 2002, and the PhD degrees from the Institute of Computing Technology, Chinese Academy of Sciences University, China, in 2008. He is currently an associate professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His research interests include data ming, computer vision, distributed computing and privacy.