J. Parallel Distrib. Comput. 117 (2018) 63–72

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

# A multi-model estimation of distribution algorithm for energy efficient scheduling under cloud computing system

Chu-ge Wu, Ling Wang *

*Department of Automation, Tsinghua University, Beijing, 100084, PR China*

## HIGHLIGHTS

- An estimation of distribution algorithm (EDA) scheme is used to solve energy efficient scheduling problem.
- Two probability models of EDA are adapted for the multi-objective problem.
- The operators which could optimize makespan and energy consumption are designed based on the problem characteristics.
- A diversity improvement operator is designed to optimize the archive set.
- A large amount of numerical tests is implemented to verify the efficiency of the proposed algorithm.

## ARTICLE INFO

*Article history:*
Received 25 July 2017
Received in revised form 30 January 2018
Accepted 17 February 2018
Available online 27 February 2018

*Keywords:*
Task graph scheduling
Energy efficient scheduling
Cloud computing
Estimation of distribution algorithm
Precedence-constrained parallel application

## ABSTRACT

How to manage the applications under computing systems such as a cloud computing system in a more efficient way is a focus problem. The primary performance goal is to reduce the execution time (makespan) of the application. As the need to cloud computing grows, the environmental influence of data centers attracts much attention. This paper aims at the scheduling of the precedence-constrained parallel application to minimize time and energy consumption efficiently. A multi-model estimation of distribution (mEDA) algorithm is adopted to determine both task processing permutation and voltage supply levels (VSLs). Specific operators to decrease execution time and energy consumption are designed. An improvement operator is also designed to enhance the diversity of the non-dominated solutions. The proposed algorithm is compared with the standard heuristic methods and a parallel bi-objective genetic algorithm (bGA). The comparative results show the Pareto solution set by the proposed algorithm is able to dominate a large proportion of those solutions by both the heuristic methods and the bGA.



## 1. Introduction

As a computing environment, cloud computing has attracted much attention during recent years and has been applied to solve a variety of problems from consumer, technology and business fields. Cloud computing offers an appropriate platform to manage large amount of data. Sufficient resources such as computing infrastructure, platform and software application are provided as services with the "pay-as-you-go" model in the cloud computing architecture. The customers only need to pay for the necessary resources and services and do not have to afford the peak-provisioned IT resources.

Meanwhile, as the cloud computing services grow rapidly, the energy consumption of the cloud computing data center increases quickly. Nature resources defense council (NRDC) [22] reported that in 2013 U.S. data center consumed an estimated 91 billion kilowatt-hours of electricity and by 2020 it will increase to roughly 140 billion. The way to improve resource efficiency and minimize the environmental impact has become a focus problem for cloud computing. Much effort has been made to improve the present situation. Anton et al. [2] defined a green cloud architecture and reviewed the energy efficient allocation of the data center resource on each step. The scheduling techniques on dynamic voltage/frequency scaling (DVFS), thermal management and asymmetric multicore designs were surveyed in [32], where DVFS is the hardware basis for dynamically adjusting voltage supply levels (VSLs). A survey on energy efficient resource allocation techniques was provided in [8], where different resources (servers, storage disks and networks) were considered. Thus, it can be concluded that energy efficient resource allocation and scheduling problems under a cloud computing system are very important.

Essentially, the cloud resource allocation and scheduling problem is a large scale optimization problem resulted from multiple users with different requirements and objectives [7]. The problem

* Corresponding author.
*E-mail addresses:* wucg15@mails.tsinghua.edu.cn (C.-g. Wu), wangling@mail.tsinghua.edu.cn (L. Wang).

https://doi.org/10.1016/j.jpdc.2018.02.009
0743-7315/

is NP-hard, and thus it cannot obtain the optimal solution in limited time. During the past two decades, many difficult scheduling, mapping and load balance problems have been successfully solved by evolutionary algorithms. The estimation of distribution algorithm (EDA) was adopted to solve complex scheduling problems [26]. The EDA is a population-based evolutionary algorithm, which describes the distribution of the promising solutions in the search space with a probability model and generates new solutions by sampling the model. Inspired by the successful applications of the EDA, especially in the field of scheduling, the EDA will be adopted to solve the precedence-constrained parallel application scheduling problem in this paper for the minimization of both makespan and energy consumption.

The reminder of the paper is organized as follows: Section 2 reviews the related work on the problem. Section 3 provides the description and the mathematical model for scheduling problem. Then, an algorithm named multi-model EDA (mEDA) to solve the problem is proposed in Section 4. In Section 5, numerical comparisons are given to demonstrate the effectiveness of the algorithm. Finally, the paper is ended with some conclusions and future work in Section 6.

## 2. Related work

In this section, the existing algorithms to solve the precedence-constrained parallel application scheduling problems are reviewed, including the single-objective and multi-objective optimization algorithms.

### 2.1. Single-objective DAG scheduling problem

The precedence-constrained parallel application can be represented by Directed Acyclic Graph (DAG). To assign and schedule DAG tasks onto a cloud computing system can be modeled as a task graph allocation on multiprocessors. DAG scheduling with unit processing time and no communication has been proved to be NP-hard problem [25]. Kwok and Ahmad [11] provided a detailed taxonomy of DAG scheduling problems. As DAG scheduling problem is NP-hard, the existing algorithms focus on heuristic methods and evolutionary algorithms. Selvi and Manimegalai [20] and Topcuoglu et al. [24] summarized the heuristic methods into three categories: list scheduling heuristics, clustering heuristics and task duplication. List scheduling orders the tasks into a list with a certain priority and schedules the tasks onto the processors in order. Heterogeneous earliest finish time (*HEFT*) proposed by Topcuoglu et al. [24] is considered as an efficient list scheduling heuristic, where the tasks are scheduled in descending order of bottom level (b-level) values. The processor assignment scheme is earliest finishing time first (EFTF) method with an insertion based policy. Clustering heuristics cluster the tasks into many categories and merge some categories until the number of categories equals the number of processors. As for task duplication, such as duplication based bottom-up scheduling algorithm (*DBUS*) [4], it focuses on processing some tasks repeatedly on different processors, and thus the communication time on data transfer can be decreased. During recent years, some evolutionary algorithms are adopted to solve DAG scheduling problem and perform well, including genetic algorithm (GA) [29] ant colony optimization (ACO) [6], and chemical reaction optimization (CRO) [28].

### 2.2. Energy efficient DAG scheduling problem

For the energy efficient DAG scheduling problem, it is a multi-objective optimization problem since energy objective should be considered as well. The existing work mainly considered the optimization of both makespan and energy consumption. Lee et al. [12] described a detailed operating condition including system, application, energy and scheduling models. With the models, scheduling algorithms including two heuristics, energy-conscious scheduling (*ECS*) and *ECS* + *idle* were presented, where *ECS* defines a relative superiority metric (RS) to compare two processor–VSL combinations. The processor–VSL combination with the biggest RS is chosen for a certain task. In such a way, a trade-off can be made between time performance and energy consumption. Then, the processor–VSL combination is determined according to task processing order. In addition, a makespan-conservative energy reduction technique (MCER) is incorporated to reduce energy consumption without increasing makespan. As for *ECS* + *idle*, it modifies RS calculation and MCER to incorporate indirect energy consumptions.

As the population-based algorithms, evolutionary algorithms and machine learning algorithms are very suitable to solve the multi-objective optimization problems. Mezmaz et al. [15] proposed a parallel bi-objective genetic algorithm (bGA) for the energy-aware scheduling problem, which is a hybridization of multi-objective GA and *ECS*. In bGA, GA consists of mutation, crossover and migration operators to provide tasks processing permutation, while *ECS* builds the processor assignment array and voltage level array for the solution. Because *ECS* costs more CPU time than GA, a multi-start approach is adopted to make it faster. In [30], a multi-objective discrete particle swarm optimization algorithm combined with DVFS technique (DVFS-MODPSO) is developed by Yassa et al. In [3], cat swarm optimization algorithm (CSO) was embedded in the energy efficient partitioning and scheduling algorithm developed by Bousselmi. In [18], Ranjbari and Torkestani applied a learning automata-based algorithm to solve the consolidate virtual machines and to decrease the energy consumption.

In addition, constraint optimization technique is also used to solve the problem. Many techniques are proposed under a set of scheduling solution and optimize energy objective without increasing a given makespan. Zhang et al. [31] turned the topological structure of tasks into a runtime DAG. Thus, the minimization of energy consumption under constraints is transferred into a linear programming problem. In [10], slack reclamation was adopted to make scheduling more energy efficient. Hu et al. [9] provided a slack allocation algorithm to downscale the task processing frequency and increase utilization of processors. Li et al. [13] proposed a deadline constrained energy aware algorithm, which provided the merging approach of sequence tasks and the merging approach of parallel tasks. A virtual machine (VM) reuse method was adopted as well in [13].

To the best of our knowledge, the EDA has not been used to solve the DAG scheduling problem. In this paper, a multi-model estimation of distribution (mEDA) algorithm will be proposed to solve the scheduling of the precedence-constrained parallel application with the criteria of minimizing both time and energy consumption. Compared with the single objective algorithms, the considerable index is added in this paper. And compared with the constraint optimization techniques and heuristic algorithms on energy efficient DAG scheduling problem, multi-objective optimization technique is adopted and a Pareto set is given by our proposed algorithm. Meanwhile, the slack reclamation and *ECS* heuristic methods are employed in the proposed algorithm. Some multi-objective evolutionary algorithms are adopted to solve this problem directly. By contrast, the scheme and model of EDA is altered according to problem characteristics in our proposed algorithm.

## 3. Problem description

The detailed system, application, energy and scheduling models for the precedence constraint parallel application scheduling

**Table 1**
Voltage–relative speed pair.

| Level | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Voltage | 1.75 | 1.4 | 1.2 | 0.9 |
| Relative speed (%) | 100 | 80 | 60 | 40 |

problem were given by Lee and Zomaya [12]. In this paper, a simplified model is considered. In particular, the heterogeneous processing system is simplified as homogeneous situation. In this section, some basic concepts of multi-objective optimization are introduced, and then the models about system, application, energy, and scheduling are presented. A mathematical model is given at last.

### 3.1. Basic concepts of multi-objective optimization problem

Multi-objective optimization problem concerns more than one objective simultaneously to find a trade-off between the conflicting objectives. Usually, it can be formulated as follows:

$$\min y = f(x) = (f_1(x), f_2(x), \ldots, f_q(x)) \tag{1}$$

where $x \in R^p$ is a $p$ dimensional decision vector, and $y \in R^q$ is the objective vector with $q$ objectives.

Usually, a solution which minimizes all the objectives might not exist. Optimal Pareto solutions are used as the results of a multi-objective optimization algorithm. Next, some widely used concepts are introduced.

**Pareto dominance:** A solution $x_1$ is considered to dominate solution $x_2$ (denoted as $x_1 \succ x_2$) if and only if (a) $f_i(x_1) \leq f_i(x_2)$ $\forall i \in \{1, 2 \ldots, q\}$, and (b) $f_i(x_1) < f_i(x_2)$ $\exists i \in \{1, 2 \ldots, q\}$.

**Optimal Pareto solution:** A solution $x$ is called a non-dominated solution or optimal Pareto solution if it is not dominated by any other solution.

**Optimal Pareto set/Pareto Front:** The solution set containing all optimal Pareto solutions is defined as the optimal Pareto set/Pareto Front.

The Optimal Pareto set obtained by a certain algorithm is defined as **non-dominated solution set/archive set** (AS). Besides, the following metrics used to evaluate multi-objective algorithms [14].

**CM**: the $C$ metric (CM) is used to compare the archive sets $AS_1$ and $AS_2$, which reflects the dominance relationship between the solutions in the two sets. CM is calculated as (2):

$$C(AS_1, AS_2) = |\{x_2 \in AS_2 \,|\exists x_1 \in AS_1, x_2 \prec x_1 \text{ or } x_2 = x_1\}| / |AS_2| \tag{2}$$

where $C(AS_1, AS_2)$ reflects the percentage of the solutions in $AS_2$ that are dominated by or the same as the solutions in $AS_1$. Usually, $C(AS_1, AS_2)$ and $C(AS_2, AS_1)$ are both calculated to compare the Pareto dominance of the archive sets. A larger $C(AS_1, AS_2)$ and a smaller $C(AS_2, AS_1)$ imply that $AS_1$ is better than $AS_2$ under the Pareto Optimal metric.

### 3.2. Models

#### 3.2.1. System and energy model

The cloud computing system is consisted of $m$ fully connected processors in a data center. The bandwidth between the processors is represented by $B$ (MB/s). Each processor is DVS-enabled which can operate with different voltage and clock frequency. In reference [12], four pairs of VSL and relative speed are provided and chosen randomly for each processor. In this work, a certain pair of VSL–relative speed pair (Table 1) is set for all processors.

The energy model for the system refers to [12]. The energy consumption of the processors is considered, which consists of the busy and idle time power consumption. The energy consumption of busy time ($E_{busy}$) is significantly influenced by supply voltage value and frequency for a certain task, which can be calculated as (3). The energy consumption of idle time ($E_{idle}$) is defined as (4).

$$E_{busy} = \sum_{i=1}^{n} \beta V_i^2 t_i / rs_i, \tag{3}$$

where $V_i$, $t_i$ and $rs_i$ are the supply voltage, processing time and relative speed of task $i$, respectively, $\beta$ is a constant parameter.

$$E_{idle} = \sum_{idle_k} \beta V_{low}^2 d_k, \tag{4}$$

where $idle_k$ denotes the $k$th idle of the scheduling solution, $V_{low}$ denotes the lowest supply voltage value, and $d_k$ denotes the length of the $k$th idle.

Then, the total energy consumption is defined as follows:

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

#### 3.2.2. Application and scheduling model

The parallel programs can be represented by DAG, which consists of $n$ nodes and a set of edges. Nodes represent the tasks, and the processing time of task $i$ is $t_i$. Edge $(i, j)$ represents the precedence constraint between tasks $i$ and $j$. $D_{i,j}$ represents the inter-task data required to be transmitted from task $i$ to $j$. If there is a precedence constraint between task $i$ and $j$, task $j$ cannot be processed before data is transmitted entirely. The weight $cm_{i,j}$ on the edge $(i, j)$ represents the communication time between two tasks which could be calculated by $D_{i,j}/B$. If the tasks with inter-task file are assigned to the same processor, the data could be read from memory and the communication time between the tasks can be ignored. The earliest start and finish time of task $j$ on different processors are defined as follows.

$$EST(j, m_j) = \begin{cases} 0, & j = 0 \\ \max\limits_i \{EFT(i, m_i) + \delta(m_i, m_j) \cdot cm_{ij}\}, & j \neq 0, \forall i, i \to j \end{cases} \tag{6}$$

$$EFT(j, m_j) = EST(j, m_j) + t_j, \tag{7}$$

where $EST(j, m_j)$ and $EFT(j, m_j)$ denote the earliest start time ($EST$) and the earliest finish time ($EFT$) of task $j$ on processor $m_j$, $\delta(m_i, m_j)$ is an indicative function. $\delta(m_i, m_j) = 0$ if $m_i = m_j$; otherwise, $\delta(m_i, m_j) = 1$.

The actual start and finish time of task $j$ on processor $i$ must be larger than $EST(j, m_j)$ and $EFT(j, m_j)$. Critical path (CP) is defined as the longest path in the direct acyclic graph without considering communication time between the tasks. The communication to computation ratio ($CCR$) is calculated as average communication time divided by average computation time, which can be denoted as $\text{avg}(cm_{i,j})/\text{avg}(t_j)$, where avg() denotes the average value. The larger $CCR$ is, more likely the application is to be communication intensive. Otherwise, it is computation intensive.

#### 3.2.3. Scheduling model, encoding and decoding method

The objective is to determine the task processing permutation, task-processor and task-VSL assignment to minimize the execution time of the application, i.e., makespan ($C_{max}$) and the total energy consumption (TEC). In this paper, the processing permutation $\pi$ and VSL assignment array are represented as following to encode a solution.

$$\begin{bmatrix} i_1 & i_2 & \cdots & i_n \\ rs_{i_1} & rs_{i_2} & \cdots & rs_{i_n} \end{bmatrix} \tag{8}$$

where $i_k$ represents task $i_k$ ($i_k = 0, 1, \ldots, n$) assigned to the $k$th position in the permutation. The first row is a permutation of the

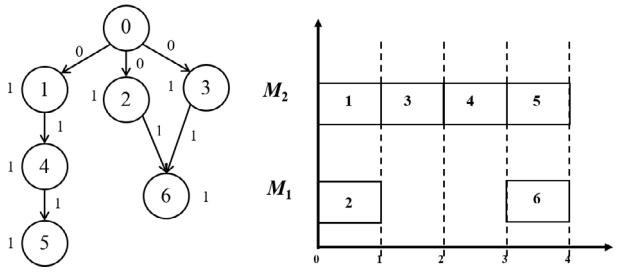


**Fig. 1.** Corresponding Gantt chart of a simple task graph.

tasks satisfying the precedence constraints, in the second row, *rs* denotes VSL assignment of a certain task.

The processor assignment is determined by earliest finish time first (EFTF), which means the tasks are allocated to the processor with the smallest completion time. When a solution is decoded, the task-processor assignment is obtained by EFTF. For example, suppose the task processing permutation of task graph in the left of Fig. 1 is (1-2-3-4-5-6) and all VSLs are set as the top level. At first, task 1 is scheduled. As the completion time on both processors is the same, it is assigned to $M_2$ randomly. Then, according to the processing permutation, task 2 is scheduled. Since there is no data between task 1 and 2, the completion time of task 2 on $M_1$ is 1 and on $M_2$ is 2. According to EFTF rule, task 2 is assigned to $M_1$. It is similar for task 3 and task 4, but for task 5 it is different. Due to the precedence constraint and communication time, task 5 cannot be processed before 4, task 5 cannot be assigned to $M_1$. So, it is assigned to $M_2$, and the communication time is ignored. The Gantt chart of the corresponding schedule is illustrated in the right of Fig. 1.

### 3.3. Mathematical model

The mathematical model modified from [27] is given as follows. The notations used in the mathematical model are explained as Table 2.

$$\min \{C_{\max}, TEC\}. \tag{9}$$

Subject to:

$$\sum_{i=1}^{m}\sum_{r=1}^{n} x_{j,i,r} = 1, \quad j = 1, \ldots, n \tag{10}$$

$$\sum_{j=1}^{n} x_{j,i,r} \le 1, \quad i = 1, 2, \ldots, m; \forall r \tag{11}$$

$$\sum_{j_1=1}^{n} x_{j_1,i,r+1} - \sum_{j_2=1}^{n} x_{j_2,i,r} \le 0, \quad i = 1, 2, \ldots, m; \forall r \tag{12}$$

$$C_{j_1} - C_{j_2} + M(2 - x_{j_1,i,r+1} - x_{j_2,i,r}) \ge t_{j_1}/rs_{j_1}, \quad i = 1, 2, \ldots, m; \forall r; \forall j_1 \ne j_2 \tag{13}$$

$$C_j \ge \sum_{i=1}^{n} x_{j,i,1} \cdot t_j/rs_j, \quad j = 1, \ldots, n \tag{14}$$

$$C_{j_1} - C_{j_2} \ge \sum_{i=1}^{m}\sum_{r=1}^{n} x_{j_1,i,r} \cdot t_{j_1}/rs_{j_1} + \delta(m_{j_1}, m_{j_2}) \cdot cm_{j_1,j_2}, \quad \forall j_1 \to j_2 \tag{15}$$

$$C_{\max} \ge C_j, \quad j = 1, \ldots, n \tag{16}$$

$$TEC = \beta\{\sum_{j=1}^{n} V_j^2 \cdot t_j/rs_j + (m \cdot C_{\max} - \sum_{j=1}^{n} t_j/rs_j) \cdot V_{\text{low}}^2\} \tag{17}$$

where (9) defines the objective functions. (10)–(12) guarantee the scheduling validity. (10) ensures that each task must be processed once and only once. (11) avoids the time conflict on processors. In (12), $\sum_{j=1}^{n} x_{j,i,r} = 1$ means a task is scheduled on the $r$th place of processor $i$, which ensures that the tasks on a certain processor are scheduled one by one. (13) and (14) limit the completion times of the tasks scheduled on the same machine. $M$ is a coefficient large enough to ensure the validity of machine processing time. When task $j_1$ and $j_2$ are scheduled adjacently on processor $i$, i.e. $x_{j_1,i,r+1} = x_{j_2,i,r} = 1$, then the completion time satisfies $C_{j_1} - C_{j_2} \ge t_{j_1}/rs_{j_1}$. If either of the tasks does not occur on the certain processor, then the inequality always holds because $M$ is a large constant. The completion time of the first task on each processor is calculated in (14). (15) is defined from (6)–(7) to ensure the precedence constraint. (16) defines makespan. According to (3)–(5), total energy consumption (TEC) is calculated as (17), where $\beta = 1$ in this paper.

Uncertainty exists anywhere in real life. Cloud computing system is a large, complicated and highly dynamic system. The research of an uncertain scheduling model is of real significance than a deterministic scheduling model. In most existing literature, only deterministic models are studied, or the expectation values are used in the models. For the uncertain cases, stochastic models with probabilistic values, fuzzy models with fuzzy membership functions, and interval models with interval values can be used [5,19], and the related algorithms like stochastic programming can be studied.

Interval model is considered to describe cloud computing scheduling model in [5]. A interval number can be represented as $\widetilde{x} = [x^-, x^+]$, where $x^-$ and $x^+$ denote low and up bound of $x$. As in real system, the processing time of task is uncertain before scheduling, and the internet bandwidth varies over time, based on the system and math model mentioned before, computation time and bandwidth can be transferred into $\widetilde{t_i}$ and $\widetilde{B}_{i,j}$. Then communication time $\widetilde{cm}_{i,j}$ can be presented as $[D_{i,j}/B_{i,j}^+, D_{i,j}/B_{i,j}^-]$. And the math model can be calculated with interval arithmetic operators to obtain the objective values. This paper focuses on the design of the deterministic scheduling algorithm and the uncertainty is not considered, but it will be considered in our future work.

## 4. Details of the proposed algorithm

As a population-based evolutionary algorithm, EDA is good at exploration among solution space by sampling the probability model, which is designed to describe the distribution of solutions. In our previous work, eEDA was proposed to solve the DAG scheduling problem [27]. In this paper, cooperated with the task processing sequence probability model in eEDA, a VSL assignment probability model is added to form a multi-model EDA (mEDA) to solve the energy-efficient DAG scheduling problem.

### 4.1. Flowchart

The flowchart of the multi-model EDA for energy efficient DAG scheduling problem is illustrated in Fig. 2 and the notations are explained as Table 3.

For the mEDA, the population is divided into three sub-populations, including fast population, energy-efficient population and learning population. Fast sub-population is a sub-population in which VSL assignment arrays are initialized as the highest VSLs. Energy efficient sub-population is set the lowest VSLs to save energy. Learning population is initialized by sampling heuristic solutions. Meanwhile, three corresponding VSL assignment probability (*VP*) models, including fast *VP* (F-*VP*), energy efficient *VP* (E-*VP*) and learning *VP* (L-*VP*), are used to generate VSL assignment array. These three *VP*s are designed to learn and update the VSL assignment arrays for each sub-population. In addition, a

**Table 2**
Mathematical model notation.

| | Notation | Implication |
|---|---|---|
| Decision variables | $x_{j,i,r}$ | $\{0, 1\}$, $x_{j,i,r} = 1$ denotes that task $j$ is the $r$th task processed on machine $i$, otherwise, $x_{j,i,r} = 0$; |
| | $rs_j$ | Relative speed of task $j$; |
| | $m_j$ | $\{0, 1, 2, \ldots, m-1\}$, processor assignment of task $j$; |
| Problem variable and constraints | $n$ | Number of tasks to be processed; |
| | $m$ | Number of processors; |
| | $t_j$ | Computation time of task $j$; |
| | $cm_{i,j}$ | Communication time between task $i$ and task $j$; |
| | $i \rightarrow j$ | Task $i$ is precedent of task $j$; |
| Intermediate variables | $C_j$ | $C_j \geq 0$, the completion time of task $j$; |
| | $V_j$ | Voltage supply of task $j$, $V_{\text{low}}$ represents the lowest supply voltage; |
| | $\delta(m_i, m_j)$ | $\{0, 1\}$, $\delta(m_i, m_j) = 1$ denotes $m_i \neq m_j$, $\delta(m_i, m_j) = 0$ denotes $m_i = m_j$; |
| Others | $M$ | A large constant; |
| | $\beta$ | A constant for energy consumption calculation. |

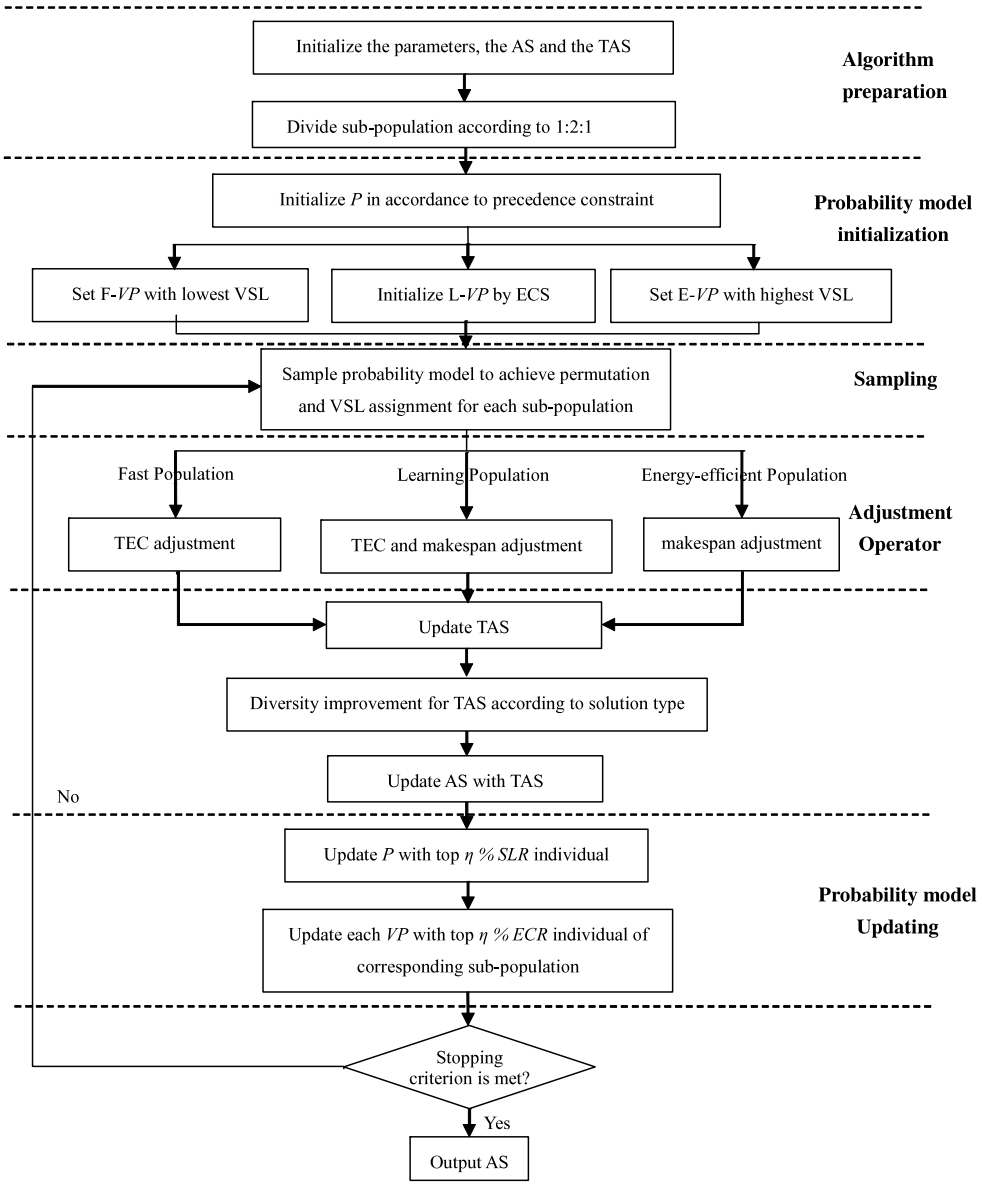


**Fig. 2.** Flowchart of mEDA for energy efficient DAG scheduling problem.

task permutation probability model ($P$) is used to generate task processing permutation.

At each generation, the solutions in three sub-populations are adjusted by the corresponding operators, and TAS is reset first

**Table 3**
Algorithm flowchart notation.

| Notation | Implication |
|---|---|
| AS | Archive set, store the non-dominated solutions generated by the algorithm; |
| TAS | Temporary archive set, store the non-dominated solutions generated in one generation; |
| *P* | Task permutation probability model, used to generate task processing permutation; |
| *VP* | VSL assignment probability model, used to generate VSL assignment array; |
| *SLR* | Schedule length ratio; |
| *ECR* | Energy consumption ratio. |

and then used to store the non-dominate solutions explored in current generation. Meanwhile, an improvement operator is used to increase the diversity of TAS. Then, it updates AS with TAS. With the new elite solutions, the probability models are updated, it goes to the next generation until the stopping condition is met.

### 4.2. Multi-model EDA scheme for energy-efficient DAG scheduling problem

#### 4.2.1. Probability model for task permutation

The task permutation probability model is designed to produce task processing permutation for the whole population. Since there are precedence constraints between the tasks, a relative position probability is designed as (18), where $p_{i,j}(g)$ represents the probability that task $i$ is placed in front of task $j$ in the permutation at the $g$th generation. Clearly, $p_{i,j}(g) + p_{j,i}(g) = 1$ because task $i$ is in front of or behind task $j$.

$$P(g) = \begin{bmatrix} 0 & p_{1,2}(g) & \cdots & p_{1,n}(g) \\ p_{2,1}(g) & 0 & \cdots & p_{2,n}(g) \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1}(g) & p_{n,2}(g) & \cdots & 0 \end{bmatrix}. \tag{18}$$

**Initialization:**

As the task processing permutation should satisfy the precedence constraints, the probability value of $p_{i,j}$ is initialized as follows, which ensures the precedence constraints of the task pairs. For the task pairs without constraints, their order is given randomly.

$$p_{i,j}(0) = \begin{cases} 1, & i \to j \\ 0, & j \to i \\ 0.5, & otherwise. \end{cases} \tag{19}$$

**Updating method:**

For the single-objective optimization problem [27], the individuals with small makespan are used to update $P(g)$. For the multi-objective optimization problem discussed in this paper, makespan depends on both task processing permutation and VSL assignment. As $P$ aims at learning good task permutation structure from solutions with less makespan, to reduce the influence of VSL assignment, makespan is normalized by the length of CP called "schedule length ratio" (*SLR*) as follows:

$$SLR = C_{\max} \Big/ \sum_{j \in CP} \frac{t_j}{rs_j}. \tag{20}$$

All the individuals are sorted in an ascending order with respect to *SLR*. Then, the top $\eta$% of solutions are selected as elite population. The probability value is updated by using the following population based incremental learning method (PBIL) [1]:

$$p_{i,j}(g+1) = (1-\alpha)p_{i,j}(g) + \alpha \frac{1}{E} \sum_{k=1}^{E} I_{i,j}^{k}(g) \tag{21}$$

$$I_{i,j}^{k}(g) = \begin{cases} 1, & \text{if task } i \text{ is before task } j \text{ in the } k\text{th individual} \\ 0, & \text{otherwise} \end{cases} \tag{22}$$

where $\alpha \in (0, 1)$ denotes the learning rate, $E$ denote the size of the elite population, and $I_{i,j}^{k}(g)$ is the indicator function corresponding to the $k$th solution of the elite population.

**Sampling method**

In EDA, the new individuals are produced by sampling the updated probability model. For each position which has not been determined, an array of length $n$ is calculated. The $l$th element denotes the probability that task $l$ is assigned to this position. If task $l$ has already been scheduled, the $l$th element is set as 0. Otherwise, it is calculated by $p_{k,l}$ where $k$ denotes that all the tasks have not been scheduled. Then the array is normalized and sampled by the roulette wheel method. Please refer [27] for the details.

#### 4.2.2. Probability model for VSL assignment

Three VSL assignment probability models are designed to produce VSL assignment array for corresponding sub-populations. *VP* is designed as (23), where $vp_{jl}(g)$ shows that task $j$ is set the $l$th VSL at the $g$th generation ($l = 0, 1, 2, 3$). Obviously, $\sum_l vp_{jl} = 1, \forall j$.

$$VP(g) = \begin{bmatrix} vp_{10}(g) & vp_{11}(g) & vp_{12}(g) & vp_{13}(g) \\ vp_{20}(g) & vp_{21}(g) & \cdots & vp_{23}(g) \\ \vdots & \vdots & \ddots & \vdots \\ vp_{n0}(g) & vp_{n1}(g) & \cdots & vp_{n4}(g) \end{bmatrix}. \tag{23}$$

**Initialization:**

Three sub-populations are divided to produce solutions with different VSLs. Fast population, 25% of the population, is designed to be efficient on makespan, and then VSLs of tasks are set the lowest values. The tasks are processed at the fastest speed and thus the energy consumption is the largest. F-*VP* is initialized as (24). Energy efficient population, 25% of the population, is designed to save energy so the VSLs are set the highest values. The tasks are processed at the slowest speed and thus the energy consumption is the smallest. E-*VP* is initialized as (25).

$$\text{F-}VP(g) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix} \tag{24}$$

$$\text{E-}VP(g) = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{25}$$

Learning population, 50% of the population, is designed to find appropriate VSL assignment array by learning from the heuristic *ECS* during the initialization of L-*VP*. The frequency that task $j$ is set

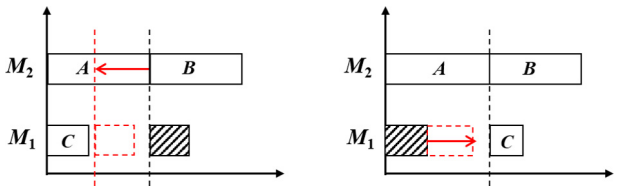


**Fig. 3.** Corresponding Gantt graph of the adjustment operators.

as VSL $l$ is set as the element $l-vp_{jl}$ of $L-VP(0)$. $L-VP$ is initialized as (26) where $S$ denotes the size of learning sub-population.

$$l-vp_{jl}(0)=\frac{1}{S}\sum_{k=1}^{S}\widetilde{l}_{ij}^{k} \tag{26}$$

$$\widetilde{l}_{jl}^{k}=\begin{cases}1, \text{ if under ECS heuristic, task } j \text{ is assignd } l\text{th}\\ \quad \text{VSL in the } k\text{th individual}\\ 0, \text{ otherwise.}\end{cases} \tag{27}$$

**Updating and sampling method:**

As the updating method of $P$, a normalized TEC, "energy consumption ratio" ($ECR$) is used as the metric to choose elite population as follows.

$$ECR=TEC\Big/\sum_{j\in CP}\frac{t_j}{rs_j}\cdot V_{low}^2. \tag{28}$$

The individuals of each sub-population are sorted in an ascending order with respect to $ECR$. Then, the top $\eta$% of each sub population are selected to update the corresponding VAPM. PBIL is also adopted to update the probability model.

$$vp_{jl}(g+1)=(1-\alpha)vp_{jl}(g)+\alpha\frac{1}{E}\sum_{k=1}^{E}\widetilde{l}_{ij}^{k}(g) \tag{29}$$

$$\widetilde{l}_{jl}^{k}(g)=\begin{cases}1, \text{ if task } j \text{ is assignd } l\text{th VSL in the } k\text{th individual}\\ 0, \text{ otherwise}\end{cases} \tag{30}$$

where $\alpha \in (0,1)$ denotes the learning rate, and $\widetilde{l}_{jl}^{k}(g)$ is the indicator function corresponding to the $k$th solution of the elite population.

In the $g$th generation, the VSL assignment of task $j$ is sampled from the $j$th row of $VP(g)$ by the roulette wheel method.

### 4.3. Adjustment operators

To optimize the solutions, two adjustment operators are designed to decrease makespan and TEC, respectively, in accordance with the idle between the tasks.

#### 4.3.1. Makespan adjustment operator

For each idle of the scheduling solution, the task after the idle (the shadowed task in the left of Fig. 3) is considered. The parent task which delays the start time of the task (task A) is found. Then it speeds up task A to make the shadowed task produced earlier. The energy increase by speeding up and energy decrease by shortening the idle are compared. If the energy consumption is not increased, then VSL of task A is decreased. The simplified Gantt graph is shown in the left of Fig. 3.

#### 4.3.2. TEC adjustment operator

For each idle of the scheduling solution, the task before the idle (the shadowed task in the right of Fig. 3) is slowed down as much as possible to decrease the energy consumption without delaying the successive tasks. The simplified Gantt graph is shown in the right of Fig. 3.

The above two operators are used to improve the solutions. According to the type of sub-population, different adjustment operators are chosen. Makespan adjustment operator is adopted to optimize fast population, while TEC adjustment operator is adopted to optimize energy efficient population. Learning population is improved by both TEC and makespan adjustment operators. As TEC adjustment operator does not increase makespan, it is used after makespan adjustment for learning population.

```
Indi: the k-th individuals in the population; Indi': new Indi after VSL alteration;
ls: local search record number, initialized as 0.
FOR each Indi in TAS
    WHILE ls < n
        WHILE VSL of task is legal
            VSL = VSL+1 (VSL-1), obtain Indi'
            ls ++
            IF Indi' ≻ Indi
                Restore VSL of task
            ELSE IF Indi' ≺ Indi
                Update TAS with Indi'
                Indi ← Indi'
            ELSE
                Update TAS with Indi'
                IF rand(0, 1) < probability
                    Indi ← Indi'
                ELSE
                    Restore VSL of task
```

**Fig. 4.** Pseudo code of diversity increasement scheme.

#### 4.3.3. Diversity improvement

To improve the diversity of TAS, an improvement operator based on VSL alteration is designed. For each individual in TAS, if the individual belongs to fast or energy efficient sub-population, energy consumption value of each task is calculated and VSL is increased; and if it belongs to learning sub-population, b-level value is calculated and VSL is decreased. The tasks of the individual are considered in a descending order of energy consumption or bottom level (b-level) value. B-level value of the task is the longest path from the task to exit task [11]. Decrease the VSL of a task with large b-level value, $EST$ of many tasks might be decreased so as to optimize the makespan. After the VSL alteration, a new individual is obtained and the new makespan and TEC are calculated for it. If neither makespan nor TEC decreases, the solution is abandoned; and if both makespan and TEC decreases, the solution is used to replace the former individual; Otherwise, the solution is used to update TAS and replace the former solution at a certain probability. The corresponding pseudo code is presented as Fig. 4.

The probability is set as 0.5 for individuals in learning sub-population and 0.8 for the other individuals. The reason is that for the individuals in fast population and energy efficient population, the VSL assignment is settled as the lowest and highest at first and the individuals in Pareto set are close to each other. Set a high probability for these individuals may increase the diversity of VSL assignment settings. On the other hand, a lower probability for learning population may retain the VSL assignment learned from $ECS$ and elite population accordingly.

## 5. Numerical test and comparisons

To test the effectiveness of the proposed mEDA, we compare it with some existing algorithms including two heuristic methods,

**Table 4**
Factor levels of parameters.

| Parameters | Factor level | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| $\eta$ | 0.10 | 015 | 0.20 | 0.25 |
| $\alpha$ | 0.05 | 0.10 | 0.15 | 0.20 |

*ECS*, *ECS* + *idle* [12] and parallel bi-objective hybrid genetic algorithm (bGA) [15]. For fair comparison, *ECS*, *ECS* + *idle* and bGA are coded in C language according to [15–16] and run on the same personal computer with a 2.83 GHz processor and 4 GB RAM.

### 5.1. Benchmark problem study

The benchmark problem is generated by the standard task graph benchmark instances on homogeneous DAG scheduling problem provide by Tobita [23]. Instances in [23] do not include the inter-task data size, the communication time values contributing to normal distribution according to different *CCR* settings are produced and used for experiments. For the testing set, $m = \{2, 4, 8, 16\}$, $n = \{50, 100, 300\}$, and each scale of task number consists of 180 independent testing instances. In addition, *CCR* is considered in this paper, and communication time values are produced contributing to normal distribution. As five *CCR* values are considered, $CCR = \{0.1, 0.2, 0.5, 1.0, 10.0\}$, the benchmark is enlarged to $4 \times 3 \times 5 \times 180 = 10800$ instances.

#### 5.1.1. Experiment settings

To compare mEDA with bGA fairly, the population size ($N$) of mEDA is set 50 as the same of bGA. The stopping criterion of both algorithms is set 200 generations unless AS is not updated for more than 20 generations. For bGA, the parameters are set as [15]. The crossover rate and mutation rate are set 1.0 and 0.35, respectively. The migration topology is ring and the migration rate is 20 which means that the individuals are migrated every 20 generations. In addition, the migrations size is 5.

For mEDA, the other two key parameters are $\eta$ (percentage of elite population, $E = N \times \eta\%$) and $\alpha$ (learning rate) which are determined by Taguchi method of design-of-experiment method (DOE) [16]. DOE is adopted to investigate the influence of parameters on performance of the algorithm under different parameter levels. Orthogonal arrays of different parameter levels are used to conduct the experiment with few runs. In this experiment, four levels are set for each parameter as Table 4 and $4^2$ full-factorial experiments are employed.

For each combination of $n \times m \times CCR$ ($3 \times 4 \times 5 = 60$), an instance is randomly chosen. For each instance, 16 combinations of $\eta \times \alpha$ are tested independently and the obtained AS $AS_{c\_i}$ ($c\_i = 1, 2, \ldots, 16$) are stored. The final AS (FAS) is obtained by integrating $AS_1, AS_2, \ldots, AS_{16}$. Then, the contribution of a certain combination ($CON$) is calculated as $CON(c\_i) = \left|AS'_{c\_i}\right| / |FE|$, where $AS'_{c\_i} = \{X_l \in AS_{c\_i} | \exists X_{l'} \in FAS, X_l = X_{l'}\}$. The average $CON$ of each combination is calculated and used as the response value (RV), shown in Table 5.

The influence trend of DOE is listed in Table 6 and the main effects plot is shown in Fig. 5. From Table 6, it can be seen that the influence of $\alpha$ on the performance is the largest, and the response value variance under different levels of $\eta$ is little. From Fig. 5 and Table 6, it can be seen that for $\alpha$ with a maximum of 200 generations, when the scale of problem is not too big, a little $\alpha$ is suitable for the benchmark problem; for $\eta$, the influence is slight, and its value affects the ratio of local search. According to the above investigation, $\eta = 20$ and $\alpha = 0.05$ are suggested.

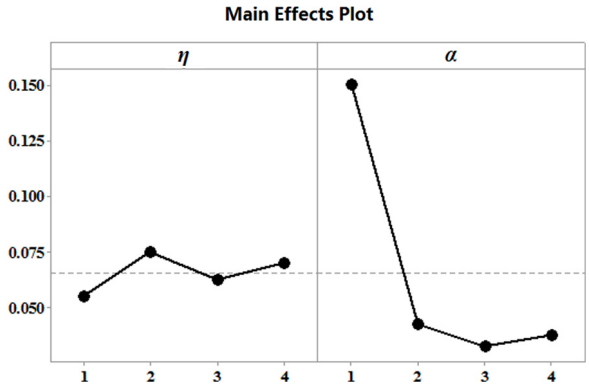


**Fig. 5.** The influence trend of each parameter.

**Table 6**
Influence trend table.

| Level | $\eta$ | $\alpha$ |
|---|---|---|
| 1 | 0.0568 | 0.13454 |
| 2 | 0.0723 | 0.04432 |
| 3 | 0.04736 | 0.03183 |
| 4 | 0.07105 | 0.03681 |
| Delta | 0.02494 | 0.10271 |
| Rank | 2 | 1 |

#### 5.1.2. Comparison with heuristic methods

As the heuristic method produces one solution for each problem, mEDA is compared with the heuristic methods in terms of the percentage of the solutions of heuristics dominated by the solutions of mEDA. The average value of 180 instances under each problem scale is calculated and summarized according to *CCR* (communication to computation ratio), $m$ (number of processors) and $n$ (number of tasks) in Table 7.

From this table, it can be seen that mEDA improves *ECS* solutions on 75.6% instances and *ECS*+*idle* solutions on 78.6% instances. It can be seen clearly that in terms of *CCR*, mEDA performs better as *CCR* grows compared with both heuristic methods. The possible reason is when the transfer data are large, the processor assignment dominated by RS function in *ECS* is almost fixed. The tasks are assigned to the processor of its parents with a large amount of data and it is difficult to improve the task assignment. In mEDA, the diversity of task processing permutation and the adjustment operators improve the performance of solutions. In terms of $m$, compared with *ECS*, mEDA performs better when $m$ is large and compared with *ECS* + *idle* performs better when $m$ is small. This is mainly because that *ECS* is good at finding efficient VSL–processor

**Table 5**
The RV value.

| Experiment numbers | Factor | | RV (s%) | Experiment numbers | Factor | | RV (%) | Experiment numbers | Factor | | RV (%) | Experiment numbers | Factor | | RV (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\eta$ | $\alpha$ | | | $\eta$ | $\alpha$ | | | $\eta$ | $\alpha$ | | | $\eta$ | $\alpha$ | |
| 1 | 1 | 1 | 0.13 | 5 | 2 | 1 | 0.17 | 9 | 3 | 1 | 0.18 | 13 | 4 | 1 | 0.12 |
| 2 | 1 | 2 | 0.05 | 6 | 2 | 2 | 0.05 | 10 | 3 | 2 | 0.03 | 14 | 4 | 2 | 0.04 |
| 3 | 1 | 3 | 0.02 | 7 | 2 | 3 | 0.05 | 11 | 3 | 3 | 0.03 | 15 | 4 | 3 | 0.03 |
| 4 | 1 | 4 | 0.02 | 8 | 2 | 4 | 0.03 | 12 | 3 | 4 | 0.01 | 16 | 4 | 4 | 0.09 |

**Table 7**
*C* metric comparative results.

| | | Dominating *ECS* (%) | Dominating *ECS* + *idle* (%) |
|---|---|---|---|
| *CCR* | 0.1 | 54.54 | 54.31 |
| | 0.2 | 63.47 | 62.18 |
| | 0.5 | 78.47 | 86.71 |
| | 1 | 87.73 | 94.91 |
| | 10 | 90.60 | 95.05 |
| *m* | 2 | 71.11 | 94.63 |
| | 4 | 69.89 | 83.63 |
| | 8 | 73.41 | 68.74 |
| | 16 | 85.44 | 67.52 |
| *n* | 50 | 80.13 | 79.83 |
| | 100 | 76.19 | 77.67 |
| | 300 | 70.42 | 78.17 |

combination. When the *m* is small, the solution scale of task assignment is small, *ECS* is more likely to find good solutions. And when *m* is large, for the same task graph, the number and length of idles in scheduling solution increase. As *ECS* + *idle* is good at handling the idles, *ECS* + *idle* performs better when *m* is larger. mEDA makes a trade-off between these two heuristic methods. In terms of *n*, mEDA has a large superiority under problems with different scales.

#### 5.1.3. Comparison with bGA

Next, it compares mEDA with bGA in terms of *C* metrics. The average *C* metric values of 180 instances under each scale are calculated and summarized according to *CCR*, *m* and *n* in Table 8.

From this table, it can be seen that 67% bGA solutions are dominated by mEDA. On the other hand, only 11.6% mEDA solutions are dominated by bGA. For the average number of Pareto solutions, mEDA produces 18.26 solutions for each instance while bGA produces 6.14 solutions. So, mEDA is able to offer more choices for the decision maker.

In terms of *C* metric, it can be seen that mEDA performs better than bGA. Compared with heuristic methods before, task processing permutations are evolved and optimized in bGA and mEDA instead of fixed by priority. In bGA, VSL and processor assignment are determined by *ECS* methods and this may lead to local minimum. On the other hand, two probability models are used to record, learn and evolve both task processing permutation and VSL assignment array in mEDA. When *m* is little, *C*(mEDA, bGA) is small as bGA performs better with efficient VSL assignment. When *m* increases, mEDA performs better as the algorithm is good at exploration. In terms of *CCR*, *C*(mEDA, bGA) and *C*(bGA, mEDA) both increase. One possible reason is that the numbers of Pareto solutions decrease and the proportion of solutions dominated by each other grows. In terms of *n*, it can be seen when $n = 300$, *C*(mEDA, bGA) is large and *C*(bGA, mEDA) is small. It shows that mEDA performs much better than bGA when the problem scale is large.

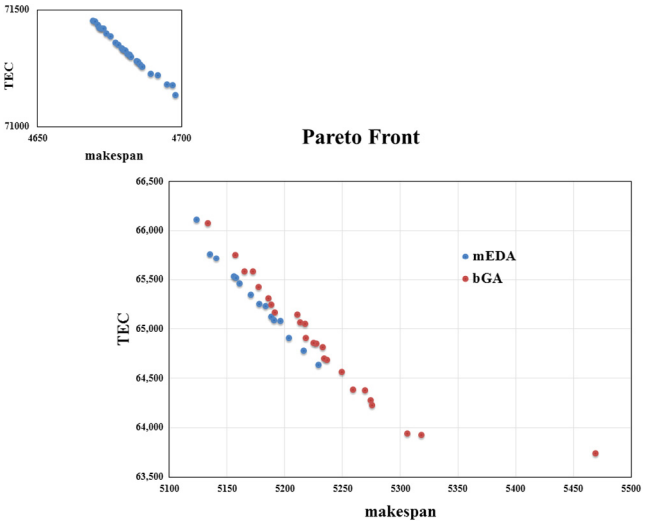


**Fig. 6.** Pareto fronts of both algorithm.

### 5.2. Case study

A case study is carried out to verify the performance of mEDA on large scale problems. CyberShake workflow instance of pegasus projects [17] is used. The CyberShake workflow is used to characterize earthquake hazards in a region by the Southern California Earthquake Center. The biggest scale instance is tested in our work where $n = 1000$. The problem settings are set as [21] where $m = 5$, $B = 12$ MB/s. The parameters of both bGA and mEDA are set as before. The algorithms are tested independently and the Pareto fronts obtained by both algorithms are presented in Fig. 6 (other 3 solutions with very large makespan and low *TEC* obtained by mEDA). It can be seen that more than half solutions of bGA are dominated by mEDA and no solutions of mEDA is dominated by bGA. In addition, some extreme solutions are obtained by mEDA. Thus the Pareto front of bGA is more uniform than mEDA's. Improving the diversity and uniformity of Pareto fronts of our algorithm is one of our future works.

## 6. Conclusion

Energy efficient scheduling problem under cloud computing system is considered in this paper. A multi model EDA is designed to where one probability model produces task processing permutation and the other produces VSL assignment array. Three subpopulations are adopted to explore larger solution space. Besides, some operators and scheme are designed to optimize the solutions. Compared with heuristic methods, a large proportion of solutions

**Table 8**
*C* metric comparative results.

| | | Avg number of Pareto solutions of mEDA | Avg number of Pareto solutions of bGA | *C* (mEDA, bGA) | *C* (bGA, mEDA) |
|---|---|---|---|---|---|
| *CCR* | 0.1 | 23.25 | 11.31 | 0.647 | 0.033 |
| | 0.2 | 23.12 | 10.40 | 0.621 | 0.064 |
| | 0.5 | 22.02 | 5.63 | 0.635 | 0.129 |
| | 1 | 13.64 | 1.86 | 0.721 | 0.160 |
| | 10 | 9.26 | 1.51 | 0.725 | 0.197 |
| *m* | 2 | 40.72 | 17.27 | 0.444 | 0.090 |
| | 4 | 18.46 | 4.32 | 0.673 | 0.109 |
| | 8 | 9.43 | 1.78 | 0.772 | 0.118 |
| | 16 | 4.41 | 1.20 | 0.791 | 0.149 |
| *n* | 50 | 15.27 | 4.80 | 0.613 | 0.108 |
| | 100 | 17.91 | 5.89 | 0.621 | 0.210 |
| | 300 | 21.59 | 7.72 | 0.776 | 0.031 |

are dominated by mEDA. Compared with bGA, mEDA is better than bGA in terms of both number of Pareto solutions and $C$ metric. The experimental comparisons demonstrate the efficiency of mEDA.

The future work could focus on heterogeneous system which is closer to the actual cloud computing scheduling problem. Uncertainty is considered to be added into our model and we are going to design a corresponding scheduling algorithm. On the other hand, in terms of the objective, the economic factor is considered. Utilization of processors influences the cost of guests. A large number of processors lead to good time performance but cost a lot of money and energy. We would like to find a trade-off between time performance, energy consumption and expense.

## Acknowledgments

This research is supported by the National Key R&D Program of China [No. 2016YFB0901900] and the National Natural Science Fund for Distinguished Young Scholars of China [No. 61525304].

## References

[1] S. Baluja, Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning, Carnegie-Mellon University (Pittsburgh), Department of Computer Science, 1994, pp. 1–20.
[2] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.
[3] K. Bousselmi, Z. Brahmi, M.M. Gammoudi, Energy efficient partitioning and scheduling approach for scientific workflows in the cloud, in: IEEE International Conference on Services Computing, IEEE Computer Society, 2016, pp. 146–154.
[4] D. Bozdag, U. Catalyurek, F. Ozguner, A task duplication based bottom-up scheduling algorithm for heterogeneous environments, in: Parallel and Distributed Processing Symposium, IPDPS, 2016, pp. 1–12.
[5] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, J. Wu, Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment, J. Syst. Softw. 99 (2) (2015) 20–35.
[6] F. Ferrandi, P.L. Lanzi, C. Pilato, D. Sciuto, A. Tumeo, Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 29 (6) (2010) 911–924.
[7] M. Guzek, P. Bouvry, E.G. Talbi, A survey of evolutionary computation for resource management of processing in cloud computing, IEEE Comput. Intell. Mag. 10 (2) (2015) 53–67.
[8] A. Hameed, A. Khoshkbarforoushha, R. Ranjan, et al., A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems, Computing 98 (7) (2016) 751–774.
[9] Y. Hu, C. Liu, K. Li, Slack allocation algorithm for energy minimization in cluster systems, Future Gener. Comput. Syst. 74 (2017) 119–131.
[10] H. Kimura, M. Sato, Y. Hotta, Emprical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster, in: IEEE International Conference on CLUSTER Computing, IEEE, 2006, pp. 1–10.
[11] Y.K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (1999) 406–471.
[12] Y.C. Lee, A.Y. Zomaya, Energy conscious scheduling for distributed computing systems under different operating conditions, IEEE Trans. Parallel Distrib. Syst. 22 (8) (2011) 1374–1381.
[13] Z. Li, J. Ge, H. Hu, W. Song, H. Hu, B. Luo, Cost and energy aware scheduling algorithm for scientific workflows with deadline constraint in clouds, IEEE Trans. Serv. Comput. 1 (1) (2015) 1–15.
[14] B. Li, L. Wang, B. Liu, An effective pso-based hybrid algorithm for multiobjective permutation flow shop scheduling, IEEE Trans. Syst. Man Cybern. A 38 (4) (2008) 818–831.
[15] M. Mezmaz, N. Melab, Y. Kessaci, Y.C. Lee, E.G. Talbi, A.Y. Zomaya, D. Tuyttens, A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems, J. Parallel Distrib. Comput. 71 (11) (2011) 1497–1508.
[16] D.C. Montgomery, Design and Analysis of Experiments, Wiley, New York, 1984.
[17] Pegasus projects: https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.
[18] M. Ranjbari, J.A. Torkestani, A learning automata-based algorithm for energy and SLA efficient consolidation of virtual machines in cloud data centers, J. Parallel Distrib. Comput. 113 (2018) 55–62.
[19] N. Sample, P. Keyani, G. Wiederhold, Scheduling under uncertainty: Planning for the ubiquitous grid, in: Coordination Models and Languages, Springer, Berlin/Heidelberg, 2002, pp. 323–338.
[20] S. Selvi, D. Manimegalai, DAG scheduling in heterogeneous computing and grid environments using variable neighborhood search algorithm, Appl. Artif. Intell. 31 (2) (2017) 134–173.
[21] C. Szabo, T. Kroeger, Evolving multi-objective strategies for task allocation of scientific workflows on public clouds, in: Evolutionary Computation, IEEE, 2012, pp. 1–8.
[22] The website of NRDC: https://www.nrdc.org/.
[23] T. Tobita, H. Kasahara, A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, J. Sched. 5 (5) (2002) 379–394.
[24] H. Topcuoglu, S. Harir, M.Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Trans. Parallel Distrib. Syst. 13 (3) (2002) 260–274.
[25] J.D. Ullman, NP-complete scheduling problems, J. Comput. System Sci. 10 (3) (1975) 384–393.
[26] S.Y. Wang, L. Wang, An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, IEEE Trans. Syst. Man Cybern. Syst. 46 (1) (2015) 139–149.
[27] C.G. Wu, L. Wang, X.L. Zheng, An effective estimation of distribution algorithm for solving uniform parallel machine scheduling problem with precedence constraints, in: IEEE International Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 2626–2632.
[28] Y. Xu, K. Li, L. He, T.K. Truong, A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization, J. Parallel Distrib. Comput. 73 (9) (2013) 1306–1322.
[29] Y. Xu, K. Li, J. Hu, K. Li, A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues, Inform. Sci. 270 (6) (2014) 255–287.
[30] S. Yassa, R. Chelouah, H. Kadima, B. Granado, Multi-objective approach for energy-aware workflow scheduling in cloud computing environments, Sci. World J. (2013).
[31] Y. Zhang, Y. Wang, H. Wang, Energy-efficient task scheduling for DVFS-enabled heterogeneous computing systems using a linear programming approach, in: PERFORMANCE Computing and Communications Conference, IEEE, 2017, pp. 1–8.
[32] S. Zhuravlev, J.C. Saez, S. Blagodurov, A. Fedorova, M. Prieto, Survey of energy-cognizant scheduling techniques, IEEE Trans. Parallel Distrib. Syst. 24 (7) (2013) 1447–1464.

**Chu-ge Wu** received the B.Sc. degree in automation from Tsinghua University, Beijing, China, in 2015, where she is currently pursuing the Ph.D. degree in control theory and control engineering.

Her current research interest includes scheduling and optimization algorithms for heterogeneous computing systems, intelligent optimization.

**Ling Wang** received the B.Sc. in automation and Ph.D. degrees in control theory and control engineering from Tsinghua University, Beijing, China, in 1995 and 1999, respectively. Since 1999, he has been with the Department of Automation, Tsinghua University, where he became a full professor in 2008.

His current research interests include intelligent optimization and production scheduling. He has authored five academic books and more than 260 refereed papers.