electronics

MDPI

Article

Bi-Objective Workflow Scheduling on Heterogeneous Computing Systems Using a Memetic Algorithm

Yujian Zhang [1,2,*], Fei Tong [1,2], Chuanyou Li [1,3] and Yuwei Xu [1,2]

[1] Key Laboratory of Computer Network and Information Integration, Southeast University, Nanjing 211189, China; ftong@seu.edu.cn (F.T.); cyli@seu.edu.cn (C.L.); xuyw@seu.edu.cn (Y.X.)
[2] School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China
[3] School of Computer Science and Engineering, Southeast University, Nanjing 211189, China
* Correspondence: yjzhang@seu.edu.cn

Abstract: Due to the high power bills and the negative environmental impacts, workflow scheduling with energy consciousness has been an emerging need for modern heterogeneous computing systems. A number of approaches have been developed to find suboptimal schedules through heuristics by means of slack reclamation or trade-off functions. In this article, a memetic algorithm for energy-efficient workflow scheduling is proposed for a quality-guaranteed solution with high runtime efficiency. The basic idea is to retain the advantages of population-based, heuristic-based, and local search methods while avoiding their drawbacks. Specifically, the proposed algorithm incorporates an improved non-dominated sorting genetic algorithm (NSGA-II) to explore potential task priorities and allocates tasks to processors by an earliest finish time (EFT)-based heuristic to provide a time-efficient candidate. Then, a local search method integrated with a pruning technique is launched with a low possibility, to exploit the feasible region indicated by the candidate schedule. Experimental results on workflows from both randomly-generated and real-world applications suggest that the proposed algorithm achieves bi-objective optimization, improving makespan, and energy saving by 4.9% and 24.3%, respectively. Meanwhile, it has a low time complexity compared to the similar work HECS.

Keywords: energy efficiency; heterogeneous computing system; workflow scheduling; bi-objective; memetic algorithm

Citation: Zhang, Y.; Tong, F.; Li, C.; Xu, Y. Bi-Objective Workflow Scheduling on Heterogeneous Computing Systems Using a Memetic Algorithm. *Electronics* **2021**, *10*, 209. https://doi.org/10.3390/electronics10020209

Received: 15 December 2020
Accepted: 14 January 2021
Published: 18 January 2021

1. Introduction

A heterogeneous computing system (HCS) refers to a system that incorporates different types of processing units (PUs). It has been ubiquitous in both scientific and industrial applications, not only because it can provide parallel processing and high performance powered by large numbers of PUs, but also due to its high efficiency and scalability derived from the complementarity of diverse types of PUs [1]. For example, more than half of the top 10 supercomputing systems in the world employ CPU-accelerator heterogeneous architectures to maximize performance and efficiency [2]. Besides the optimization in hardware organization and architecture, the efficiency of a HCS heavily depends on the effective utilization of the PUs inside. Hence, extensive efforts have been made in task scheduling approaches [3–13], which are commonly regarded as software techniques to improve system efficiency. Generally, a parallel application to run in a HCS can be decomposed into a set of lightweight tasks with precedence constraints, which can be described by a directed acyclic graph (DAG) or a workflow. Traditional workflow scheduling schemes concentrate on minimizing the total completion time, namely makespan, without violating precedence constraints.

However, it has been found that the single metric from the aspect of time efficiency is insufficient to evaluate modern HCSs, since their power consumption has become a critical issue due to the high cost of energy along with the negative environmental impacts. According to the report by National Resources Defense Council (NRDC) in the USA,

the data centers in 2013 consumed 91 billion kWh of electricity, comparable to the production of 34 large coal-fired power plants [14]. Furthermore, global data centers are predicted to cost 5% of the world's electricity production while causing 3.2% of the worldwide carbon emissions by 2025 [15]. On the other hand, a large portion of PUs tend to have relatively low average utilization, spending most of their time in the 10–50% utilization range [16], which results in a massive waste of electricity and resources. Therefore, a growing number of workflow scheduling approaches have been developed to accommodate the needs for energy reduction coupled with makespan minimization [17–25]. With the aid of the dynamic voltage frequency scaling (DVFS) technique that has been incorporated into common processors, schedulers are enabled to reduce the energy consumption at the expense of processing speed, which may increase the overall completion time of the application. Regarding the incompatibility of the two interests, a trade-off between makespan and energy consumption, or bi-objective optimization, still remains a challenge for energy-efficient workflow schedulers.

Given the NP-complete complexity of its general form [26], the problem of bi-objective workflow scheduling is even more complicated, since the scheduling algorithm needs to take additional considerations for frequency selections and makespan-energy trade-offs, other than task-to-processor mappings and precedence constraint satisfaction. From the experience of traditional time-efficient workflow scheduling schemes, the methodologies to solve the scheduling problem can be dichotomized into two major groups, namely heuristic and metaheuristic [4,8,9]. The heuristic-based algorithms normally have high runtime efficiency as they narrow down the search process to an extremely limited solution space by a set of efficient rule-based policies. These rules have significant effects on the results, but are not likely to be consistent for a wide range of problems. In contrast, the metaheuristic-based algorithms are less efficient because of the high computational cost generated by the incorporated combinatorial process, but they have demonstrated robust performance in various scheduling problems due to their power in searching more solution regions [8–13]. Furthermore, the metaheuristic group can be classified into two subcategories: single solution-based (e.g., tabu search, simulated annealing, and local search) and population-based (e.g., genetic algorithm and particle swarm optimization) [27]. The single solution-based method exploits more solutions along a promising trajectory from a single starting point, while the population-based method concurrently track a set of seeding schedules to explore more solution space. Each of the two methods has its own strengths and weaknesses, but have high complementarity to each other. To this end, the memetic algorithm (MA) appears to be a natural choice, but is rarely applied to the problem of energy-efficient workflow scheduling. Formally, a memetic algorithm is a population-based metaheuristic composed of an evolutionary framework and a set of local search algorithms that are activated within the generation cycle of the external framework [28].

In this article, a memetic algorithm for workflow scheduling on a DVFS-enabled HCS, namely MA-DVFS, is proposed to optimize makespan and energy consumption for executing a parallel application. Moreover, to avoid the extreme points on the Pareto front (e.g., low energy consumption with large makespan, and vice versa), as well produce a quality-guaranteed solution, MA-DVFS introduces a baseline that is also used as a seeding point during the bi-objective optimization search. The overall scheme generally involves three major phases. The first phase is task prioritizing, which has already revealed its significant effect on the quality of schedules [3,4]. Each task permutation under the precedence constraints indicates an independent portion of the solution space, which can be explored by a population-based method. The second phase is inspired by the fact that minimizing makespan usually helps with energy reduction. Thus, an earliest finish time (EFT)-based heuristic is utilized to provide a time-efficient candidate solution in the given portion. Based on this candidate, a local search method is applied with a certain probability to exploit better solutions in the third phase. To accommodate bi-objective optimization,

the improved non-dominated sorting genetic algorithm (NSGA-II) [29] is employed for the evolutionary framework. The main contributions of this article are listed below.

- A memetic algorithm for energy-efficient workflow scheduling is proposed to integrate the abilities of exploration and exploitation with a relatively low time complexity. The search process towards optimal schedules is expected to spread intentionally and deeply.
- A novel local search algorithm incorporated with a pruning technique is developed to accelerate the exploitation process. Furthermore, it is proven that launching the local search with a low probability is sufficient for a stable result.
- A baseline solution generated by a time-efficient scheduling algorithm is introduced as a good seed, as well as a direction for the evolutionary search, ensuring the bi-objective optimization to produce quality-guaranteed schedules.
- Extensive simulations are conducted to validate the proposed algorithm by comparisons with related algorithms on workflows of both randomly-generated and real-world applications. Experimental results reveal the superior performance and the high efficiency of the proposed algorithm.

The rest of this article is organized as follows: Section 2 briefly reviews related work and existing approaches. Section 3 describes the system model, the application model, and the energy model used in this article. Section 4 details the proposed algorithm, while Section 5 gives the experimental results and analyses. Conclusions and suggestions for future work are provided in Section 6.

2. Related Work

A workflow means a group of tasks and their dependencies, while workflow scheduling is the allocation of tasks to resources without violating precedence constraints. There can be multiple objectives for workflow scheduling. In this article, we concentrate on the overall completion time (makespan) and the total energy consumption while executing the workflow. In this section, we first review related work on time-efficient workflow scheduling, which has been extensively studied, then investigate recent energy-efficient workflow scheduling schemes.

2.1. Time-Efficient Workflow Scheduling

The goal of time-efficient scheduling is to minimize the makespan. Existing approaches fall into two major groups: heuristic [3–7] and metaheuristic [8–13].

The heuristic-based group can be further classified into three subcategories: list [3–5], clustering [6], and duplication scheduling [7]. They normally offer suboptimal solutions with polynomial time complexity. Among these heuristics, the majority is list scheduling, which is usually composed of two phases: a task prioritizing phase to arrange task execution sequences and a processor selection phase to designate the best processor for each individual task. For example, the heterogeneous earliest finish time (HEFT) algorithm [3] considers upward rank values as task priorities and selects processors by an earliest finish time (EFT)-based heuristic. In the predict earliest finish time (PEFT) algorithm [4], an optimistic cost table (OCT) that implements a look-ahead feature is used for task prioritization, as well as processor selection to obtain makespan improvements. Similarly, the improved PEFT (IPEFT) algorithm [5] calculates task priorities with a pessimistic cost table (PCT) and selects processors with extra considerations of the critical tasks, so that the makespan can be further optimized. However, the performance of list scheduling algorithms relies on rule-based policies. Once the task sequence is determined, the search of list scheduling is narrowed down to a quite small portion of the solution space, thus producing a local optimal solution. The other two heuristic-based subcategories, clustering [6] and duplication [7], make a strong assumption that enough processors are available for the clustering process and task duplication, respectively, which cannot be satisfied in most real-world applications. Moreover, these algorithms require higher time complexity than list scheduling.

In comparison to heuristic-based approaches, metaheuristic ones usually incorporate a combinatorial search. The quality of the solution can be improved by more iterations, at the expense of more computational cost. Recently, most well-known metaheuristics, such as the genetic algorithm (GA) [8–10], ant colony optimization (ACO) [11], particle swarm optimization (PSO) [12], and tabu search (TS) [13], have been successfully applied to workflow scheduling. For example, a novel heuristic-based hybrid genetic-variable neighborhood search (GVNS) algorithm [8] adopts the task priorities generated by HEFT and combines a genetic algorithm with a variable neighborhood search (VNS) algorithm to exploit the intrinsic structure of the solutions. The multiple priority queues genetic algorithm (MPQGA) [9] incorporates an evolutionary framework with a heuristic-based algorithm, where a genetic algorithm is used to offer task priorities and the EFT-based heuristic is employed for task-to-processor mappings. Based on that, the task scheduling algorithm using multiple priority queues and a memetic algorithm (MPQMA) [10] leverages a genetic algorithm along with a hill climbing method to assign a priority to each task while using the EFT-based heuristic for processor selections. These algorithms appear to have different encoding schemes or different evolutionary operations, but they happen to coincide in collaboration with heuristics or local search methods. This is due to the fact that the search incurs a large solution space and high computational cost, especially when the scale of the workflow grows. The hybrid scheme has revealed its effectiveness and efficiency in this research field.

2.2. Energy-Efficient Workflow Scheduling

In practice, most HCSs are built for high performance computing. Thus, time efficiency is almost always the first priority for workflow scheduling, even with additional considerations. In this case, both energy consumption and makespan should be taken into account in energy-efficient workflow scheduling. According to the outcome expectancy of the algorithms, existing approaches can be divided into two categories: namely independently [17–22] and simultaneously [23–25,30].

In the independent mode, a "two-pass" method is widely used, in which a time-efficient schedule is provided in the first pass by a separate time-efficient scheduler, such as HEFT [3], then the slack time in the schedule is reclaimed for energy reduction in the second pass. For examples, R-DVFS [17] reviews a given schedule and executes the task along the same path as uniformly as possible using the lowest available frequency. The enhanced energy-efficient scheduling (EES) [18] heuristic performs energy reduction by executing the nearby tasks on the same processor at a uniform frequency for global optimality. In the group-then-individual (GTI) algorithm [19], the working frequency level of each task would be rescaled in the group of assigned processors and then individually, to obtain global optimization. The maximum-minimum-frequency DVFS (MMF-DVFS) method [20] and the multiple voltage–frequency selection DVFS (MVFS-DVFS) method [21] leverage linear combinations of multiple frequency levels to execute tasks so that the slack time in the given schedule can be fully utilized. Furthermore, our previous work, the linear-programming DVFS (LP-DVFS) algorithm [22], gives the best combination of multiple frequencies for energy minimization through a linear-programming method. These algorithms can achieve bounded schedule qualities since the makespan has been determined by a time-efficient scheduler beforehand. Thus, makespan can be regarded as a deadline constraint during the energy optimization, which also implies that there is no room for makespan improvement.

In the simultaneous mode, makespan and energy consumption are considered at the same time, usually through a global function, in which trade-offs are made for each task in every decision episode. For examples, the energy-conscious scheduling (ECS) heuristic [23] develops a relative superiority (RS) metric to balance time and energy factors for each task-to-processor mapping, and relies on a makespan-conservative energy reduction (MCER) process for further optimization. Based on this study, a hybridization of the genetic algorithm with ECS, namely HECS [24], is proposed to explore more solution space. Furthermore, respecting that ECS is costly in CPU time, HECS develops a multi-start

approach to make the algorithm run faster. Another similar bi-objective heuristic EECS [25] is proposed to enhance ECS by developing a new global function with considerations on more factors and replacing MCER with a new global energy saving (GES) technique. The effectiveness of these algorithms heavily depends on the objective function. Meanwhile, these algorithms always incorporate a further optimization phase (e.g., MCER and GES), which is essentially a local search method and computationally expensive. On the other hand, they cannot provide quality-guaranteed schedules due to the lack of guiding information, such as a deadline constraint.

Besides the above categories, a number of research works focus on energy-efficient workflow scheduling on specific applications. For example, as the streaming jobs on Hadoop can be formulated as a DAG, two types of energy-efficient workflow scheduling heuristics are proposed for the energy efficiency extension on YARN (yet another resource negotiator) [31]. To address the energy consumption of interconnection networks, a heuristic list-based network energy-efficient workflow scheduling (NEEWS) algorithm is proposed to investigate the efficiency of the computing nodes, as well as the interconnection networks in the HCS [32].

In summary, although metaheuristics and hybrid methods have been successfully used for time-efficient workflow scheduling, few have been applied to energy-efficient scenarios. The HECS approach provides a combination of the genetic algorithm and an ECS-based method, which is CPU-expensive and lacks intentional exploitation. In this article, a hybrid scheme MA-DVFS based on the memetic algorithm is proposed to realize bi-objective optimization with high runtime efficiency.

3. Models and Problem Formulation

In this section, we describe the system, application, and energy models used in our work, then formulate the optimization problem we studied.

3.1. System Model

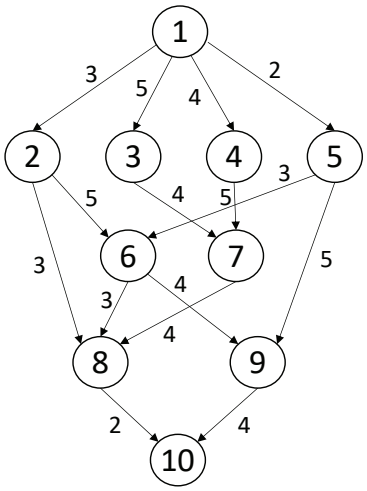
A HCS consists of a set H of m heterogeneous processors that are fully connected by a high-speed network. Each processor p_i is DVFS-enabled and has a discrete set of voltages $V_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,M_i}\}$ coupled with a discrete set of frequencies $f_i = \{f_{i,1}, f_{i,2}, \ldots, f_{i,M_i}\}$, where M_i is the number of p_i's voltage/frequency levels. The power consumption of p_i running at $V_{i,j}/f_{i,j}$ is denoted as $P_{i,j}$. The DVFS technique enables processor p_i to utilize either the maximum voltage/frequency level $V_{i,max}/f_{i,max}$ for high performance or the minimum voltage/frequency level $V_{i,min}/f_{i,min}$ for power savings. For example, the Intel i7-4770K CPU consumes approximately 64 watts during idle time, while its power consumption increases to 148 watts during peak time. Compared to the execution time of one task (at least 1 s), the switching overhead between different voltage/frequency levels (30–150 µs) can be ignored [33]. Table 1 lists the voltage/frequency pairs of four real-world processors. It can be observed that variations in both voltage and frequency occur simultaneously.

Table 1. The voltage/frequency pairs of four commodity processors.

Level	AMD Athlon-64		Intel Pentium M		AMD Opteron 2218		AMD Turion MT-34	
	V (V)	f (GHz)	V (V)	f (GHz)	V (V)	f (GHz)	V (V)	f (GHz)
1	1.5	2.0	1.484	1.4	1.30	2.6	1.20	1.8
2	1.4	1.8	1.463	1.2	1.25	2.4	1.15	1.6
3	1.3	1.6	1.308	1.0	1.20	2.2	1.10	1.4
4	1.2	1.4	1.180	0.8	1.15	2.0	1.05	1.2
5	1.1	1.2	0.956	0.6	1.10	1.8	1.00	1.0
6	1.0	1.0	-	-	1.05	1.0	0.90	0.8
7	0.9	0.8	-	-	-	-	-	-

3.2. Application Model

A large-scale application is decomposed into a set of lightweight tasks for parallel processing, which are commonly described by a directed acyclic graph (DAG) or a workflow. A workflow $G = (V, E, W, C)$ is mainly comprised of n nodes and e edges, which represent tasks and dependencies, respectively. Due to the heterogeneous architecture, the execution time of task $v_i \in V$ varies among different processors, and the computation cost of task v_i executed on processor p_j (at its maximum frequency level) is defined as $w_{i,j}$. For n processors, we have an $n \times m$ matrix W to describe the computation cost distribution of workflow G. An edge $e_{i,j} \in E$ denotes that task v_j cannot be started until it finishes data transmission from task v_i, which implies a strict partial order on task set V. The communication cost between task v_i and task v_j is defined as $c_{i,j}$. For e edges, we have a set C to describe the communication cost distribution of workflow G. Figure 1 illustrates a sample workflow with 10 tasks and its complementary matrix W of the computation cost on four processors in Table 1. For more clarity, a set of related definitions is given as follows.



Task	p_1	p_2	p_3	p_4
1	10	11	9	15
2	8	14	12	17
3	13	6	17	9
4	10	12	14	8
5	9	7	16	17
6	7	15	18	7
7	14	17	9	13
8	9	16	17	10
9	12	17	8	14
10	18	14	6	8

Figure 1. A sample workflow with 10 tasks and the complementary matrix.

Definition 1. *$v_i \prec_e v_j$: denotes the partial order between task v_i and task v_j if $\exists e_{i,j} \in E$.*

Definition 2. *$pred(v_i)$: denotes the set of all predecessor tasks of task v_i. A task with $pred(v_i) = \emptyset$ is called an entry task, namely v_{entry}.*

Definition 3. *$succ(v_i)$: denotes the set of all successor tasks of task v_i. A task with $succ(v_i) = \emptyset$ is called an exit task, namely v_{exit}.*

Definition 4. *Critical path: denotes the longest path from an entry task to an exit task, including all computation and communication costs.*

Definition 5. *CCR (communication-to-computation ratio): denotes the ratio of the average communication cost to the average computation cost. A high CCR value indicates a communication-intensive workflow, while a low CCR value implies a computation-intensive workflow.*

Definition 6. *t_i: denotes the actual execution time of task v_i in a given schedule, which can be calculated by:*

$$t_i = \frac{f_{j,max}}{f_{j,actual}} \times w_{i,j}, \tag{1}$$

where $w_{i,j}$ is task v_i's computation cost when running on processor p_j and $f_{j,max}$ and $f_{j,actual}$ are the maximum frequency level and the actual working frequency level, respectively.

Definition 7. *$AST(v_i)$: denotes the actual start time (AST) of task v_i.*

Definition 8. *$AFT(v_i)$: denotes the actual finish time (AFT) of task v_i.*

Definition 9. *$EST(v_i, p_j)$: denotes the earliest start time (EST) of task v_i when executing on processor p_j. It is given by:*

$$EST(v_i, p_j) = \max\left\{avail(p_j), \max_{v_k \in pred(v_i)}\{AFT(v_k) + c_{k,i}\}\right\}, \tag{2}$$

where $avail(p_j)$ denotes the earliest available time of processor p_j.

Definition 10. *$EFT(v_i, p_j)$: denotes the earliest finish time (EFT) of task v_i when executing on processor p_j. It is given by:*

$$EFT(v_i, p_j) = EST(v_i, p_j) + w_{i,j}. \tag{3}$$

Definition 11. *makespan: denotes the overall completion time of a workflow, which is determined by the actual finish time of the last exit task. It can be represented by:*

$$makespan = \max_{succ(v_i)=\emptyset}\left\{AFT(v_i)\right\}. \tag{4}$$

3.3. Energy Model

During the whole execution period of a workflow, each individual processor in the HCS is either activated to run tasks or in the idle state for receiving data. Therefore, the overall energy consumption is comprised of two parts: the active part and the idle part [23], given by:

$$E_{total} = E_{active} + E_{idle}. \tag{5}$$

The active part E_{active} is contributed by the execution of all tasks, which can be calculated by:

$$E_{active} = \sum_{i=1}^{n}\left(P_i \times t_i\right), \tag{6}$$

where P_i is the power consumption for executing task v_i. The idle part E_{idle} is consumed by processors in the idle state before the completion of all tasks, which can be calculated by:

$$E_{idle} = \sum_{j=1}^{m}\left(P_{j,ilde} \times t_{j,idle}\right), \tag{7}$$

where $P_{j,idle}$ is the power consumption of processor p_j in the idle state and $t_{j,idle}$ is the idle time of processor p_i before the application completes.

3.4. Problem Formulation

In this article, we consider static scheduling in which task dependencies and the cost distribution are given beforehand. Based on the above models, the problem of energy-efficient workflow scheduling can be described as: Given a workflow G, we need to search for a schedule π that includes task prioritization, processor assignment, and frequency selection, so that the overall completion time *makespan* and the total energy consump-

tion E_{total} can be minimized simultaneously. It can be classified as a typical bi-objective optimization problem, which can be formulated by:

$$\begin{cases} \min \left\{ makespan(\pi), \quad E_{total}(\pi) \right\} \\ subject\ to: \\ (1) \qquad AFT(v_i) + c_{i,j} \leq AST(v_j), \quad \forall v_i \prec_e v_j,\ v_i, v_j \in V \\ (2) \qquad \sum_{j=1}^{m} \sum_{k=1}^{M_j} x_{i,j,k} = 1, \quad \forall i \in \{1, 2, \ldots, n\}, \forall x_{i,j,k} \in \pi \end{cases} . \tag{8}$$

where Constraint (1) ensures that the precedence constraints in the workflow are not violated and $x_{i,j,k} = 1$ represents that task v_i is scheduled to run on processor p_j at frequency level $f_{j,k}$, otherwise $x_{i,j,k} = 0$. Consequently, Constraint (2) implies that a task is executed only once by using one frequency level.

4. The MA-DVFS Algorithm

This section presents the detailed description of the proposed algorithm, including the overall algorithm flow and the concepts of the memetic algorithm.

4.1. The Algorithm Flow

The main algorithm flowchart of the proposed MA-DVFS is demonstrated in Figure 2, which can be divided into three parts: First, a multi-objective evolutionary algorithm, e.g., NSGA-II, is employed as the main framework to explore a new solution space, as well as to evaluate individuals in each generation. Second, an EFT-based heuristic is utilized to point out a time-efficient schedule in the specified portion of the solution space. Third, a local search method integrated with a pruning technique is adopted with a low probability to exploit energy-efficient schedules in the given region. After several rounds of repeats, the best solution is reported.

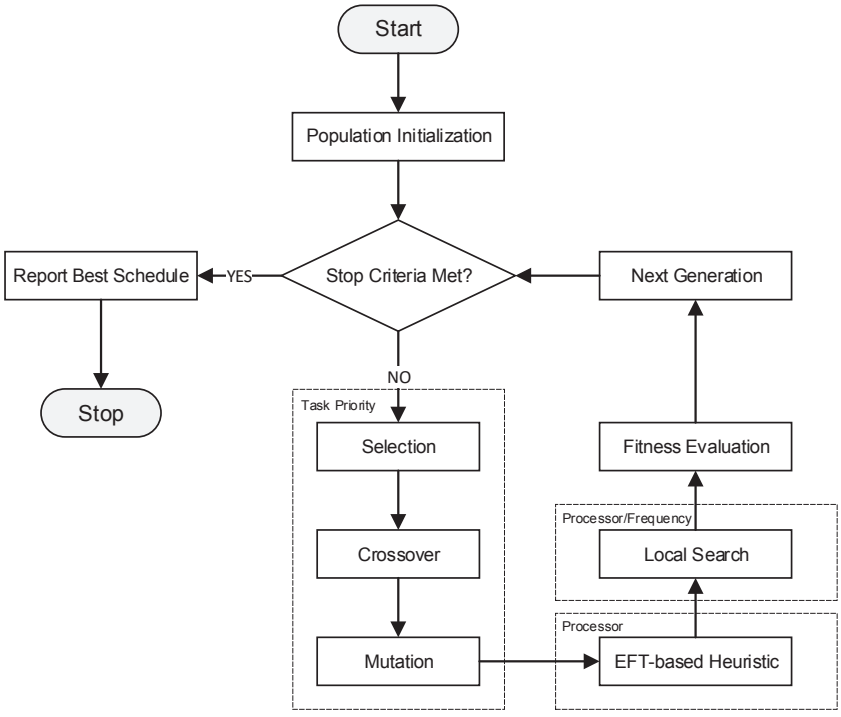


Figure 2. The algorithm flowchart of the memetic algorithm (MA)-dynamic voltage frequency scaling (DVFS). EFT, earliest finish time.

From the aspect of the final schedule, the task priority queue is produced by the evolutionary algorithm, which indicates a feasible region in the solution space, as shown in Figure 3. The processor/frequency selections are determined by the EFT-based heuristic together with the local search method. The overall algorithm leverages a combination of population-based, heuristic-based, and local search methods to coordinate exploration and exploitation for bi-objective workflow scheduling.

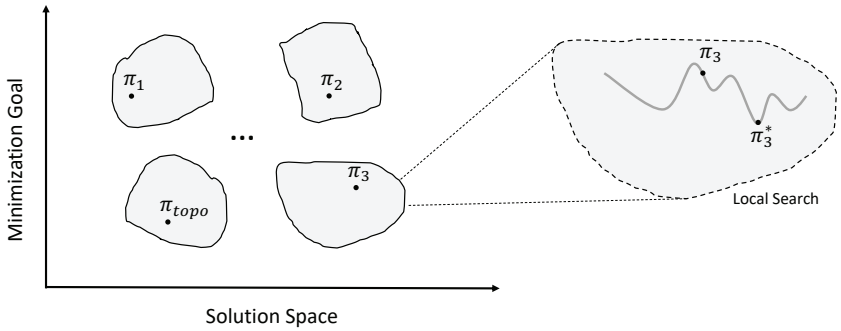


Figure 3. Feasible regions indicated by different task priority queues.

4.2. Encoding Scheme and Search Space Analysis

Encoding of individuals is one of the most fundamental and important steps in an evolutionary algorithm. In the scenario of energy-efficient workflow scheduling, each solution consists of three segments, including task, processor, and frequency, as illustrated in Figure 4. The task segment contains a permutation of integers $1, 2, \ldots, n$, representing a valid task priority queue, which must be one of the topological orders of the workflow [9]. Each gene in the processor segment can be an arbitrary index ranging from one to m, while each gene in the frequency segment is restricted to the corresponding interval from one to M_i of the candidate processor p_i. Hence, each column of the individual represents one element of the schedule. For example, the first column in Figure 4 denotes $x_{1,3,1} = 1$, which represents that task v_1 is scheduled to run on processor p_3 using frequency $f_{3,1}$.

task	1	2	4	5	3	6	7	9	8	10
processor	3	1	4	2	2	1	3	1	1	3
frequency	1	4	2	2	3	1	1	4	1	2

primary

supplementary

Figure 4. Encoding of a solution.

In fact, the encoding process reveals a huge search space. Task prioritizing has at most $n!$ possibilities when all tasks are independent. Moreover, there can be M^n possible assignments in the processor/frequency selection phase, where $M = \sum_{j=1}^{m} M_j$ represents all frequency levels in the HCS. Thus, the search space of the energy-efficient workflow scheduling problem is in the order of $O(n! \times M^n)$, which challenges the search ability and efficiency of conventional evolutionary algorithms.

4.3. Population Initialization

The evolutionary algorithm starts from an initial population, the quality of which is critical for the search process and the final result. The initial population consists of *popSize* individuals, in each of which the task segment can be randomly generated under precedence constraints while the rest of the segments can be filled by an EFT-based heuristic and the local search method described later. Specifically, the task segments in the initial population are chosen from the set of topological orders for diversification. With a good uniform coverage, the individuals can be well spread to cover the whole feasible solution space [9], as demonstrated in Figure 3. However, the quality of the initial population

cannot be guaranteed if it is generated in a totally random manner. In this case, a good seeding schedule can be introduced to improve the population quality and convergence speed. Meanwhile, it can be utilized as a reference point in the solution space to guide the searching process.

Based on the above idea, the process of population initialization is depicted in Algorithm 1. First, a seeding individual π_0 is generated by the HEFT algorithm and added into the initial population *pop* (Line 1). Then, a task priority queue is randomly chosen from the set of topological orders with a tabu list to avoid duplication (Line 3). Each task in the queue is allocated to a processor by an EFT-based heuristic to generate a schedule π_i (Line 4). After $popSize - 1$ rounds of repeats, the rest of the individuals of the initial population are finally generated.

Algorithm 1. Population initialization.

Input: G, H, $popSize$

Output: pop

1: Generate a seeding individual π_0 by HEFT and add it to pop
2: **for** $i = 1$ to $popSize - 1$ **do**
3: Randomly generate a task priority queue by topological sorting with a tabu list
4: Allocate tasks to processors by the EFT-based heuristic to generate a schedule π_i
5: Add π_i to pop
6: **end for**
7: **return** pop

4.4. Fitness Evaluation and Pareto Archive

Fitness in an evolutionary algorithm represents how close a given solution is to the optimum solution. In single objective optimization, a fitness function (also known as the evaluation function) is defined to evaluate a solution. In time-efficient workflow scheduling, *makespan* can be used for fitness evaluation, where a smaller *makespan* implies a better fitness. In the bi-objective scenario, a fitness function also can be defined by normalization, but the coefficients of the fitness function need to be tested and optimized. Instead, MA-DVFS applies the non-dominated sorting of NSGA-II to evaluate the fitness of each solution. The non-dominated sorting aims to divide a solution set into a number of disjoint ranks, by means of comparing the values of the same objective. The non-dominated comparison operator is defined as follows.

Definition 12. *$\pi_a \prec \pi_b$: π_a is said to be better than π_b, if and only if $\forall i \in \{1,2\}, f_i(\pi_a) \leq f_i(\pi_b)$ and at least $\exists j \in \{1,2\}, f_j(\pi_a) \leq f_j(\pi_b)$, where f_1, f_2 represent makespan and E_{total}, alternatively.*

After non-dominated comparison, solutions of a smaller rank are better than those of a larger rank, and solutions of the same rank are viewed equally important. Solutions of the smallest rank comprise a Pareto set, which is the ultimate goal for bi-objective optimization.

In order to obtain quality-guaranteed schedules, as well as to accelerate the convergence speed, we introduce a Pareto archive to store and maintain schedules that are better than the seeding solution during each cycle of the generation. The population evaluation process is shown as Algorithm 2. First, the current population *pop* is evaluated by the non-dominated sorting (Line 1). Then, schedules better than the seeding solution π_0 are regarded as good candidates and stored in pop'' (Lines 2–7). Finally, each schedule in pop'' is added to *paretoArchive* if it did not already exist (Line 8). Note that when updating *paretoArchive*, the non-dominated sorting order is always maintained simultaneously. In this case, if the size of *paretoArchive* is limited, solutions in the tail can be dropped.

Algorithm 2. Population evaluation.

Input: π_0, pop, $paretoArchive$

Output: pop', $paretoArchive$

1: Apply non-dominated sorting in NSGA-II to pop to generate pop'
2: $pop'' \leftarrow \emptyset$
3: **for** each individual $\pi_i \in pop'$ **do**
4: **if** $\pi_i \prec \pi_0$ **then**
5: Add π_i to pop''
6: **end if**
7: **end for**
8: Update $paretoArchive$ with pop'' to obtain a new $paretoArchive'$
9: **return** pop' and $paretoArchive'$

4.5. Evolutionary Operations

To produce offspring for the next generation, evolutionary algorithms have to use the current population to create the children by a series of evolutionary operations, including crossover, mutation, and selection. In MA-DVFS, the evolutionary framework is mainly used to explore diverse task priority queues. In this case, the above operations are applied to the task segment, which is commonly used in population-based methods for time-efficient workflow scheduling [9,24].

4.5.1. Crossover Operator

Crossover is the process of emulating generation alternation and producing offspring from selected parents. As previously mentioned, the crossover operator is applied to the task segment of the schedule. In this case, the operator should be responsible for producing valid offspring, which means new task priority queues are also topological orders of the workflow. To this end, we use a topological order preserving heuristic to generate offspring, as shown in Algorithm 3. First, two individuals are selected from the parent population, named as $Parent1$ and $Parent2$, on which a crossover point is randomly selected (Line 2). Then, for $Offspring1$, genes of the left part are cloned from $Parent2$ (Line 3) while the rest are inherited from $Parent2$ in its original topological order (Lines 4–8), and $Offspring2$ is generated in the same way (Lines 9–14). Figure 5 demonstrates a sample process of the crossover operator. We leverage a single crossover point in MA-DVFS since it has been proven to be topological order preserving [9] and is simple yet sufficient for task priority exploration.

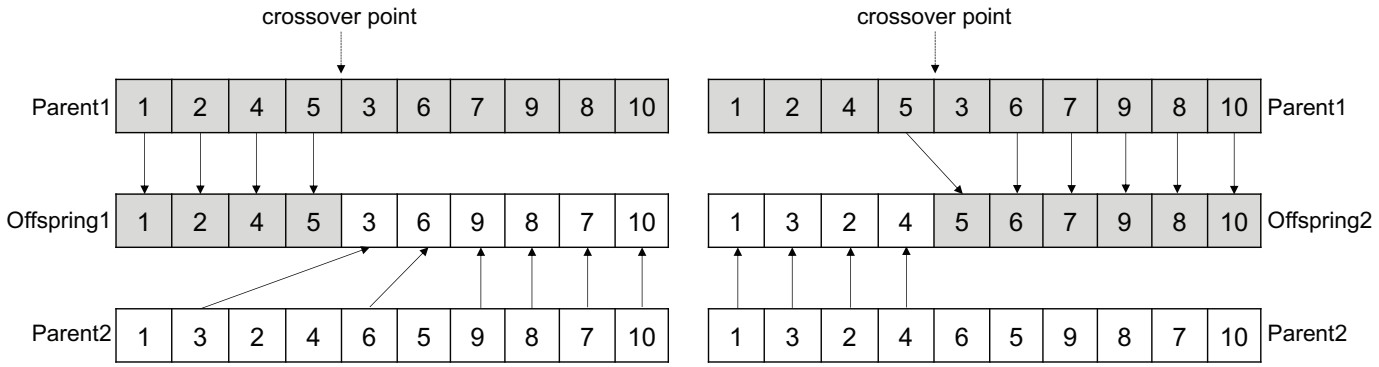


Figure 5. Crossover operation.

Algorithm 3. Crossover operator.

Input: $Parent1$, $Parent2$

Output: $Offspring1$, $Offspring2$

$Offspring \leftarrow \emptyset$, $Offspring2 \leftarrow \emptyset$
Choose a random crossover point $i \in (1, n)$
$Offspring1[1..i] \leftarrow Parent1[1..i]$
for each j from 1 to n **do**
    if $Parent2[j]$ does not exist in $Offspring1$ yet **then**
        Append $Parent2[j]$ to the tail of $Offspring1$
    end if
end for
$Offspring2[1..i] \leftarrow Parent2[1..i]$
for each j from 1 to n **do**
    if $Parent1[j]$ does not exist in $Offspring2$ yet **then**
        Append $Parent1[j]$ to the tail of $Offspring2$
    end if
end for
return $Offspring1$ and $Offspring2$

4.5.2. Mutation Operator

Mutation is analogous to biological mutation, which is used to maintain genetic diversity. The mutation operator changes a gene with a certain probability, thus helping the search algorithm escape from local optimal solutions. For a task priority queue, mutation should be performed without violating the precedence constraints. To this end, once a mutation point is provided, the closest predecessor and successor can be determined according to Definitions 2 and 3. Only positions between the predecessor and the successor (except the mutation point) should be considered as a new position, as shown in Figure 6. If the new position is selected, the two genes are interchanged to produce a new individual.

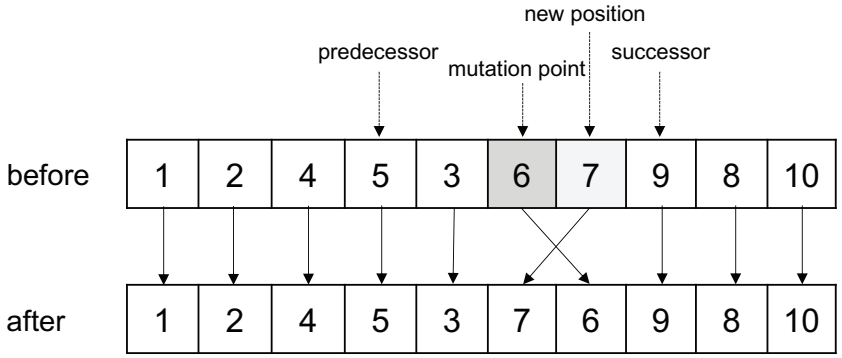


Figure 6. Mutation operation.

4.5.3. Selection Operator

The selection operator is used to select individuals from the parent population for breeding the next generation. The primary objective of the selection operator is to emphasize the good individuals and eliminate bad ones; thus, the population should be evaluated, normally by a fitness function, which is not used in MA-DVFS. Instead, we

adopt a tournament [34] strategy for selecting candidates. The measurement of fitness of the population is implemented by the non-dominated sorting as mentioned in Section 4.4; thus, individuals having the best fitness are then selected.

4.6. Local Search

Since a candidate schedule, which indicates a feasible region in the solution space, has been determined by the population-based metaheuristic and the EFT-based heuristic, the goal of the following step is to find the best solution in this region. However, it is a classic combinatorial optimization problem, which has M^n possible solutions. We leverage a local search technique to tackle this problem.

Specifically, a hill climbing method is employed to get the local optima. Hill climbing is one of the local optimization methods, which starts from a given solution and seeks an improvement by incrementally modifying its configurations. The algorithm of local search with pruning is described in Algorithm 4. For a given schedule π (Line 1), each processor/frequency pair can be examined in the order of task priorities in π (Lines 2–17). Furthermore, two pruning steps, which consider the factors of makespan (Lines 6–8) and energy (Lines 9–11), respectively, are incorporated into each iteration to narrow down the searching space, thus reducing the runtime cost of the algorithm significantly. Since the best solution is recorded during the searching process (Lines 12–14), a local optimum is finally obtained (Line 18). For example, when the algorithm is applied to the workflow shown in Figure 7 and the processors listed in Table 1, it can improve the HEFT on energy saving by 15.9% (no makespan improvement). Meanwhile, the incorporated pruning technique contributes to the acceleration of the local search process by 69.2%.

Algorithm 4. Local search with pruning.

Input: a solution π

Output: a local optimal solution π^*

1: $\pi^* \leftarrow \pi$
2: **for** each v_i in π^* from left to right **do**
3: **for** each $p_j \in H$ **do**
4: **for** each $f_{j,k}$ of p_j **do**
5: Reallocate v_i to p_j at $f_{j,k}$ to formulate a new solution π'
6: **if** $(f_{j,k} == f_{j,max})$ && $(makespan(x') > makespan(x^*))$ **then**
7: **break**
8: **end if**
9: **if** $(v_i \notin V_{CP})$ && $(f_{j,k} == f_{j,min})$ && $(E_{total}(\pi') > E_{total}(\pi^*))$ **then**
10: **break**
11: **end if**
12: **if** $\pi' \prec \pi^*$ **then**
13: $\pi^* \leftarrow \pi'$
14: **end if**
15: **end for**
16: **end for**
17: **end for**
18: **return** π^*

4.7. The Overall Algorithm

The overall algorithm is described by Algorithm 5. The initial population *pop* is generated by Algorithm 1 and evaluated in the next step to initialize the *paretoArchive* (Line 2). So far, the seeding solution provided by HEFT has been included in *paretoArchive* to ensure the quality of the final result. Then, the algorithm falls into g_{max} rounds of iterations as a population-based method algorithm always does. In each generation, a series of evolutionary operations, including selection, crossover, and mutation, as described above, is applied to the current individuals to generate a new population pop' (Line 5). The EFT-based heuristic used in HEFT is utilized to allocate tasks to processors without considering frequency assignments (Lines 6–8). The underlying intuition of this strategy is that reducing the total execution time usually results in a more energy-efficient schedule, since the occupancy of a HCS consumes energy every second. After that, the new population pop' is evaluated while the *paretoArchive* is updated accordingly by Algorithm 2 (Line 9). Then, the local search method as defined in Algorithm 4 is launched with a certain sampling rate to a random subset of the population (Lines 10–14). Finally, the routine replacement strategy in NSGA-II is applied to form the new generation (Line 15). The evolution process is terminated when the maximum number of generations is reached. The final solution can be reported by popping the first item of *paretoArchive* (Line 17).

Algorithm 5. The MA-DVFS algorithm.

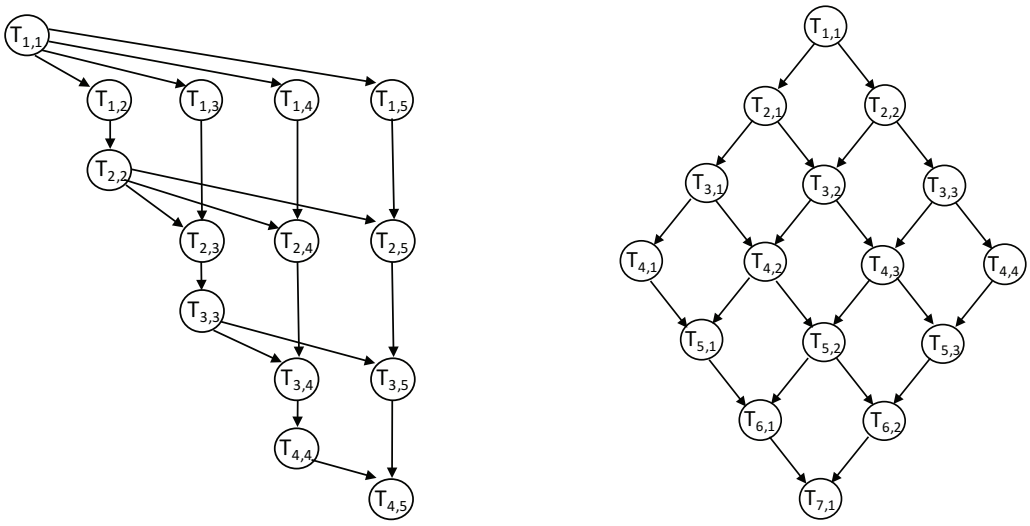
Input: G, H, g_{max}

Output: π^*

1: Call Algorithm 1 to generate initial population *pop*
2: Call Algorithm 2 to evaluate *pop* and initialize *paretoArchive*
3: $g \leftarrow 0$
4: **while** $g{+}{+} < g_{max}$ **do**
5: Selection, crossover, and mutation to generate a new population pop'
6: **for** each π_i in pop' **do**
7: Allocate tasks in π_i to processors among H by the EFT-based heuristic
8: **end for**
9: Call Algorithm 2 to evaluate *pop*‘ and update *paretoArchive*
10: **if** Sampling condition is met **then**
11: Randomly select an individual π from pop' and *paretoArchive*
12: Call Algorithm 4 to do local search to produce π'
13: Update *paretoArchive* with π'
14: **end if**
15: Combine and sort *pop* and pop' to select individuals for the next generation *pop*
16: **end while**
17: $\pi^* \leftarrow$ the first population in *paretoArchive*
18: **return** π^*

According to the procedures of the algorithm, MA-DVFS is able to provide a quality-guaranteed schedule, benefiting from the seeding schedule and the Pareto archive. Furthermore, with the aid of non-dominated sorting in NSGA-II, MA-DVFS keeps approaching the Pareto front to obtain bi-objective optimization. The time complexity of MA-DVFS is analyzed as follows. Each individual needs to execute evolutionary operations, the time

complexity of which is $O(g \times p \times e \times m)$, where g is the number of generations, p is the population size, e is the edge number of the workflow, and m is the number of processors. Taking the local search into account, the overall time complexity of the algorithm is in the order of $O(g \times (p \times e \times m + (e + n) \times M))$.



(**a**) Gaussian ($s = 5$) (**b**) Laplace ($v = 4$)

Figure 7. Sample workflow of two real-world applications.

5. Evaluation

In this section, we first present the experimental settings and comparison metrics, then validate the effectiveness of the proposed algorithm in terms of scheduling performance, Pareto dominance, and runtime efficiency. In addition, we examine the impact of the local search possibility to derive a reasonable sampling rate.

5.1. Experimental Setting

We considered experiments on two sets of workflows: randomly-generated application and real-world application. For the randomly-generated application, we used a synthetic workflow generation program and followed the same parameters in related work [3,4]. For real-world applications, two well-known parallel applications, including Gaussian elimination and the Laplace equation solver were considered as the benchmark workflows, as shown in Figure 7. Rather than the number of tasks, two new parameters, namely matrix size s and number of variables v, are used to represent the scales of the Gaussian workflow and the Laplacian workflow, while the total number of nodes in the workflow of the two applications is equal to $\frac{s^2+s-2}{2}$ and v^2, respectively. These applications retain the same graph structures once the scales of the workflow are fixed, but they can be various with different CCRs. Besides, the experimental HCSs are comprised of processors listed in Table 1, and different numbers of processors are considered to simulate different concurrent environments. Table 2 summarizes the key parameters used in the experiments.

Table 2. Experimental parameters. CCR, communication-to-computation ratio.

Parameter	Value
Number of tasks for random, n	$\{20, 40, 60, 80, 100, 200\}$
Matrix size for Gaussian, s	$\{5, 8, 11, 14, 17, 20\}$
Number of variables for Laplacian, v	$\{5, 7, 9, 11, 13, 15\}$
Number of processors, m	$\{4, 8, 16, 32\}$
CCR	$\{0.1, 0.5, 1, 2, 5, 10\}$

In order to verify the effectiveness of the proposed MA-DVFS algorithm, we compared it with four related algorithms, including HEFT [3], LP-DVFS [22], local search, and HECS [24]. Among these algorithms, HEFT is the most cited time-efficient scheduler and considered as the baseline for comparisons; LP-DVFS is a single objective workflow scheduler, which has been proven to achieve the upper bound of energy savings; local search is the method given in Algorithm 4, which is able to realize bi-objective optimization independently; HECS is the most similar work, which is also a hybrid evolutionary algorithm. Table 3 lists the parameters used in the two evolutionary algorithms, which follow the setup in HECS. All simulations were programmed in C and performed on a Dell PowerEdge R730 server with an Intel Xeon E5-2620 @ 2.10 GHz CPU and 32GB RAM.

Table 3. Evolutionary parameters.

Parameter	HECS	MA-DVFS
Population size	20	20
Number of generations	200	200
Crossover rate	1	1
Mutation rate	0.35	0.35
Sampling rate	-	0.1

For comparative purposes, the two optimization objectives, *makespan* and E_{total}, are normalized to their low bounds: the total completion time and energy consumption of the tasks along the critical path (CP) without considerations of the communication costs, which are commonly referred to as the schedule length ratio (SLR) and energy consumption ratio (ECR) [22–25], defined as follows.

$$SLR = \frac{makespan}{\sum_{v_i \in V_{CP}} \min_{p_j \in P}\{w_{i,j}\}}, \tag{9}$$

$$ECR = \frac{E_{total}}{\sum_{v_i \in V_{CP}} \min_{p_j \in P}\{w_{i,j}\} \times \max_{k \in M_j}\{P_{j,k}\}}, \tag{10}$$

where V_{CP} is the set of on-critical-path tasks of the workflow and $P_{j,k}$ denotes the power consumption of processor p_j running at the kth frequency level.

5.2. Scheduling Performance

We compared the performance of the four algorithms with respect to different application scenarios as listed in Table 2. For different workflow scales, we used eight processors for each type as listed in Table 1, and the CCRs were randomly selected. For different processor numbers, we used the maximum scales of three workflows, namely 200, 20, and 15 for random, Gaussian and Laplacian, respectively, and the CCRs were also randomly selected. For different CCRs, we chose the maximum scale of each workflow, as well as the maximum number of processors to simulate large-scale applications. For each parameter setting, we conducted 100 experiments; thus, we tested the performance on 2800 different scenarios.

We collected all the data from the above experiments and compared the energy-efficient schedulers in terms of makespan and energy consumption based on the baseline provided by HEFT. The overall comparison results are summarized in Table 4. It can be observed that LP-DVFS had no improvement on makespan since it performs a single objective optimization. Considering the energy consumption, LP-DVFS obtained the maximum energy savings under its problem constraints (only frequency levels are adjustable). Local search achieved bi-objective optimization, as we expected, improving the average makespan and energy savings by 0.3% and 14.5%, respectively. The most similar competitor HECS gained an average reduction of 19.2% on energy consumption at the expense of a –9.8% degradation of the makespan, which can be attributed to the lack of guided random

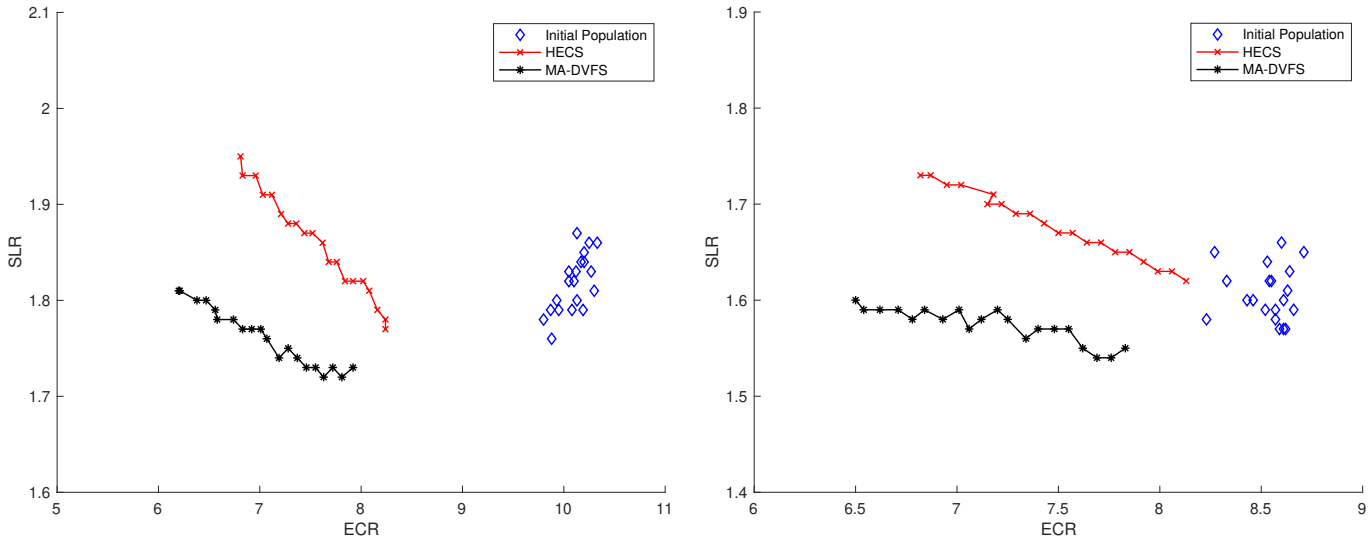
search. In terms of bi-objective optimization, MA-DVFS was more competitive with the average improvements on makespan and energy saving by 4.9% and 24.3%, which are superior to the improvements achieved by the local search algorithm. This improvement was due to the exploration ability of MA-DVFS.

Table 4. Performance comparison of four scheduling algorithms. LP, linear-programming.

Application	LP-DVFS		Local Search		HECS		MA-DVFS	
	Makespan	Energy	Makespan	Energy	Makespan	Energy	Makespan	Energy
Random	0%	9.7%	0.4%	18.2%	−11.4%	19.4%	2.2%	24.2%
Gaussian	0%	10.3%	0.3%	15.1%	−8.7%	20.7%	4.5%	23.9%
Laplacian	0%	9.4%	0.3%	10.3%	−9.3%	17.5%	7.9%	24.9%
Average	0%	9.8%	0.3%	14.5%	−9.8%	19.2%	4.9%	24.3%

5.3. Pareto Dominance

In order to demonstrate the detailed comparison of the two evolutionary algorithms, we compared their Pareto dominance of bi-objective optimization on two applications, including Gaussian and Laplacian. For the scale of the workflow, the matrix size s was set to 14 for Gaussian while the number of variables v was set to 10 for Laplacian, respectively. In both applications, the number of processors m was set to 16, and CCR was set to one. Figure 8 illustrates the Pareto fronts of HECS and MA-DVFS, while the diamond points represent the initial population. It can be seen that HECS performed well in energy saving, but failed in makespan improvement. In contrast, the Pareto front provided by MA-DVFS was closer to the best region. Therefore, by reviewing the Pareto front, MA-DVFS was able to offer a better solution.



(**a**) Gaussian (s = 14, m = 16, CCR = 1) (**b**) Laplacian (v = 10, m = 16, CCR = 1)

Figure 8. Pareto fronts of two evolutionary algorithms. SLR, schedule length ratio; ECR, energy consumption ratio.

5.4. Runtime Efficiency

Among the four algorithms, HEFT and LP-DVFS are heuristics, which have polynomial running time, and local search belongs to the single solution-based metaheuristic, which is also relatively runtime efficient. Hence, the comparison was carried out between the two population-based algorithms, namely HECS and MA-DVFS, which are more comparable. Figure 9 depicts the average time cost of the two algorithms to process each generation in the above experiment. Requiring executing the ECS search for each individual, the HECS algorithm spends much more time in each generation, with 18.9 s and

15.9 s for Gaussian and Laplacian, respectively. Comparatively, the MA-DVFS algorithm consumes only 0.134 s and 0.071 s for the two applications, respectively. This significant improvement can be attributed to the combination of the EFT-based heuristic and the local search method with pruning. In addition, launching the local search process with a relatively low sampling rate helps to accelerate the algorithm as well.

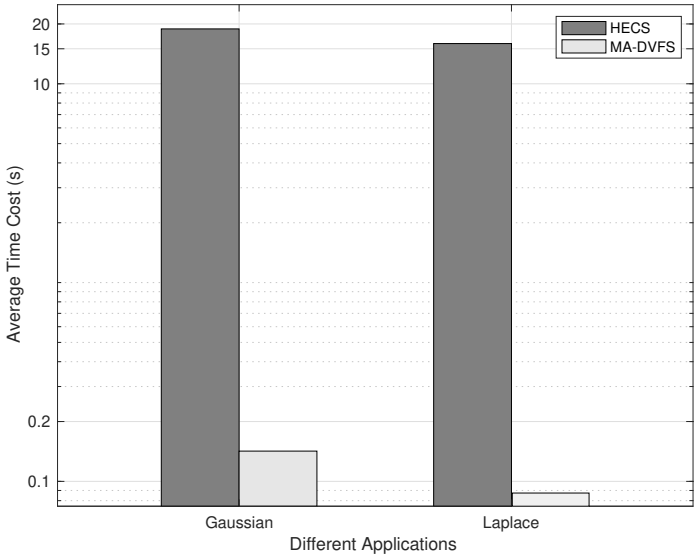
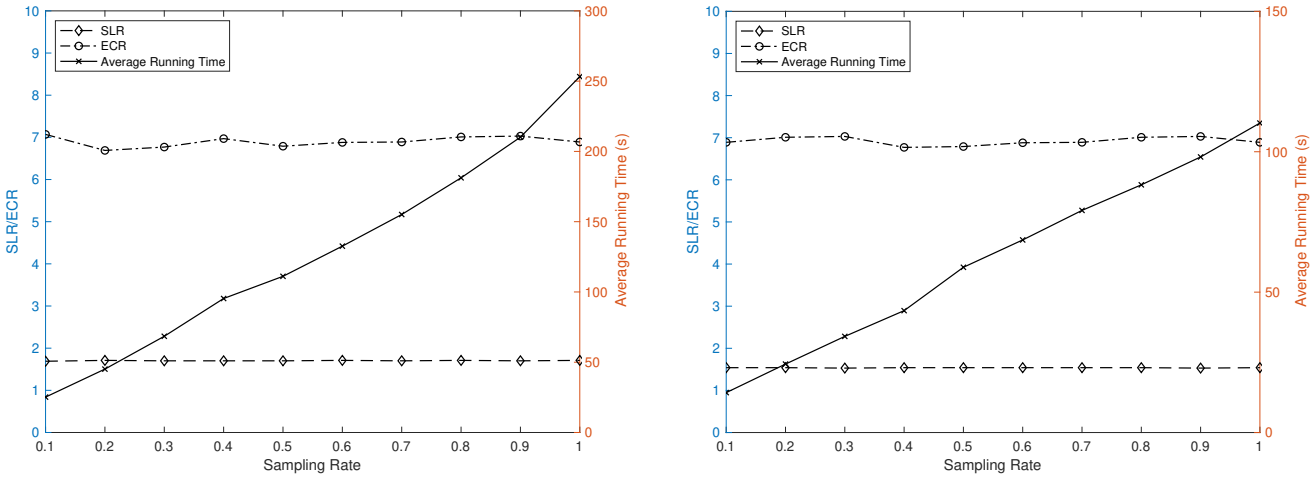


Figure 9. Average time cost for each generation.

5.5. Impact of the Local Search Possibility

As listed in Table 3, the sampling rate of local search in the above experiments was set to only 0.1. In order to validate this parameter setting, we examined the impacts of different sampling rates on the Gaussian and Laplacian applications used above. Figure 10 exhibits the impacts of different sample rates from 0.1 to one, in steps of 0.1, on SLR, ECR, and average runtime cost. It can be observed that the scheduling performance represented by SLR and ECR has rare fluctuations when the sampling rate changes, whereas the runtime cost increases sharply with the increasing sampling rate. Hence, a sampling rate as low as 0.1 is sufficient to guarantee the quality of the final result.



(**a**) Gaussian (**b**) Laplace

Figure 10. Impact of the sampling rate.

6. Conclusions

In this article, a novel memetic algorithm for energy-efficient workflow scheduling on DVFS-enabled heterogeneous computing systems, MA-DVFS, is proposed by means of incorporating NSGA-II with an EFT-based heuristic and a local search method. Although the proposed local search method was already able to provide suboptimal schedules through single solution-based exploitation, it can be enhanced by a genetic framework to explore more task priority queues. Furthermore, it has been proven that a low sampling rate of local search is sufficient to provide high-quality solutions. Experimental results demonstrate the superior performance of the proposed scheme to other related algorithms in terms of makespan and energy saving. Moreover, it has higher runtime efficiency than the population-based competitor.

In future work, the performance of the proposed algorithm can be tested on large-scale HCSs, as well as workflows extracted from other applications. Incorporating other local search methods, such as tabu search [13] and variable neighborhood search [8], can also be considered.

Author Contributions: The research for this article was undertaken by Y.Z., F.T., C.L., and Y.X.; conceptualization and investigation, Y.Z. and C.L.; software and validation, Y.Z. and Y.X.; writing, original draft preparation, Y.Z., F.T., C.L., and Y.X.; writing, review and editing, Y.Z. and F.T. All authors read and agreed to the published version of the manuscript.

Funding: This research is funded by the Natural Science Foundation of Jiangsu Province of China (Grant No. BK20190346) and the 2019 Industrial Internet Innovation and Development Project, Ministry of Industry and Information Technology, China (Grant No. 6709010003).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mittal, S.; Vetter, J. A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv.* **2015**, *47*, 1–35. [CrossRef]
2. Top500. Top 10 sites for November 2020. Available online: https://www.top500.org/lists/2020/11/ (accessed on 1 December 2020).
3. Topcuoglu, H.; Hariri, S.; Wu, M. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
4. Arabnejad, H.; Barbosa, J. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 682–694. [CrossRef]
5. Zhou, N.; Qi, D.; Wang, X.; Zheng, Z. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. *Concurr. Comput. Pract. Exper.* **2017**, *29*, e3944. [CrossRef]
6. Cirou, B.; Jeannot, E. Triplet: A clustering scheduling algorithm for heterogeneous systems. In Proceedings of the International Conference on Parallel Processing Workshops (ICPPW), Valencia, Spain, 3–7 September 2001; pp. 231–236.
7. Sinnen, O.; To, A.; Kaur, M. Contention-aware scheduling with task duplication. *J. Parallel Distrib. Comput.* **2011**, *71*, 77–86. [CrossRef]
8. Wen, Y.; Xu, H.; Yang, J. A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system. *Inf. Sci.* **2011**, *181*, 565–581. [CrossRef]
9. Xu, Y.; Li, K.; Hu, J.; Li, K. A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues. *Inf. Sci.* **2014**, *270*, 255–287. [CrossRef]
10. Keshanchi, B.; Navimipour, N. Priority-based task scheduling in the cloud systems using a memetic algorithm. *J. Circuit Syst. Comp.* **2016**, *25*, 1650119.
11. Ferrandi, F.; Lanzi, P.; Pilato, C.; Sciuto, D.; Tumeo, A. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2010**, *29*, 911–924. [CrossRef]
12. Kashan, A.; Kashan, M.; Karimiyan, S. A particle swarm optimizer for grouping problems. *Inf. Sci.* **2013**, *252*, 81–95. [CrossRef]
13. Shanmugapriya, R.; Padmavathi, S.; Shalinie, S. Contention awareness in task scheduling using tabu search. In Proceedings of the IEEE International Advanced Computing Conference (IACC), Patiala, India, 6–7 March 2009; pp. 272–277.
14. NRDC. America's Data Centers Consuming and Wasting Growing Amounts of Energy. Available online: https://www.nrdc.org/resources/americas-data-centers-consuming-and-wastinggrowing-amounts-energy (accessed on 2 December 2020).
15. Trueman, C. Why Data Centres Are the New Frontier in the Fight against Climate Change. Available online: https://www.computerworld.com/article/3431148/why-data-centres-are-the-new-frontier-in-the-fight-against-climate-change.html (accessed on 2 December 2020).
16. Barroso, L.; Hölzle, U.; Ranganathan, P. The Datacenter as a Computer: Designing Warehouse-Scale Machines. *Synth. Lect. Comput. Archit.* **2018**, *13*, 189. [CrossRef]

17. Kimura, H.; Sato, M.; Hotta, Y.; Boku, T.; Takahashi, D. Empirical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster. In Proceedings of the IEEE International Conference on Cluster Computing, Barcelona, Spain, 25–28 September 2006; pp. 1–10.
18. Su, S.; Huang, S.; Li, J. Enhanced energy-efficient scheduling for parallel tasks using partial optimal slacking. *Comput. J.* **2014**, *58*, 246–257. [CrossRef]
19. Zheng, W.; Huang, S. An adaptive deadline constrained energy-efficient scheduling heuristic for workflows in clouds. *Concurr. Comput. Pract. Exper.* **2015**, *27*, 5590–5605. [CrossRef]
20. Rizvandi, N.; Taheri, J.; Zomaya, A.; Lee, Y. Linear combinations of DVFS-enabled processor frequencies to modify the energy-aware scheduling algorithms. In Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), Melbourne, VIC, Australia, 17–20 May 2010; pp. 388–397.
21. Rizvandi, N.; Taheri, J.; Zomaya, A. Some observations on optimal frequency selection in DVFS-based energy consumption minimization. *J. Parallel Distrib. Comput.* **2011**, *71*, 1154–1164. [CrossRef]
22. Zhang, Y.; Wang, Y.; Tang, X.; Yuan, X.; Xu, Y. Energy-efficient task scheduling on heterogeneous computing systems by linear programming. *Concurr. Comput. Pract. Exper.* **2018**, *30*, e4731. [CrossRef]
23. Lee, Y.; Zomaya, A. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *22*, 1374–1381. [CrossRef]
24. Mezmaz, M.; Melab, N.; Kessaci, Y.; Lee, Y.; Talbi, E.; Zomaya, A.; Tuyttens, D. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *J. Parallel Distrib. Comput.* **2011**, *71*, 1497–1508. [CrossRef]
25. Zhou, P.; Zheng, W. An efficient bi-objective heuristic for scheduling workflows on heterogeneous DVS-enabled processors. *J. Appl. Math.* **2014**, *2014*, 370917. [CrossRef]
26. Ullman, J. Np-complete scheduling problems. *J. Comput. Syst. Sci.* **1975**, *10*, 384–393. [CrossRef]
27. Gogna, A.; Tayal, A. Metaheuristics: Review and application. *J. Exp. Theor. Artif. Intell.* **2013**, *25*, 503–526. [CrossRef]
28. Neri, F.; Cotta, C. Memetic algorithms and memetic computing optimization: A literature review. *Swarm Evol. Comput.* **2012**, *2*, 1–14. [CrossRef]
29. Deb, K.; Pratap, A.; Agawal, S.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *6*, 182–197.
30. Mahmood, A.; Khan, S.A.; Albalooshi, F.; Awwad, N. Energy-Aware Real-Time Task Scheduling in Multiprocessor Systems Using a Hybrid Genetic Algorithm. *Electronics* **2017**, *6*, 40. [CrossRef]
31. Jin, P.; Hao, X.; Wang, X.; Yue, L. Energy-efficient task scheduling for CPU-intensive streaming jobs on Hadoop. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 1298–1311. [CrossRef]
32. Tang, X.; Shi, W.; Wu, F. Interconnection network energy-aware workflow scheduling algorithm on heterogeneous systems. *IEEE Trans. Industr. Inform.* **2020**, *16*, 7637–7645. [CrossRef]
33. Mochocki, B. A unified approach to variable voltage scheduling for non-ideal DVS processors. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2014**, *13*, 260–274.
34. Blickle, T.; Thiele, L. A Comparison of Selection Schemes Used in Evolutionary Algorithms. *Evol. Comput.* **1996**, *4*, 361–394. [CrossRef]