Contents lists available at ScienceDirect

Knowledge-Based Systems

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

A path relinking enhanced estimation of distribution algorithm for direct acyclic graph task scheduling problem

Chu-ge Wu, Ling Wang [*], Jing-jing Wang

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

ARTICLE INFO

Article history:
Received 22 November 2020
Received in revised form 22 June 2021
Accepted 24 June 2021
Available online 25 June 2021

Keywords:
Estimation of distribution algorithm
Direct acyclic graph task scheduling
Path relinking
Makespan
Workflow scheduling

ABSTRACT

Superior task scheduling scheme is able to improve the performance in achieving shorter task completion time in multi-processor computing system. Large scale applications are generally modelled as direct acyclic graph (DAG) to be processed efficiently in parallel. To solve DAG task scheduling problem (DAG-SP) with the criterion of minimizing makespan, this paper proposes an estimation of distribution algorithm (EDA) enhanced by the path relinking. An efficient hybrid scheme integrating list scheduling heuristics is designed to take advantage of the knowledge of existing works. In addition, to describe the relative position relationships between the task pairs, a specific probability model is built and the task processing permutations are produced by sampling such a model. To enhance the exploitation of EDA, a path relinking based knowledge is used to design the local search method. Simulation experiments are carried out with both benchmark datasets and real-world graphs, where the comparative results show that the above designs can improve the performance effectively. Moreover, the numerical comparisons show that the proposed algorithm performs significantly better than the existing heuristics and evolutionary algorithms.

1. Introduction

Heterogeneous computing system (HCS) refers to a set of processors interconnected with a network, which are equipped with different computing and storage capacities. HPCs provide support for executing computationally intensive parallel applications. To facilitate scheduling decision, an application is generally modelled as a directed acyclic graph (DAG) [1], such as scientific workflows [2] and deep neural networks [3]. Efficient DAG task scheduling [4] is able to improve the application performance and the quality of user experience [5–7]. Considering both academic significance and practical value of DAG task scheduling problem (DAG-SP), this paper will address DAG-SP with the overall completion time minimization.

It is difficult to obtain efficient schedules because of the heterogeneity within the computing system and the precedence constraints between tasks, as well as the NP-hard nature of DAG-SP [8]. To solve the problem more efficiently, some works are presented to describe, model and formulate the problem. A deterministic static model is given in [1] and two schedule strategies based on this model are proved that optimal solutions can be produced if there are only two processors, and the execution periods are the same when the DAG structure is arbitrary or the execution periods are arbitrary when the DAG is an in-tree. For other situations, there does not exist a polynomial algorithm for DAG-SP due to its NP-hard nature. In addition, lower bound analysis and mixed integer linear programming (MILP) models are presented to provide a guideline for the optimization algorithms. Lower bounds of necessary number of processors, communication requests and makespan are considered in [9] and the tasks are merged and their earliest starting time (EST) are evaluated to calculate the lower bound of makespan. The above method is extended to solve the problem with total weighted completion time minimization via time-indexed linear programming relaxations in [10]. Furthermore, a knowledge-based heuristic called generalized earliest time first (GETF) is proposed where the gap between GETF solution and the optimal makespan is provided in [11]. Processors are grouped into clusters according to its speed and the task-processor assignments are suggested by the solutions of a MILP model. Similar with [11], a MILP enhanced heuristic method is proposed in [12] where an extended DAG scheduling model is given to formulate DAG-SP with deadline constraints in IoT–fog system. A greedy individual time allowance approach based on MILP with binary relaxation is proposed.

Since it is difficult to obtain the optimal solution of DAG-SP within an acceptable duration under most practical cases, heuristic methods and evolutionary algorithms are widely applied to solve DAG-SP in the past few decades. Kwok and Ahmad [13] gives a detailed review on DAG scheduling heuristics and other

[*] Corresponding author.
E-mail addresses: wucg15@mails.tsinghua.edu.cn (C. Wu), wangling@mail.tsinghua.edu.cn (L. Wang), wjj18@mails.tsinghua.edu.cn (J. Wang).

https://doi.org/10.1016/j.knosys.2021.107255
0950-7051/© 2021 Elsevier B.V. All rights reserved.

algorithms, which are categorized by hybrid processor states and DAG structures. Heuristics provide approximate solutions within polynomial time periods. For DAG-SP, [14] summarizes heuristics into three categories: clustering, task duplication and list scheduling. In clustering heuristic, tasks are divided into groups, which are merged until its number equals to the number of processors. This approach performs poor in heterogeneous system [15]. Task duplication heuristic, such as duplication-based bottom-up scheduling algorithm (DBUS) [16], repeatedly allocates a set of tasks onto different processors to decrease data transfer time between processors. However, duplicated tasks increase the power consumption of processors, which is inacceptable for some computing system. Compared to these two heuristics, list heuristic provides better solutions with lower computing complexity under most situation by sorting tasks according to DAG properties. To ensure the valid topological order of tasks, this method contains two main phases: task prioritizing and processor selection.

Dynamic level scheduling [17] sorts tasks depending on an estimate of the availability of each processor and adopts EST to select the processor. The processor selection method cannot guarantee a minimum finish time when processors are not homogeneous. To fit heterogeneous environment, heterogeneous earliest finish time (HEFT) [18], a well-known heuristic is proposed, where tasks are sorted according to upward rank values (bottom-level values), i.e., the longest path length from the node to the bottom of DAG. Then, tasks are allocated to available idle time slot of the processors towards the earliest finish time (EFT). Considering the heterogeneity of computing system, the impact of different calculations, such as mean, median, worst, best values of DAG variables on different processors is studied in [19]. DAG is divided into a set of unlisted-parent trees and the task permutation is constructed according to the trees and average EST in [20], some heuristics forecast the impact of task assignment on its successive tasks. Based on HEFT, the task is assigned onto the processor minimizing the maximum EFT of its children tasks [21]. Inspired by this work, [15] builds an optimistic cost table (OCT) to list the maximum of shortest paths from its children nodes to the exit node for each combination of task and processor. For each task, average OCT value is considered and used to sort the tasks. At last, tasks are assigned to processors considering both EFT and OCT with insert-based policy. Inspired by the efficiency of existing list heuristics, task prioritizing and processor selection methods designed according to properties of DAG-SP can be abstracted as the knowledge for task scheduling.

Evolutionary algorithms are able to provide satisfactory solutions within acceptable time by data-driven search process for complex problems. Compared to heuristics, evolutionary algorithms aim to improve the quality of solutions by iteration-based search process. For DAG-SP, a variety of evolutionary techniques are reviewed in [22], where genetic algorithm (GA) [23], ant colony optimization (ACO) [24], differential evolution (DE) [25] and chemical reaction optimization (CRO) [26] are adopted. A double molecular structure-based CRO is proposed in [26] to generate and evolve task processing sequence as well as task-to-computing-node mapping sequence simultaneously. Similarly, a multiple priority queues leading GA is proposed in [23] to solve DAG-SP. The population is initialized with multiple task priority queues produced by list heuristics and its combinations and mutations. After this efficient initialization, crossover and mutation operators are adopted to evolve the solutions. Insert-based HEFT policy is as the task-processor mapping mechanism. A hybrid evolutionary algorithm is adopted in [27] to solve the data-intensive DAG-SP. Data file and computation tasks are both modelled as vertices of DAG, and data reading and writing procedures are considered. The chromosome of the proposed algorithm consists of task and data assignment and the execution order of tasks. Local search procedures, path relinking and move-file heuristic are used to enhance the algorithm. A tailored Biogeography-Based Optimization (BBO) is adopted in [28] to solve DAG-SP, where the "task-processor" assignment array is used to encode the problem and heuristic strategy is applied to determine the task processing sequence. Migration and mutation operators are used to improve the individuals. A particle swarm optimization (PSO) algorithm is proposed in [29] for multi-objective DAG-SP to optimize makespan, load-balancing, resource utilization and speed up objectives. One task sequence is produced and kept and PSO evolves the processor selection sequence based on the task sequence. Furthermore, a deep reinforcement learning enhanced biased random-key genetic algorithm (BRKGA) is proposed in [30] to optimize both running time and peak memory usage. Task-processor affinities, task scheduling priorities and data transfer priorities are all considered as encoding chromosome. Deep reinforcement learning enhanced initialization observably improves the performance of BRKGA. In summary, task processing sequence and processor assignment rule are generally used to encode the problem; list heuristics can be adopted as the initial solutions of evolutionary algorithms; tasks are assigned to processors by heuristic-based policy.

From these existing research works, it is significant to achieve well-performance solutions within acceptable time by abstracting knowledge from DAG application properties and collecting information during data-driven search. Estimation of distribution algorithm (EDA) [31–33] is an appropriate algorithm to solve such a problem since its probability model can represent the precedence constraints between tasks, simulate the distribution of search space and record search information during evolution process. So far, EDA performs well on combinatorial optimization problems [34] including scheduling problems [35,36]. With the above motivation, a tailored permutation-based EDA is adopted in this paper to generate task processing permutations. For permutation-based optimization problems, path relinking [37] is used to generate paths between two solutions, and the solutions on these paths are evaluated as a new combination of solutions. Thus, to enhance the exploitation of EDA, a path relinking [37,38] enhanced local search mechanism is embedded into EDA to accelerate convergence. With the above special designs, DAG-SP is solved effectively by our proposed algorithm, which is demonstrated via simulation experiments with benchmark datasets and real-world graphs.

The remaining contents are organized as follows. Section 2 formulates the problems by providing DAG application and scheduling computational models. Then, four decoding methods and the proposed hybrid decoding mechanism are presented in Section 3. Section 4 introduces the proposed algorithm, including EDA and path relinking enhanced local intensification mechanism. In Section 5, numerical comparisons are given to demonstrate the effectiveness of our proposed algorithm. Some conclusions are presented in Section 6, and managerial implication and future work of this study are summarized in Section 7.

2. Problem statement and formulation

2.1. DAG application model

A four-tuple vector <$\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{w}, \boldsymbol{D}$> is used to model a DAG named G, where $\boldsymbol{V}(G) = \{V_j | j = 0,1,\ldots, n-1\}$ denotes a set of non-communication task vertexes and $\boldsymbol{w}(G) = \{w_i\}$ denotes the set of corresponding task workload. $\boldsymbol{E}(G) = \{e_{ij} |\ i, j = 0, 1, \ldots, n-1\}$ represents a set of directed edges of G where e_{ij} represents the directed edge between vertex i and j, which means that there is a precedence constraint between task i and j and $D(G) = \{D(i, j)\}$ denotes the transfer data size of corresponding edge.

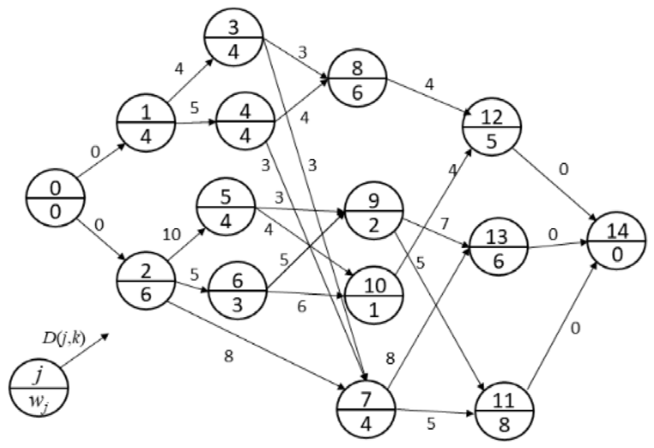


Fig. 1. A DAG containing 14 tasks.

For task j, if there exists a direct path from vertex k to j, then task k is defined as a precedent task of task j and pred(j) denotes the task set consisting of all its precedent tasks. Similarly, if vertex j is reachable to k, then task k is defined as a successive task of task j and succ(j) denotes the task set consisting of all its successive tasks. For ease of description, a virtual entry vertex (node 0) and a virtual exit vertex (node $n-1$) are set in DAG as the virtual parent vertex of all tasks without precedent tasks and the virtual child vertex of all tasks without successive tasks correspondingly. An example of DAG is shown in Fig. 1.

In addition, to measure the impact of communication latency on computing performance, communication to computation ratio (CCR) is introduced as (1). For an application, high CCR implies communication-intensive and low CCR implies computation-intensive.

$$CCR(G) = \frac{\sum_{i,j} D(i,j)/|E|}{\sum_j w_j/n} \tag{1}$$

2.2. System model

Multi-processor system consists of a set of full connected m heterogeneous processors. The processors are able to process at most one task during a certain time period. For each processor i, its processing speed is v_i. The computation time of task j processed on processor i is w_j/v_i. For each pair of processors i_1 and i_2, the bandwidth between each other is $B(i_1, i_2)$. The communication time between task j_1 and j_2 is $D(j_1, j_2)/B(i_1, i_2)$ where i_1 and i_2 denotes the processors of tasks j_1 and j_2. Specially, the communication time is 0 if the tasks are processed on the same processor as the transfer data can be read from memory. An example system is shown in Fig. 2.

2.3. Scheduling model and its notations

DAG-SP aims at optimizing the completion time by determining the task-processor assignment and task processing order. It is supposed that tasks are non-preemptive and any task can be processed only after all of its precedent tasks are finished and input data is received.

To formulate the MILP model for DAG-SP, a set of notations is given as Table 1.

$$\min \max_j C_j \tag{2}$$

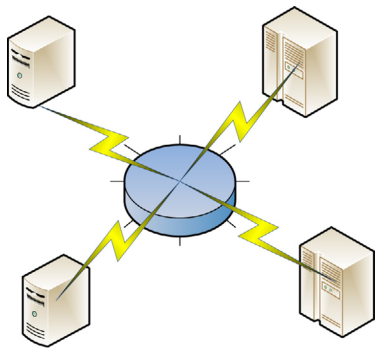


Fig. 2. A fully connected computing system with 4 heterogeneous processors.

Subject to:

$$\sum_{r=1}^{n}\sum_{i=1}^{m} x_{j,i,r} = 1, \forall j \tag{3}$$

$$\sum_{j=1}^{n} x_{j,i,r} \le 1, \forall i, r \tag{4}$$

$$\sum_{j_1=1}^{n} x_{j_1,i,r} \ge \sum_{j_2=1}^{n} x_{j_2,i,r+1}, \forall r \le n-1, i \tag{5}$$

$$C_{j_1} - C_{j_2} + M \cdot \left(2 - x_{j_1,i,r+1} - x_{j_2,i,r}\right) \ge \frac{w_{j_1}}{v_i}, \forall j_1, j_2, r \le n-1, i \tag{6}$$

$$C_j + M \cdot \left(1 - x_{j,i,1}\right) \ge \frac{w_j}{v_i}, \forall j, i \tag{7}$$

$$C_{j_2} - C_{j_1} \ge \sum_{r=1}^{n}\sum_{i_2=1}^{m} x_{j_2,i_2,r} \cdot \frac{w_{j_2}}{v_{i_2}} + \sum_{i_1=1}^{m}\sum_{i_2=1}^{m}\sum_{r=1}^{n}\sum_{t=1}^{n} x_{j_1,i_1,r} \cdot x_{j_2,i_2,t} \cdot \frac{D(j_1,j_2)}{B(i_1,i_2)}, \forall j_1 \to j_2 \tag{8}$$

where (2) defines the optimization objective, i.e., makespan; Constraints (3)–(8) guarantee feasible schedule. To be specific, (3) ensures that each task can be processed once and only once; (4) ensures that for a given position of a certain processor, no more than one task can be processed in case of avoiding the time conflict; To strictly meet the task queue, (5) means that (r+1)-th task does not exist in the queue if there is no the r-th task on the certain processor; For the tasks processed on the same machine, (6) ensures that the (r+1)-th task cannot be processed before the completion of the r-th task. In addition, (7) guarantees that the completion time of the first task on each processor is not less than its computation time. Considering the precedence constraints between the tasks, (8) ensures the tasks cannot be processed before receiving the data from its precedent tasks.

3. Encoding method and hybrid decoding mechanism

3.1. Encoding method

To represent the problem, the task processing permutation π is adopted to encode a solution as Fig. 3, where $\pi_i \in \{0, 1, \ldots, n-1\}$ denotes the ith task to be processed. As there is a virtual entry vertex and a virtual exit vertex, $\pi_0 = 0$ and $\pi_n = n-1$ is settled for the convenience of producing the permutations. Other tasks in DAG are ranked to generate the task processing permutation without violating the precedence constraints.

Table 1
Notations for the DAG-SP.

	Notation	Implication
Decision variables	$x_{j,i,r}$	{0, 1}, $x_{j,r,i} = 1$ denotes task j is the rth task processed on processor i, otherwise, $x_{j,i,r} = 0$. $i, j = \{0,1,\ldots,n-1\}$, $r = \{1,\ldots,n\}$
	C_j	$C_j \geq 0$, the completion time of task j, $j = 1\ldots, n$;
Problem data	n	Number of tasks in the DAG;
	m	Number of processors;
	$B(i_1, i_2)$	Bandwidth between processor i_1 and i_2;
	w_j	Computation workload of task j;
	v_i	Processing speed of processor i;
	$D(j_1, j_2)$	Inter-task data size between task j_1 and j_2;
Others	M	A very large constant number;
	$j_1 \rightarrow j_2$	Task j_1 is precedent of task j_2, i.e., there exists a direct path from j_1 to j_2 in DAG.

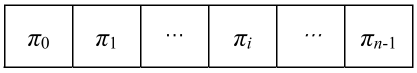


Fig. 3. Task processing permutation.

Definition 1 (*Valid Permutation*)**.** For a permutation π, if there does not exist DAG precedence constraint between task pair (π_j, π_i), $\forall i < j \in \{1, \ldots, n-1\}$, then π is defined as a valid permutation of the certain DAG. As a consequence, when the tasks are scheduled according to this permutation, the schedule will satisfy the topological order of DAG. For DAG in Fig. 1, $\pi^* =$ (0, 2, 1, 3, 4, 8, 7, 5, 6, 9, 11, 10, 13, 12, 14) is a valid permutation.

3.2. Decoding mechanism

Once a valid permutation π is achieved, a feasible schedule can be obtained via decoding methods. In this work, four existing list scheduling methods, i.e., EFT, HEFT, OCT+EFT(OEFT) and OCT+HEFT(OHEFT) are used to form an efficient decoding mechanism. Firstly, these four decoding methods are illustrated respectively as Figs. 4–7 with $\pi^* =$ (0, 2, 1, 3, 4, 8, 7, 5, 6, 9, 11, 10, 13, 12, 14), where two processors are considered and the speed values are 1 and 1.25.

EFT Tasks are assigned to the end of each processor to achieve to the earliest finish time. For task j, its processor is chosen as (11), where $FT(j,i)$ denotes the finish time of task j on processor i as (10). $EST(j)$ denotes the earliest starting time of task j as (9) shows, where m_k denotes the processor assignment of task k and C_i denotes completion time of processor i. EFT(π) denotes the schedule decoded by EFT based on π.

$$EST\ (j) = \max_{k \in pred(j)} \left(C_j + \frac{D\,(k, j)}{B\,(i, m_k)} \right) \tag{9}$$

$$FT\ (j, i) = \max\{EST(j), C_i\} + \frac{w_j}{v_i} \tag{10}$$

$$m_j = \underset{i}{\text{argmin}}(FT(j, i)) \tag{11}$$

HEFT: According to HEFT, tasks are assigned to any available idle time slot and the last period of processor i defined as (C_i, $+\infty$). For task j and processor i, its available idle time slot set *avidle* contains idle periods satisfying (12), where ST_k and CT_k denotes the starting and ending time of the kth idle. In this way, finish time of task j on processor i is calculated as (13) and the processor with minimum finish time is selected according to (14). HEFT(π) denotes the schedule decoded by HEFT based on π.

$$\max\,(EST\ (j)\ , ST_k) + \frac{w_j}{v_i} < CT_k \tag{12}$$

$$HFT\ (j, i) = \min_{k \in avidle} \{\max\,(EST\ (j)\ , ST_k) + \frac{w_j}{v_i}\} \tag{13}$$

$$m_j = \underset{i}{\text{argmin}}(HFT(j, i)) \tag{14}$$

Compared to EFT, insert-based policy is adopted in HEFT. Suppose S represents a schedule, we define $\boldsymbol{I(S)}$ as the set of tasks inserted forward into idle periods during HEFT decoding procedure. For each inserted task, define its **following task** as the first task after the idle period in the scheduling solution. In Fig. 8, I(HEFT(π^*)) = {5,6,9,10} and their following task are task 7. Obviously, when S = EFT(π), S' = HEFT(π), if $I(S')=\varnothing$, then all the tasks are assigned to the end of processors during the decoding procedure and for all i and j, $HFT(j, i)$=$FT(j, i)$. Then, $S = S$' and $C_{max}\,(S) = C_{max}\,(S')$.

OEFT: OCT values [15] can be calculated iteratively as (15) shows. The processor assignment is determined as (16) shows, where $FT(j, i)$ can be calculated by (9)–(10).

OEFT(π) denotes the schedule decoded by OCT method with EFT policy based on π.

$$OCT\ (j, i) = \max_{k \in succ(j)} \min_{l} \left\{ OCT\ (k, l) + \frac{w_k}{v_l} + \frac{D\,(j, k)}{B\,(i, l)} \right\} \tag{15}$$

$$m_j = \underset{i}{\text{argmin}}(FT\ (j, i) + OCT(j, i)) \tag{16}$$

OHEFT: The processor selection is determined as (17) shows, where $HFT(j, i)$ can be calculated by (12)–(13). OHEFT(π) denotes the schedule decoded by OCT method using HEFT policy based on π.

$$m_j = \underset{i}{\text{argmin}}(HFT\ (j, i) + OCT(j, i)) \tag{17}$$

It can be seen, for the same permutation, different decoding mechanisms lead to different schedules. HEFT and OHEFT are more efficient for the mentioned permutation because of its insert-based policy.

3.3. Hybrid decoding mechanism

Based on the above decoding methods, it can be seen that, the idle periods are shorten or removed by the insert-based policy. It is possible to decrease makespan through the insert-based policy. However, it is time consuming to traverse the idle periods when assigning each task. The efficient schedule produced by HEFT cannot be presented in the task processing permutations, which leads to the difficulty for our EDA in learning the effective structure of permutations.

Thus, to take advantage the high quality of HEFT schedule solution and low complexity of EFT, it is efficient adopting EFT to achieve same scheduling solution as HEFT, i.e., $\pi' =$ EFT^{-1} (HEFT(π)) where the schedule solution encoded by EFT based on π' is the same with HEFT(π). HEFT2EFT(π) is designed to achieve the objective, which includes two methods: transfer(π) is designed to achieve π' in permutation space and produce(S) is

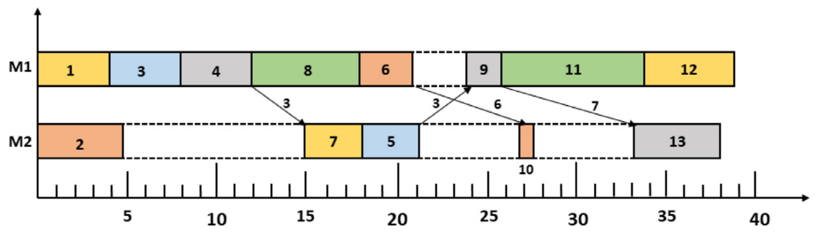


Fig. 4. Gantt graph of EFT(π^*), $C_{\max} = 39.4$.

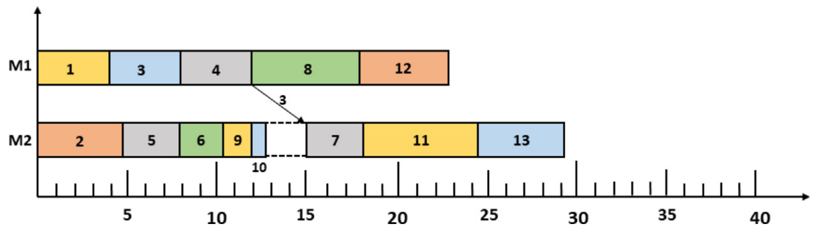


Fig. 5. Gantt graph of HEFT(π^*), $C_{\max} = 29.4$.

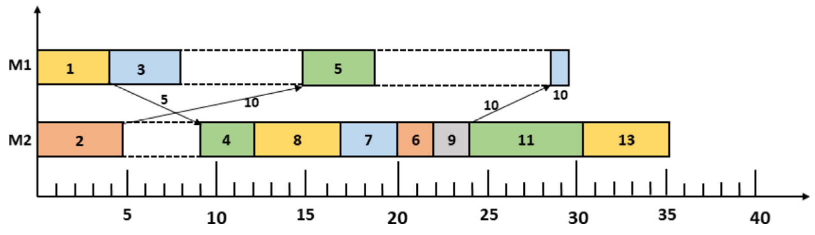


Fig. 6. Gantt graph of OEFT(π^*), $C_{\max} = 35.4$.

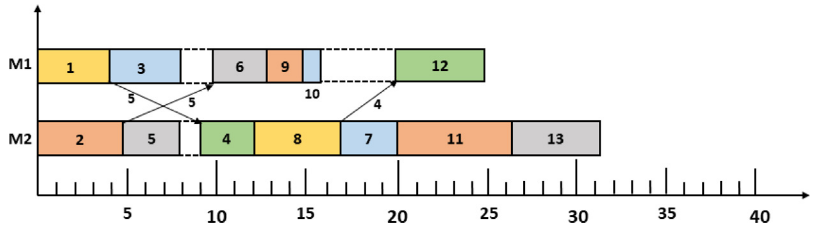


Fig. 7. Gantt graph of OHEFT(π^*), $C_{\max} = 31.4$.

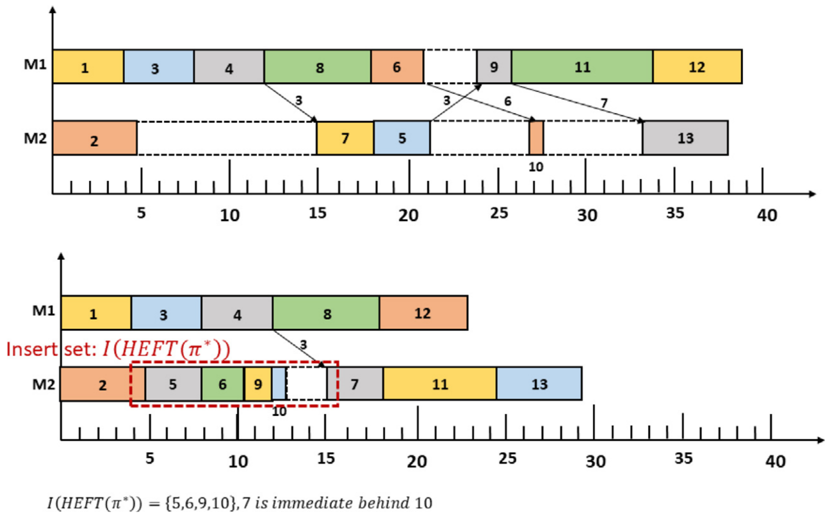


Fig. 8. Illustration of $I(S)$.

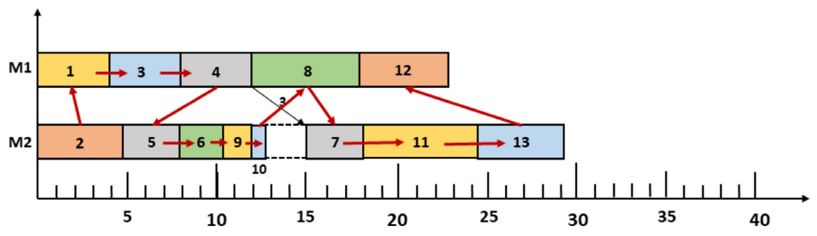


Fig. 9. Illustration of produce(S).

used to achieve π' from the schedule solution. When $I(S) \neq \varnothing$, transfer(π) is adopted which transfers the permutation through swapping and moving tasks in $I(S)$ forward. The detailed pseudo code is presented as Method 1.

Method 1: transfer(π)

```
Input: task permutation $\pi$, $S$=HEFT($\pi$)
For each task $\pi_i$ in $I(S)$ and its following task $\pi_w$:
    Insert $\pi_i$ between $\pi_w$ and $\pi_{w-1}$ to achieve $\pi$'
    If $\pi$' is valid:
        If $C_{\max}$(EFT($\pi$')) > $C_{\max}$(HEFT($\pi$)):
            Return FALSE;
        Else if $C_{\max}$(EFT($\pi$')) = $C_{\max}$(HEFT($\pi$)):
            Return TRUE, $\pi$';
        Else
            $S$ = HEFT($\pi$');
            Update $I(S)$;
            transfer($\pi$');
        End If
    Else
        Return FALSE;
End For
Output: $\pi$', if makespan of EFT($\pi$) is not worse than HEFT($\pi$)
```

If transfer(π) returns FALSE, produce(S) is used to produce π' from the scheduling solution as Fig. 9 shows. For S, the task lists on each processor is recorded as Tlist(S), where Tlist(S)[i][r] represents the r-th task on the ith processor of S. As produce(S) is a mapping from S to π', available task set **A** is defined as the tasks can be used to generate π. For example, **A** = {1,2} at the beginning of the procedure. If 2 is determined as the first element of π', then **A** = {1,5} as task 5 is the task scheduled next to task 2. The detailed pseudo code is listed as Method 2.

Method 2: produce(S)

```
Input: $S$, A={0}: available task set, $S$': empty scheduling solution;
For the $i$-th valid task $j$ in A:
    //valid task: its parent tasks have been scheduled;
    Assign $j$ to processor $m_j$ by EFT in $S$';
    If start time and end time of $j$ in $S$' are the same with $S$:
        Append $j$ to the end of $\pi$';
        Add the task assignment to $S$' and remove $j$ from A;
        Add the next task in Tlist[$m_j$] to A;
        $i$--;
    End If
End For
If the length of $\pi$' is less than $n$:
    Return FALSE;
Else
    Return TRUE;
End
```

HEFT2EFT is presented as Method 3, which is consisting of Method 1 and 2. During the local intensification, the individuals labelled 0 is decoded with EFT and ones labelled 1 is decoded with HEFT. The whole hybrid decoding mechanism consisting four heuristic-based decoding methods is listed as Procedure 1.

Method 3: HEFT2EFT(π, S)

```
Input: $\pi$, $S$ = HEFT($\pi$), $I(S)$
If $I(S)$ is empty:
    Label = 0;
    Return $\pi$' =$\pi$;
Else if transfer($\pi$) is TRUE:
    Label = 0;
    Return $\pi$';
Else
    If produce($S$) is TRUE:
        Label = 0;
        Return $\pi$';
    Else:
        Label =1;
    End If
End If
```

Procedure 1: Hybrid decoding mechanism(π)

```
Input: $\pi$
Result=[$C_{\max}$(EFT($\pi$)),$C_{\max}$(HEFT($\pi$)),$C_{\max}$(OEFT($\pi$)),$C_{\max}$(OHEFT($\pi$))]
If Result[0] is the best:
    Label = 0;
    Return $\pi$;
Else if Result[1] is the best:
    HEFT2EFT($\pi$, HEFT($\pi$))
Else if Result[2] is the best:
    Label = 2;
    Return $\pi$;
Else:
    Label = 3;
    Return $\pi$;
End If
```

4. The proposed algorithm

4.1. EDA for task processing sequence production

EDA builds a probability model to simulate the distribution of solution space, where its sampling and updating method are used to learn, evaluate and update the possibility model from the structure of elite individuals. Standard EDA procedure can be summarized as follows: Firstly, probability model P is built. A population is generated by sampling P and then P is updated in accordance to the elite individuals selected from the population. Then, a new population is generated by sampling the updated P until the stopping criterion is met.

Probability model and its initialization

A problem knowledge driven probability model is designed which adopts the information of the problem and existing heuristic solutions. Since there exist precedence constraints between the tasks, a relative position probability is designed as (18) to produce task processing permutation, 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, $p_{i,j}(g)+p_{j,i}(g) = 1$. When $i = j$, $p_{i,j}(g)$ is set as 1.

$$P(g) = \{p_{k,j}(g)\}_{n\times n} \tag{18}$$

The probability model is initialized according to the following rule. If task i is precedent of task j, $p_{i,j} = 1$, else 0 as (19).

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

For task pairs (i, j) without precedent constraints, the probability value is initialized with the best existing list scheduling permutation. An indicative function (20) is used to represent the structure of a good solution and the probability is initialized as (21) where $\alpha \in (0, 1)$ denotes the learning rate.

$$I_{i,j} = \begin{cases} 1, \text{task } i \text{ is before task } j \text{ in the best heuristic list} \\ -1, \text{otherwise} \end{cases} \tag{20}$$

$$p_{i,j}(0) = 0.5 + \alpha \times I_{i,j}, \text{ there is no precedent constraint between } i \text{ and } j \tag{21}$$

Sampling and updating method

A sampling method is designed for the relative position probability model to produce individuals. The pseudo code is given as follows. For each position not determined in the permutation, a probability array is created which denotes the probabilities that tasks in available set should be assigned to the position. $sp(t)$ denotes the probability vector of task t is set in front of all other available tasks. Then the array is normalized and sampled the next element of the permutation by the roulette wheel method. In addition, after a certain task is determined in the permutation, its child tasks are checked to update the available task set. If the parent tasks of a certain task have been determined in the processing permutation, then it is defined as an available task and pushed into the available task set. In this way, the permutations produced by EDA probability model can be guaranteed to be valid and schedules decoded from permutations by EDA satisfy the topological order of the DAG.

```
Method 4: EDA sampling
  Input: A is a task set, A={0}; $P(g)$; $k$=0.
  While A is not empty:
    For each task $t$ in $A$
      $sp(t) := \prod_{i \in A} p_{t,i}(g)$;
    End For
    Normalize probability array $sp$;
    $\pi_k$ = RouletteWheel($sp$)
    Check child tasks of $\pi_k$, and if its parent tasks are determined in $\pi$, push it into A;
    $k$++;
  End While
  Output: $\pi$
```

After sampling P, all the individuals are sorted according the increasing order of makespan. The first $Q = N \times \eta\%$ individuals are considered as the elite population, where N denotes the size of population and $\eta\%$ is the elite population proportion. The updating scheme is set as (22) and (23). For each elite individual, the permutation structure is recorded as $I_{i,j}^{k}(g)$ $(k = 1,\ldots, Q)$ as (23). Through this indicator, the frequency of a certain task pair is calculated and used to update the probability value based on population based incremental learning method (PBIL) [39].

$$p_{i,j}(g+1) = (1-\alpha) \times p_{i,j}(g) + \alpha \times \sum_{k} I_{i,j}^{k}(g)/Q \tag{22}$$

$$I_{i,j}^{k}(g) = \begin{cases} 1, \text{task } i \text{ is before task } j \text{ in the } k\text{th individual of the } g\text{th generation} \\ 0, \text{otherwise} \end{cases} \tag{23}$$

4.2. Path relinking enhanced local intensification

Path relinking is one of methods to explore the path between elite solutions, which aims at sharing the promising permutation structure [40]. Three essential elements: rules for building reference set, initial and guiding solutions and a neighbourhood structure for moving along the path, are necessary for path relinking procedure. Based on the neighbourhood structure, a path is built from the initial solution to the guiding solution based on neighbourhood structure. In this way, a set of temporary solutions are produced, and it is possible to achieve better solutions within the temporary solutions.

To take advantage of the knowledge of existing DAG-SP research works, solutions produced by EDA are chosen as the corresponding starting solution to learn from the guiding solution, which is set as the best heuristic solution or existing best solution till now. Path relinking insert and swap operators are defined and set as the neighbourhood structures. The whole procedure is illustrated as Fig. 10 where for all the elite solutions produced by EDA, "path relinking insert" operator is adopted to build the path and generate temporary solutions, and for the best solution of the current population, "path relinking swap" operator is adopted. In this section, these two neighbourhood structures as well as a corresponding property is presented at first. Then, the detailed pseudo code of the path relinking enhanced local intensification is introduced.

Definition 2 (*Path Relinking Insert*). $\boldsymbol{\pi}^{\mathbf{I}}_{\mathbf{rs}}$ is defined as a set of temporary permutations generated from π_r to π_s by "path relinking insert" operator. For starting permutation π_r and guiding permutation π_s, each element of permutations π_r and π_s is compared and if $\pi_{s,i} \neq \pi_{r,i}$, π_r is traversed to locate $\pi_{s,i}$, and is inserted to the ith position of π_r, as method 5: PRInsert(π_r, π_s) and Fig. 11(a) present. In this way, a temporary permutation is generated and added to $\boldsymbol{\pi}^{\mathbf{I}}_{\mathbf{rs}}$. The whole procedure is defined as "path relinking insert". Obviously, $|\boldsymbol{\pi}^{\mathbf{I}}_{\mathbf{rs}}| \leq n$ and when π_r and π_s are completely reversed, the equality holds.

```
Method 5: PRInsert($\pi_r$,$\pi_s$)
Input: Starting permutation $\pi_r$, guiding permutation $\pi_s$, $\Pi^{I}_{rs}$ = {}
For $i$ = 1 to $n$
  If $\pi_{r,i} \neq \pi_{r,i}$
    $\pi'$ = $\pi_r$
    For $j$ = $i$ to $n$+1
      If $\pi_{r,j}$ = $\pi_{s,i}$
        $\pi'_i$ = $\pi_{s,i}$
        For $k$ = $i$+1 to $j$
          $\pi'_k$ = $\pi_{r,k-1}$
        End For
        Add $\pi'$ into $\Pi^{I}_{rs}$
        Break
      End if
    End For
  End If
  $i$++
End For
Output: $\Pi^{I}_{rs}$
```

Definition 3 (*Path Relinking Swap*).$\boldsymbol{\pi}^{\mathbf{S}}_{\mathbf{rs}}$ is defined as a set of temporary permutations generated from π_r to π_s by "path relinking swap" operator. For starting permutation π_r and guiding permutation π_s, each element of permutations π_r and π_s is compared and if $\pi_{s,i} \neq \pi_{r,i}$, π_r is traversed to locate $\pi_{s,i}$, and then swap it with its prior element. In this way, a temporary permutation is generated and added to $\boldsymbol{\pi}^{\mathbf{S}}_{\mathbf{rs}}$ until $\pi_{s,i}$ is moved to the ith position

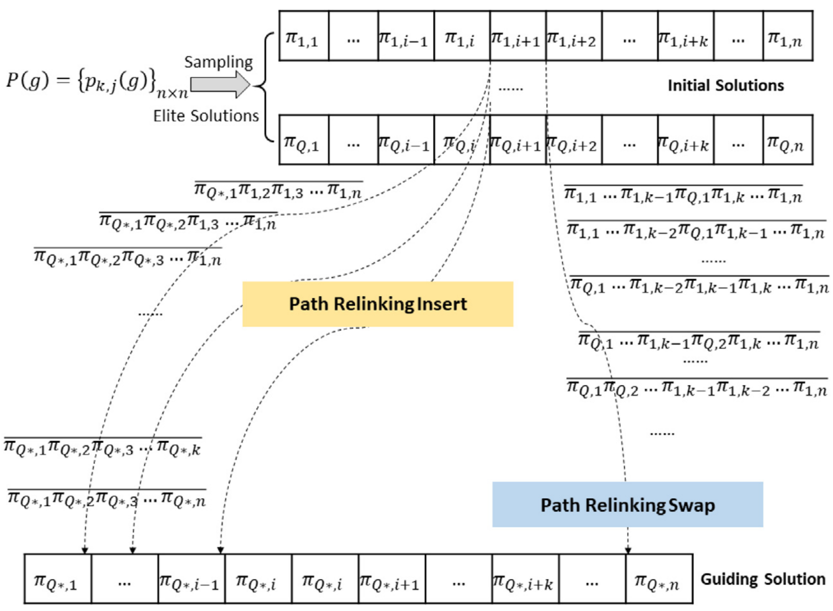


Fig. 10. Illustration of path relinking enhanced local intensification.

of π_r, as method 6: PRSwap(π_r, π_s) and Fig. 11(b) present. The whole procedure is defined as "path relinking swap". Obviously, $|\boldsymbol{\pi}^{S}{}_{rs}{}^{S}{}_{rs}| \le n(n+1)/2$ and when π_r and π_s are completely reversed, the equality holds.

```
Method 6: PRSwap($\pi_r$,$\pi_s$)
Input: Starting permutation $\pi_r$, guiding permutation $\pi_s$, $\Pi^S{}_{rs} = \{\}$
For $i$ = 1 to $n$
    If $\pi_{r,i} \neq \pi_{r,i}$
        $\pi' = \pi_r$
        For $j = i$+1 to $n$
            If $\pi_{r,j} = \pi_{s,i}$
                For $k = j$ to $i$
                    $\pi'_k = \pi_{r,k-1}$
                    $\pi'_{k-1} = \pi_{r,k}$
                    Add $\pi'$ into $\Pi^S{}_{rs}$
                End For
                break
            End If
        End For
    End If
    $i$++
End For
Output: $\Pi^S{}_{rs}$
```

Property 1. *If π_r and π_s are valid permutations, permutations in the defined sets: $\boldsymbol{\pi}^{I}{}_{rs}{}^{I}{}_{rs}$ and $\boldsymbol{\pi}^{S}{}_{rs}{}^{S}{}_{rs}$, are valid.*

Proof. These permutations into three sets as Fig. 11 shows, where ϕ_1 is the beginning set of same elements, which is green-coloured in Fig. 11. $\pi_{r,i}$ and $\pi_{s,i}$ are the first different element of these two permutations and if $\pi_{s,j} = \pi_{r,ik}$, then ϕ_2 includes the ith to the (i+k)-th elements of these permutations, which is yellow coloured in Fig. 11. At last, ϕ_3 is consist of the rest elements, which is pink coloured.

As guiding permutation π_s is valid, we have $prec(\pi_{s,i}) \subseteq \phi_1$, and for starting permutation π_r, as the elements ϕ_1 are the same, we have $\forall \beta \in \phi_2$, $\beta \notin prec(\pi_{s,i})$. Thus, inserting $\pi_{r,i+k} = \pi_{s,i}$ to the position between ϕ_1 and ϕ_2 will not violate the precedence constraints in DAG. For ϕ_3, the topological order between $\phi_1 \cup \phi_2$

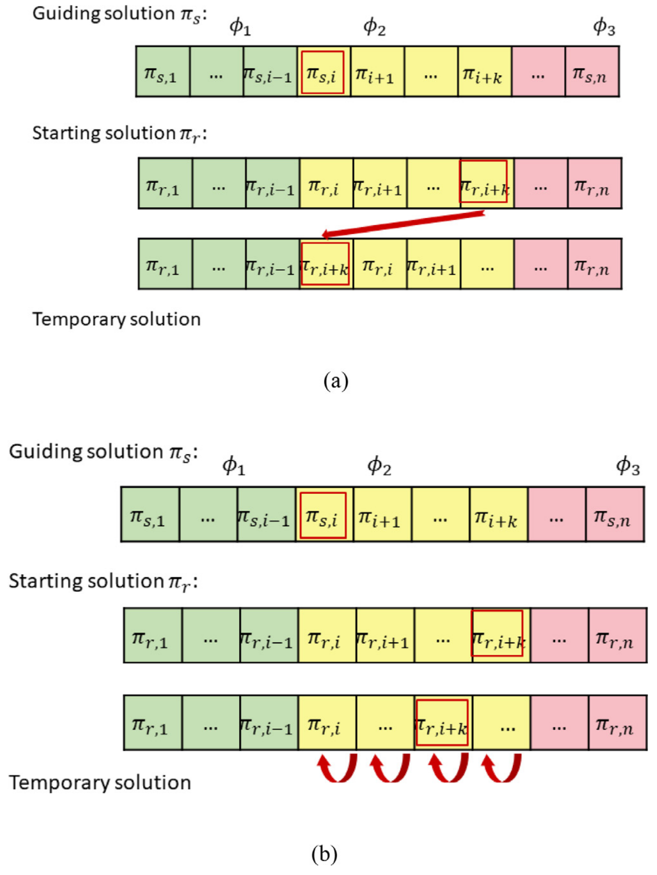


Fig. 11. Illustration of path relinking neighbourhood structures (a) Path relinking insert; (b) Path relinking swap.

and ϕ_3 does not be altered. In this way, it can be proved that permutations in π^{I}_{rs} are valid. The case of swap operator can be proved similarly. □

Based on this property, all the permutations produced by path relinking insert and swap can be proved valid. Thus, there is no need to check if the permutation is valid during the local intensification. The details of local intensification is introduced in Procedure 2 and detailed path relinking method is illustrated in Method 5, where X ={EFT, HEFT, OEFT, OHEFT} denotes the selected decoding method by the above hybrid decoding mechanism. To improve the performance of elite solutions, insert operator is carried on elite individuals at first, and the individuals are updated with better solutions. Then, the path relinking swap operator is adopted for the best individual.

```
Method 7: pathrelinking($Y$, $\pi_s$, $L$)
Input: Path relinking operator $Y$, guiding solution $\pi_s$ and elite individual $L$
$\pi_r$:= $L$.permutation; $I$ := $L$;
Produce $\Pi^{Y}_{rs}$ based on Method 5 or 6
For each $\pi$ in $\{\Pi^{Y}_{rs}\}\backslash\pi_s$:
    Choose decoding method $X$ for $\pi$ according to label of $L$;
    If X($\pi$)< $L.makespan$
        $I.permutation = \pi$
        $I.makespan = X(\pi)$
    End If
End For
If $I.makespan < L.makespan$
    Return $I$;
Else
    Return null;
End If
Output: better solution $I$
```

```
Procedure 2: Local Intensification
Input: $bIndi$ := best individual till now;
       $gbIndi$ := elite individuals in this generation of population;
$\pi_s$ := $bIndi.permutation$;
For each $L$ in $gbIndi$:
    $I$ := pathrelinking($Insert$, $\pi_s$, $L$)
    If $I$ != null
        Replace $L$ with $I$;
    End If
End For
Update $gbIndi$, $bIndi$, $\pi_s$
$I$ := pathrelinking($Swap$, $\pi_s$, $gIndi$[0])
If $I$ != null
    Replace $gbIndi$ with $I$;
End If
```

4.3. Flowchart and complexity analysis

Based on above introduction, our proposed algorithm is summarized in Fig. 12. At first, HEFT and PEFT heuristics are carried out where the best one is chosen to initialize the probability model. After that, EDA is adopted to produce the task processing permutation as the right of Fig. 12. Please refer to the corresponding pseudo codes for the details of the related procedures.

According to the flowchart, the time complexity of our proposed algorithm is analysed as follows:

Hybrid decoding mechanism: transfer(π) has an $O(n^2)$ complexity and the complexity of produce(S) is the same as HEFT, where HEFT has an $O(n^2\cdot m)$ complexity. It could be summarized that hybrid decoding mechanism complexity is of $O(n^2\cdot m)$;

EDA evolution: EDA sampling has an $O(n^3)$ complexity and other operator complexity is $O(n^2)$;

Local intensification: Path relinking insert operator is carried on elite population, and its complexity turns to $O(n^3\cdot m\cdot Q)$. Path relinking swap operator is carried on the best individual, and the complexity is $O(n^4\cdot m)$. It is worth mentioned that if the individual is labelled as 0 or 2, EFT or OEFT is adopted to decode the permutation, its complexity decreases to $O(n^2\cdot m\cdot Q)$ and $O(n^3\cdot m)$.

Thus, the complexity of the proposed algorithm is $O((N\cdot n^3\cdot m+ n^3\cdot m\cdot Q+ n^4\cdot m)\,G)$ where G denotes the number of iteration. Considering $N<n$, the total algorithm complexity is of the order $O(n^4\cdot m\cdot G)$.

5. Simulation

To evaluate the performance of our proposed algorithm, we compare it with several algorithms: two list scheduling heuristic methods and two evolutionary algorithms tailored for DAG-SP.

For fair comparison, the algorithms are all coded in C++ and run on the same computer with Intel Core i5 CPU/3.20 GHz and 16 GB RAM. For the evolutionary algorithms, the stopping criterions are set the same. In this section, the benchmark datasets, comparison algorithms and parameter settings are introduced at first. Then, the comparison with solutions of Gurobi solver are presented to review our proposed model. In addition, comparison results on randomly generated DAGs and real world instances with different scales of tasks, processors and different values of *CCR*s demonstrate the effectiveness of the proposed algorithm.

5.1. Benchmark datasets, comparison algorithms and parameter settings

Two benchmark datasets are adopted in this paper: A commonly used DAG dataset [41] (http://www.kasahara.elec.waseda.ac.jp/schedule/) and DAG applications from real world. To supplement the communication data information of [41], new datasets are produced in accordance to different *CCR* values. Data communication size is produced following normal distribution with the average value of a certain *CCR* according to (1). In addition, the settings of {n, m, *CCR*} can be referred to Table 2. For each combination of {n, m, *CCR*}, 60 independent cases are produced.

In addition, DAG applications from real world problems are carried out as the test instances: Gauss elimination algorithm (GE) [42] and Fast Fourier Transformation (FFT) [43] and scientific workflows [44]. The general structures of these DAGs are presented as Figs. 13 and 14.

The detailed datasets can be referred in https://www.researchgate.net/publication/352697650_DAG-benchmark.

For the comparison algorithms, HEFT [18], PEFT [15], GA[23] and PSO[29] are selected as. HEFT is considered as the best DAG-SP list scheduling heuristic in terms of robustness and performance compared with around 20 heuristic methods [45]. In this simulation test, the minimum makespan among HEFT-t-level, HEFT-b-level and HEFT-tb-level is adopted as its result. Compared to HEFT, PEFT considers the impact of task assignment on its children nodes. Besides, GA is an efficient evolutionary algorithm designed for solving DAG-SP, which adopts existing heuristic permutations to initialize the population. Similar with GA, PSO is also an evolutionary algorithm tailored for DAG-SP, which considers more than one objectives simultaneously.

During the simulation, two parameters of our proposed EDA: population size (N) and learning rate (α) are set as the parameters in the simulation experiments based on our previous works [35]. The population size and stopping criterion are set the same for EDA, GA and PSO. The detailed settings are listed as follows.

5.2. Comparison to Gurobi on small scale instances

To review the MILP model presented in Section 2.3 and evaluate the performance of our proposed algorithm on solving small-scale problems, math solver Gurobi (Version 9.0.0) is adopted to

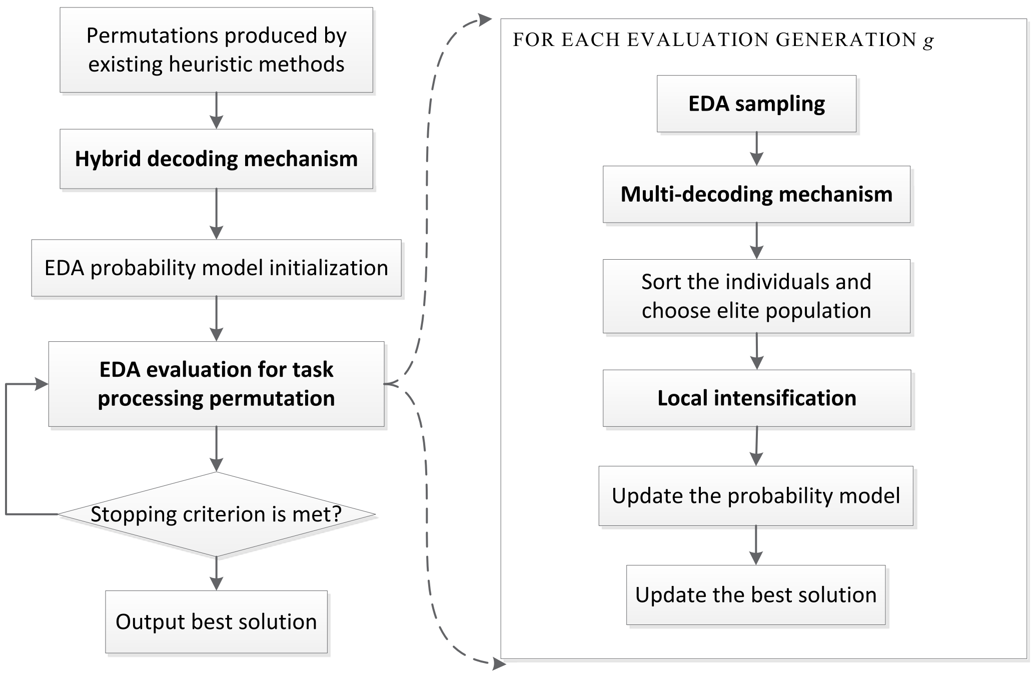

Permutations produced by existing heuristic methods
Hybrid decoding mechanism
EDA probability model initialization
EDA evaluation for task processing permutation
Stopping criterion is met?
Output best solution
FOR EACH EVALUATION GENERATION g
EDA sampling
Multi-decoding mechanism
Sort the individuals and choose elite population
Local intensification
Update the probability model
Update the best solution

Fig. 12. Flow chart of the proposed algorithm.

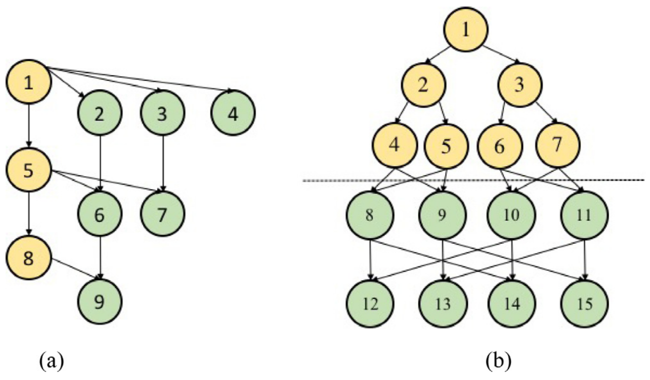


Fig. 13. DAGs from real world applications (a) Gauss elimination algorithm; (b) Fast Fourier transformation.

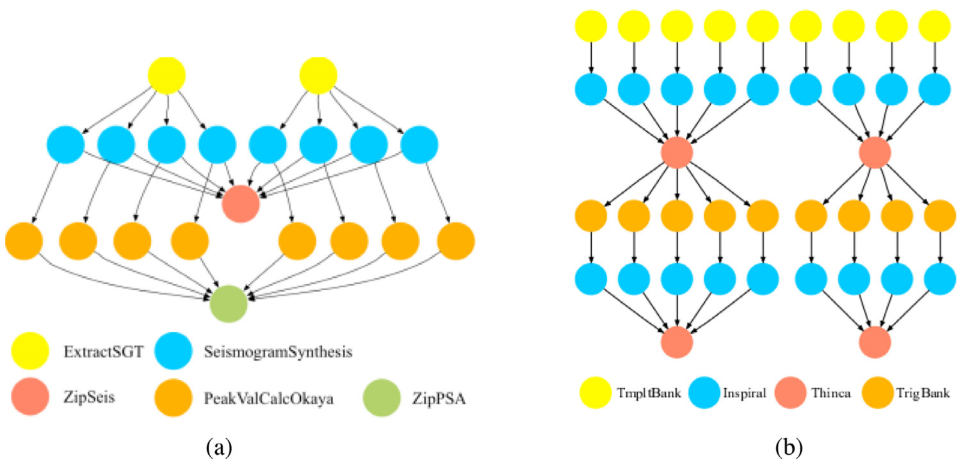


Fig. 14. DAGs from scientific workflows [44] (a) CyberShake workflow; (b) LIGO workflow.

solve the MILP model and construct an accurate algorithm on DAG-SP. Small-scale DAGs of GE with $n = 14$, CCR={0.1, 0.5, 1.0, 5.0} and m={2,4,8} are produced for the comparison.

For each instance, the proposed EDA is run 20 times independently and the best, average (Avg.) and deviation values (Std.) of EDA solutions are recorded and compared with the solutions

Table 2
Simulation settings.

	Notation	Parameter setting in this paper
Problem parameters	n	{50, 100, 300, 500};
	CCR	{0.1, 0.5, 1.0, 5.0};
	m	{2, 4, 8, 16};
	v_i	$1+0.25\times i$
	$B(i, j)$	$1\ (i \neq j);\ 0\ (i = j)$
Algorithm parameters	N	30
	α	0.05
	Stopping criterion	$n^2\log_2(m)$ (ms) and if the best solution keeps unchanged for 10 generations, the algorithm is stopped.

Table 3
Comparative results with model on small scale instance.

$n\|m\|CCR$	Gurobi[a]		EDA[b]				
	C_{max}	LB	CPU(s)	Best	Ave.	Std.	$\overline{CPU(s)}$
14|2|0.10	**23.224**	23.224	187.07	**23.224**	**23.224**	0.000	1.55
14|2|0.50	**19.841**	19.841	746.52	**19.841**	**19.841**	0.000	1.44
14|2|1.00	19.238	11.862	3600.02	**18.615**	18.727	0.225	1.59
14|2|5.00	**37.190**	23.145	3600.10	37.615	37.615	0.000	1.87
14|4|0.10	**16.360**	16.360	1118.46	**16.360**	**16.360**	0.000	3.02
14|4|0.50	**14.553**	10.890	3600.05	14.489	14.489	0.000	2.89
14|4|1.00	**14.190**	8.473	3600.06	**14.190**	**14.190**	0.000	3.50
14|4|5.00	**27.502**	16.532	3600.02	**27.502**	27.522	0.091	4.31
14|8|0.10	10.804	2.533	3600.09	**10.634**	**10.634**	0.000	4.22
14|8|0.50	10.748	2.821	3600.12	**10.214**	**10.214**	0.000	4.25
14|8|1.00	11.718	1.895	3600.07	**9.373**	**9.373**	0.000	4.34
14|8|5.00	35.105	1.895	3600.07	**20.131**	**20.131**	0.000	4.09

[a]2.50 GHz Inter Core i5 CPU.
[b]3.50 GHz Inter Core i5 CPU.

Table 4
Pair-wise comparison to different decoding mechanism/EDA without local search method.

	EDA	EDA-EFT	EDA-HEFT	EDA-OEFT	EDA-OHEFT	EDA-noLS	Naïve GA
EDA	–	12.92	27.71	10.73	35.21	24.90	3.33
		12.71	12.08	5.83	9.58	13.02	0.21
		74.38	60.21	83.44	55.21	62.08	96.46
Naïve GA	96.46	65.00	93.54	73.85	94.69	94.79	–
	0.21	0.52	0.42	0.63	0.31	0.52	
	3.33	34.48	6.04	25.52	5.00	4.69	

and low bounds (LB) of Gurobi. Considering the problem scales, the time limit of solving the model is set as 3600 s. In addition, the CPU time of each algorithm is recorded where the average CPU time ($\overline{\mathbf{CPU(s)}}$) of 20 independent runs for each instance is presented for EDA.

It can be seen from Table 3 that for the small scale instances (m<8), our proposed EDA achieves the Gurobi solutions within a relative short time period under most situations. When $m = 8$, the scale of decision variables doubles and our solutions performs better than the Gurobi solutions produced within 3600s time limit.

5.3. Effectiveness of hybrid decoding mechanism and local intensification

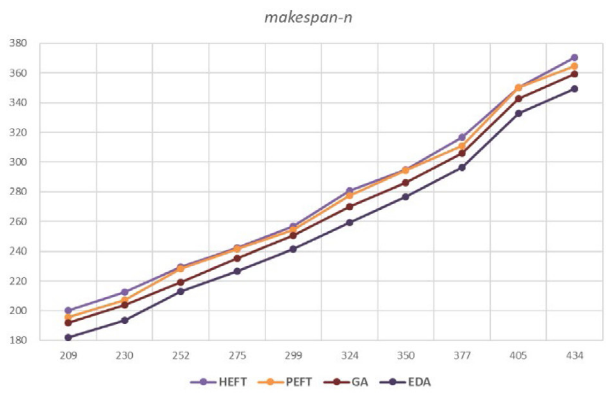
To verify the efficiency of our hybrid decoding mechanism, we use EFT, HEFT, OEFT and OHEFT as the decoding method to compare with our proposed algorithm. In addition, a naïve GA is implemented as a baseline to verify the efficiency of the scheme of EDA, where its population is initialized randomly and crossover and mutation operators are adopted for the evolution of the algorithm. The individuals of GA are encoded with task processing permutations and decoded by HEFT. Pair-wise comparison results are listed in Table 4, where "EDA-X" denotes the EDA embedded with single decoding mechanism "X". In addition, "EDA-noLS" denotes simple EDA with hybrid decoding mechanism without local intensification. 960 independent cases are randomly chosen as the test cases. Table 4 illustrates the percentage of cases where EDA and naïve GA are "worse" than, "equal" to, "better" than the corresponding comparison algorithm.

It can be seen from Table 4 that EDA-OHEFT performs well, which means that it is successful to take PEFT as a decoding method. In addition, compared to single OHEFT, hybrid decoding mechanism performs better on over 65% instances and for other single decoding methods, the superiority of our mechanism is much more obvious. The comparison between EDA-noLS and our proposed EDA shows that the local intensification improves its performance.

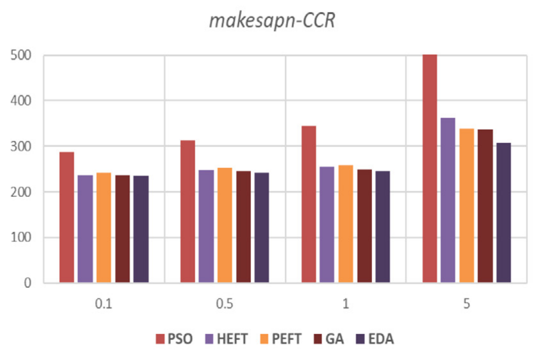
In addition, the comparison results between "EDA-HEFT" and "EDA-noLS" and naïve GA show that, the EDA scheme and the hybrid decoding mechanism are efficient in solving DAG-SP.

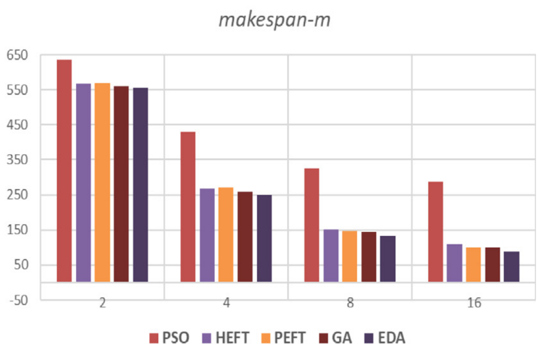
5.4. Comparisons to existing algorithms

Firstly, we compare our algorithm with list scheduling heuristics: HEFT [15] and PEFT [12], and evolutionary algorithms: GA [20] and PSO [25] based on datasets [41]. The average values of are listed according to n, m and CCR in Table 5. Paired T-test is carried out to verify the significant superiority of our algorithm. For each scale groups of comparative results, the alternative hypothesis is set as "the makespan of EDA solution is less than



(a)





(b) (c)

Fig. 15. Makespan plots of Gaussian elimination application. (a). "makespan-*n*" plot; "makespan -*CCR*" histogram (c) "makespan-*m*" histogram.

Table 5
Comparative results sorted by different parameters.

		EDA	HEFT		PEFT		GA (2014)		PSO(2020)	
		$\overline{C_{max}}$	$\overline{C_{max}}$	Sig	$\overline{C_{max}}$	Sig	$\overline{C_{max}}$	Sig	$\overline{C_{max}}$	Sig
	50	**128.68**	164.22	Y	159.58	Y	144.94	Y	245.43	Y
n	100	**219.35**	259.78	Y	249.90	Y	236.84	Y	413.94	Y
	300	**581.75**	630.68	Y	612.75	Y	599.10	Y	1092.20	Y
	500	**957.36**	1012.56	Y	984.45	Y	972.64	Y	1803.44	Y
	2	**917.42**	940.99	Y	923.63	Y	923.63	Y	1156.16	Y
m	4	**486.03**	527.79	Y	510.67	Y	498.03	Y	903.94	Y
	8	**290.35**	341.81	Y	321.46	Y	309.91	Y	776.94	Y
	16	**193.35**	250.51	Y	233.57	Y	221.95	Y	717.97	Y
	0.10	**350.18**	351.89	Y	354.03	Y	350.84	Y	419.99	Y
CCR	0.50	**384.22**	391.44	Y	392.52	Y	386.12	Y	502.15	Y
	1.00	**522.55**	566.06	Y	554.00	Y	536.80	Y	983.75	Y
	5.00	**630.21**	757.84	Y	706.13	Y	679.77	Y	1649.12	Y

Note: The bold values mean the best results.

Table 6
Pair-wise comparison to existing algorithms.

	HEFT	PEFT	GA (2014)	PSO (2020)	Ave.
WORSE	0.23	1.09	12.76	0.96	3.76
EQUAL	1.41	0.96	5.03	0.65	2.01
BETTER	98.36	97.94	82.21	98.39	94.23

the other one under 95% confidence level" and Sig='Y' denotes that the alternative hypothesis is accepted.

From Table 5, it can be seen that the average makespan of EDA performs the best significantly. In addition, it can be seen that when $CCR = 0.1$ and 0.5, difference between EDA and other algorithms, especially GA, is small. The possible reason is that relatively small *CCR* leads to less idle periods in scheduling solutions, and thus there is not much space left for optimization. As *CCR* increases, it can be seen that our superiority of EDA turns larger. The possible reason is that data transfer size increases, poor permutations and decoding method lead to a large amount of idle periods. Our hybrid decoding mechanism and local search method improve the solutions by altering the task processing sequence and choosing appropriate decoding method.

In addition, it can be seen that EDA performs better when the processor number increases. Similar with a larger *CCR*, larger amount of processors increase the difficulty of searching for the best solution. As the encoding sequence of PSO is the processor assignment sequence, when processor number increases, the possibility of processor assignment increases, and it turns to be difficult to search for a good task-processor mapping sequence.

Table 6 illustrates the percentage of cases in which the performance of proposed EDA is "worse" than, "equal" to, "better" than the corresponding comparison algorithm respectively.

From Table 6, it is obvious that our EDA performs better than these comparison algorithms. Compared to the GA, EDA refreshes 82% of its solutions. Similar with GA, knowledge from existing list scheduling heuristics is applied in the evolution and it improves the performance of the proposed algorithm significantly.

Considering the real-world applications: GE and FFT, 10 different scales of DAGs of Gauss Elimination application (b=20-29) are produced. For each scale of DAG, $CCR = \{0.1, 0.5, 1.0, 5.0\}$ and $m = \{2, 4, 8, 16\}$ are used to generate $10\times4\times4$=160 different instances. The average makespan value sorted according to n, m and *CCR* are shown in Fig. 15.

For these 160 instances, EDA overcomes 95.63% solutions produced by GA and 100% solutions from other three algorithms.

Table 7
Comparative results on GE DAGs.

n|m|CCR	RPD^{EDA}	RPD^{GA}		RPD^{PSO}	
	Best/Ave./Std	Best/Ave./Std	Sig.	Best/Ave./Std	Sig.
434|2|0.1	0.00/0.03/0.16	0.05/0.11/0.27	Y	0.51/2.13/7.53	Y
434|2|0.5	0.00/0.03/0.32	0.12/0.26/0.40	Y	2.27/4.03/7.66	Y
434|2|1.0	0.00/0.13/0.72	0.50/0.60/0.41	Y	3.73/6.08/13.01	Y
434|2|5.0	0.00/0.39/1.56	0.98/1.40/1.97	Y	32.61/37.73/23.04	Y
434|4|0.1	0.00/0.12/0.30	0.31/0.39/0.19	Y	28.75/33.64/9.82	Y
434|4|0.5	0.00/0.41/1.18	0.96/1.44/0.74	Y	34.78/39.54/10.54	Y
434|4|1.0	0.00/0.86/0.94	2.33/2.82/1.06	Y	45.11/52.11/10.28	Y
434|4|5.0	0.00/4.32/8.29	6.68/8.85/3.27	Y	124.89137.42/28.94	Y
434|8|0.1	0.00/0.62/0.35	1.24/1.99/0.43	Y	61.06/72.89/7.39	Y
434|8|0.5	0.00/0.91/0.79	2.75/4.16/0.82	Y	77.57/85.93/6.40	Y
434|8|1.0	0.00/1.39/1.05	2.29/5.23/1.43	Y	110.22/114.45/4.81	Y
434|8|5.0	0.00/0.00/0.00	6.91/11.24/3.55	Y	194.25/213.11/35.63	Y
434|16|0.1	0.00/0.08/0.02	0.41/0.50/0.02	Y	100.85/113.80/3.83	Y
434|16|0.5	0.00/1.07/0.46	1.67/3.35/1.00	Y	109.08/118.15/4.83	Y
434|16|1.0	0.00/0.00/0.00	2.20/5.70/1.62	Y	137.52/148.88/6.01	Y
434|16|5.0	0.00/0.00/0.00	2.22/6.21/3.47	Y	251.08/282.85/31.06	Y
Ave.	0.00/0.65/1.01	1.98/3.39/1.54	–	82.14/91.42/13.17	–

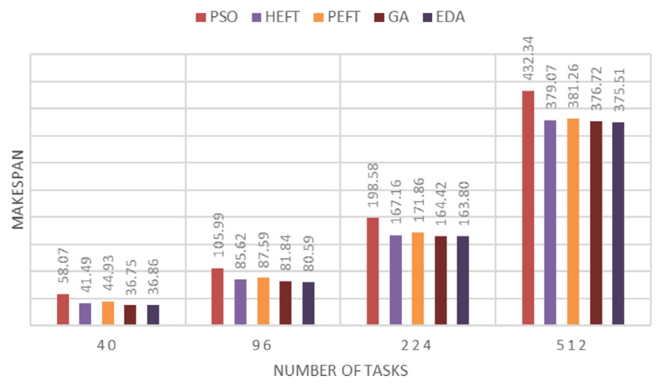


Fig. 16. "makespan -n" histogram of FFT application.

From Fig. 15(a), it can be seen that as the scale of DAG increases, EDA performs the best on average value of makespan.

Specially, for the cases of $n = 434$, 20 independent executions are run for each case and the relative percent deviation (*RPD*) is calculated as (24):

$$RPD^{alg} = \frac{C_{max}{}^{alg} - C_{max}{}^{best}}{C_{max}{}^{best}} \times 100 \tag{24}$$

The best, average *RPD* values and deviation of solutions are recorded as Table 7 and for each case, a two sample T-test is carried out and Sig='Y' denotes that our solutions are better than the others under 95% confidence level. From Table 7, it can be seen that our proposed algorithm perform better than other two algorithms significantly.

For FFT, 4×4×4=64 cases are produced respectively. The average value of makespan sorted by n is presented in Fig. 16 where it can be seen that, except for $n = 40$, EDA performs the best. The results show that it is necessary to increase the capacity of local intensification in EDA to search for optimal solutions when the problem scale is not large.

Specially, for the cases of $n = 512$, 20 independent executions are run for each case and *RPD* is calculated are listed in Table 8. From Table 8, it can be seen that our proposed algorithm performs better when *CCR* and m are relatively larger.

The results of comparison tests on instances from scientific workflows: Cybershake and LIGO, are listed as follows. For each instance, the problem scale, computation and communication data are determined, and different scales of processors (m = {2,4,8,16}) are carried out to test the proposed algorithm. The *RPD* values are listed as Table 9 and it can be seen that our proposed algorithm provides a better and robust performance compared to other algorithms. The file of detailed schedules can be downloaded from https://www.researchgate.net/publication/352697650_DAG-benchmark.

6. Conclusions

In this paper, a path relinking enhanced EDA is designed for DAG-SP. The simulation results based on extensive benchmark datasets with different scales of tasks, processors as well as *CCRs* show that the tailored hybrid decoding mechanism and path relinking based local intensification are effective to improve the performance of EDA. Moreover, our proposed algorithm is superior to existing algorithms in solving DAG-SP. The main contributions of this work in designing the optimization algorithm are summarized as follows.

1. A hybrid decoding mechanism based on existing list heuristics is developed to adjust the produced permutations during the EDA-based search. With such a mechanism, appropriate decoding method can be chosen according to the structure of permutation.
2. A relative position based probability model is designed to describe the probability distribution of solutions. In addition, the probability model is initialized according to the DAG structure and heuristic solutions. Via sampling such a model, diverse and promising task processing permutations can be produced.
3. Two search operators based on problem-specific path relinking scheme are designed, and a path relinking enhanced local intensification procedure is presented. With the local intensification procedure, exploitation ability is enhanced.

7. Managerial implication and future work

Efficient DAG task scheduling algorithm can uplift the performance of IoT, fog or cloud computing applications. Due to the wide application of cloud and fog computing in our daily life, such a solution has significant impact on the optimization of the quality of user experience [46,47]. In addition, appropriate task assignment solution gives rise to the resource balancing within multi-processors, multi-cores and multi virtual machines (VMs) within a single processor. Therefore, DAG task scheduling techniques such as our proposed algorithm can be well applied in the related areas.

For the large and distributed cloud computing environment, it is essential to cut down the extra time generated by scheduling overhead, which results in a shorter execution time of the entire

Table 8
Comparative results on FFT DAGs.

n|*m*|*CCR*	RPD^{EDA}	RPD^{GA}		RPD^{PSO}	
	Best/Ave./Std	Best/Ave./Std	Sig.	Best/Ave./Std	Sig.
512|2|0.10	0.00/0.00/0.00	0.00/0.00/0.02	Y	0.00/0.20/1.58	Y
512|2|0.20	0.00/0.00/0.00	0.01/0.02/0.04	Y	0.25/0.49/1.33	Y
512|2|0.50	0.00/0.00/0.00	0.00/0.00/0.00	Y	0.01/0.39/1.34	Y
512|2|1.00	0.00/0.00/0.00	0.00/0.00/0.00	Y	0.57/2.53/7.31	Y
512|4|0.10	0.00/0.00/0.01	0.01/0.03/0.05	Y	11.63/16.50/12.09	Y
512|4|0.20	0.00/0.03/0.09	0.06/0.12/0.11	Y	12.91/18.86/11.86	Y
512|4|0.50	0.00/0.11/0.33	0.00/0.11/0.41	N	10.61/18.03/11.43	Y
512|4|1.00	0.59/0.73/0.40	0.00/0.46/1.34	N	26.79/32.17/11.76	Y
512|8|0.10	0.00/0.08/0.04	0.00/0.11/0.05	Y	29.34/37.16/4.75	Y
512|8|0.20	0.00/0.14/0.14	0.00/0.40/0.16	Y	36.16/43.18/5.28	Y
512|8|0.50	0.00/0.20/0.16	0.05/0.50/0.24	Y	33.25/43.19/6.87	Y
512|8|1.00	0.00/2.26/1.78	7.89/9.29/1.88	Y	77.39/86.11/8.71	Y
512|16|0.10	0.00/0.19/0.04	0.00/0.27/0.05	Y	62.05/72.78/2.96	Y
512|16|0.20	0.00/0.69/0.18	1.01/1.65/0.18	Y	75.09/82.91/2.22	Y
512|16|0.50	0.00/1.65/0.43	3.75/4.75/0.39	Y	77.23/88.43/2.56	Y
512|16| 1.00	0.00/3.73/3.13	8.61/14.47/2.69	Y	115.72/124.50/5.01	Y
Ave.	0.04/0.61/0.42	1.34/2.01/0.48	–	35.56/41.71/6.07	–

Table 9
Comparative results on scientific workflow instances.

Instance	RPD^{EDA}	RPD^{GA}		RPD^{PSO}	
	Best/Ave./Std	Best/Ave./Std	Sig.	Best/Ave./Std	Sig.
Cybershake_100_2	0.00/0.00/0.00	0.00/0.00/0.01	N	0.02/0.13/1.19	Y
Cybershake_100_4	0.00/0.00/0.00	0.00/0.01/0.01	Y	1.94/10.5/15.34	Y
Cybershake_100_8	0.00/0.03/0.06	0.10/0.36/0.49	Y	23.70/51.75/23.38	Y
Cybershake_100_16	0.00/0.85/0.23	2.56/4.09/0.37	Y	95.98/278.61/124.33	Y
LIGO_100_2	0.00/0.00/0.02	0.00/0.00/0.01	Y	0.02/0.29/21.23	Y
LIGO_100_4	0.00/0.07/2.20	0.07/0.45/10.36	Y	3.39/7.65/82.01	Y
LIGO_100_8	0.00/0.28/3.07	0.44/0.80/2.07	Y	16.14/26.76/74.73	Y
LIGO_100_16	0.00/1.48/3.07	1.17/2.51/3.24	Y	51.68/73.21/47.86	Y

process. Efficient DAG task scheduling algorithm plays an important part in distributed workflow task balancing [5], offering a good solution to the assignment of the tasks within the workflow to the processors.

Similarly, resource consumption cost optimization within many-core processor is important because of the limited resource and high computation load of embedded systems [6]. Efficient DAG task scheduling algorithm is adopted to handle the task precedence constraints during the task scheduling process. In addition, DAG task scheduling algorithm increases the utilization of VMs and brings forth resource balancing in multi-vCPU VMs [7].

In the future, we will study DAG-SP with different metrics and constraints by considering the uncertainty in computing system and applications. In addition, we will develop effective task scheduling algorithms by fusing the evolutionary computing and the problem-specific search knowledge. It is also interesting to apply the proposed algorithm for the real-world applications.

CRediT authorship contribution statement

Chu-ge Wu: Data curation, Analysis, Methodology, Writing draft. **Ling Wang:** Conceptualization, Analysis, Methodology, Review, Funding acquisition. **Jing-jing Wang:** Conceptualization, Methodology, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research is supported by the National Science Fund for Distinguished Young Scholars of China [No. 61525304], the National Natural Science Foundation of China [No. 61873328].

References

[1] E.G. Coffman, P.J. Denning, Operating Systems Theory, Prentice-Hall, Englewood Cliffs, NJ, 1973.
[2] E. Deelman, S. Callaghan, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, et al., Managing large-scale workflow execution from resource provisioning to provenance tracking: the cybershake example, in: 2nd International Conference on E-Science and Grid Computing, e-Science'06, IEEE, Amsterdam, The Netherlands, 2006, pp. 1–14.
[3] TensorFlow Authours, Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016, CoRR, abs/1603.04467.
[4] L. Zhang, L. Zhou, A. Salah, Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments, Inform. Sci. 53 (1) (2020) 31–46.
[5] D. Yu, Y. Ying Y, L. Zhang, et al., Balanced scheduling of distributed workflow tasks based on clustering, Knowl-Based Syst. 199 (2020) 105930.
[6] B. Hu, Z. Cao, Minimizing resource consumption cost of dag applications with reliability requirement on heterogeneous processor systems, IEEE Trans. Ind. Inform. 16 (12) (2020) 7437–7447.
[7] H. Wu, X. Chen, X. Song, Scheduling large-scale scientific workflow on virtual machines with different numbers of vCPUs, J Supercomput. 77 (2021) 679–710.
[8] J.K. Lenstra, A.H.G.R. Kan, Complexity of scheduling under precedence constraints, Oper. Res. 26 (1) (1978) 22–35.
[9] M.A. Al-Mouhamed, Lower bound on the number of processors and time for scheduling precedence graphs with communication costs, IEEE Trans. Softw. Eng. 16 (12) (1990) 1390–1401.
[10] S. Li, Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations, in: 58th Annual Symposium on Foundations of Computer Science, FOCS17, IEEE, Berkeley, CA, 2017, pp. 1–10.
[11] Y. Su, X.Q. Ren, V. Shai, A. Wierman, Y.X. He, Communication-aware scheduling of precedence-constrained tasks, SIGMETRICS Perform. Eval. Rev. 47 (2) (2019) 21–23.
[12] S. Sundar, B. Liang, Offloading dependent tasks with communication delay and deadline constraint, in: IEEE Conference on Computer Communications, INFOCOM 2018, IEEE, Honolulu, USA, 2018, pp. 37–45.
[13] Y.K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surv. 31 (4) (1999) 406–471.
[14] 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.

[15] H. Arabnejad, J.G. Barbosa, List scheduling algorithm for heterogeneous systems by an optimistic cost table, IEEE Trans. Parallel Distrib. Syst. 25 (3) (2014) 682–694.

[16] D. Bozdag, U. Catalyurek, F. Ozguner, A task duplication based bottom-up scheduling algorithm for heterogeneous environments, in: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, IEEE, Rhodes Island, 2006, pp. 12–20.

[17] G.C. Sih, E.A. Lee, A compile-time scheduling heuristic for interconnection-constrained heterogeneous processors architecture, IEEE Trans. Parallel Distrib. Syst. 4 (2) (1993) 175–187.

[18] 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.

[19] R. Sakellariou, H. Zhao, A hybrid heuristic for DAG scheduling on heterogeneous systems, in: 18th International Parallel and Distributed Processing Symposium, IPDPS04, Santa Fe, New Mexico, USA, 2004, pp. 1–11.

[20] E. Ilavarasan, P. Thambidurai, R. Mahilmannan, High performance task scheduling algorithm for heterogeneous computing system, in: 6th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP, Springer Berlin Heidelberg, Melbourne, Australia, 2005, pp. 1–11.

[21] L.F. Bittencourt, R. Sakellariou, E.R.M. Madeira, DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm, in: 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, IEEE, Pisa, 2010, pp. 27–34.

[22] A. Priya, S.K. Sahana, A survey on multiprocessor scheduling using evolutionary technique, in: V. Nath, J. Mandal (Eds.), Nanoelectronics, Circuits and Communication Systems, in: Lecture Notes in Electrical Engineering, Springer Singapore, 2019, pp. 1–10.

[23] 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 (2014) 255–287.

[24] 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.

[25] M.A. Elaziz, S.W. Xiong, K.P.N. Jayasena, L. Li, N. Dong, C. Dai, Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution, Knowl-Based Syst. 169 (2019) 39–52.

[26] 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.

[27] T. Luan, U. de Paula, Y. Frota, D. de Oliveira, L.M.A. Drummond, A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds, Futur. Gener. Comp. Syst. 76 (1) (2017) 1–17.

[28] K. Deng, K. Ren, S. Liu, J. Song, DAG scheduling for heterogeneous systems using biogeography-based optimization, in: 21st International Conference on Parallel and Distributed Systems, ICPADS, Melbourne, VIC, Australia, 2015, pp. 708–716.

[29] T. Biswas, P. Kuila, A.K. Ray, A novel workflow scheduling with hybrid criteria using particle swarm optimization for heterogeneous computing systems, Cluster Comput. 1 (1) (2020) 1–17.

[30] A. Paliwal, F. Gimeno, V. Nair, Y. Li, M. Lubin, P. Kohli, O. Vinyals, Reinforced genetic algorithm learning for optimizing computation graphs, in: International Conference on Learning Representations, 2020, pp. 1–10.

[31] P. Larranaga, J.A. Lozano, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Springer, US, 2002.

[32] M. Pelikan, M.W. Hauschild, F.G. Lobo, Estimation of distribution algorithms, in: J. Kacprzyk, W. Pedrycz (Eds.), Handbook of Computational Intelligence, Springer, Berlin Heidelberg, 2015.

[33] Y. Chen, Y.C. Jin, X.Y. Sun, An improvement decomposition-based multi-objective evolutionary algorithm using multi-search strategy, Knowl-Based Syst. 200 (1) (2020) 1.

[34] J. Ceberio, E. Irurozki, A. Mendiburu, et al., A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems, Prog. Artif. Intell. 1 (1) (2012) 103–117.

[35] C.G. Wu, W. Li, L. Wang, A.Y. Zomaya, Hybrid evolutionary scheduling for energy-efficient fog-enhanced internet of things, IEEE Trans. Cloud Comput. 1 (1) (2018) 1–8.

[36] Z.C. Li, B. Qian, R. Hu, L.L. Chang, J.B. Yang, An elitist nondominated sorting hybrid algorithm for multi-objective flexible job-shop scheduling problem with sequence-dependent setups, Knowl-Based Syst. 173 (2019) 83–112.

[37] F. Glover, M. Laguna, R. Marti, et al., Fundamentals of scatter search and path relinking, Control Cybern. 29 (3) (2000) 653–684.

[38] M.G. Resende, C.C. Ribeiro, F. Glover, R. Martí, Scatter search and path-relinking: fundamentals, advances, and applications, in: M. Gendreau, J.Y. Potvin (Eds.), Handbook of Metaheuristics, in: International Series in Operations Research & Management Science, Springer, Boston, MA, 2010.

[39] S. Baluja, Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning, Carnegie-Mellon Univ Pittsburgh, Dept of Computer Science, 1994, pp. 1–20.

[40] S. Ho, M. Gendreau, Path relinking for the vehicle routing problem, J. Heuristics 12 (1) (2006) 55–72.

[41] T. Tobita, H. Kasahara, A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, J. Sched. 5 (5) (2002) 379–394.

[42] M. Cosnard, M. Marrakchi, Y. Robert, D. Trystram, Parallel Gaussian elimination on an MIMD computer, Parallel Comput. 6 (3) (1988) 275–296.

[43] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction To Algorithms, MIT Press, 1990.

[44] G. Juve, A. Chervenak, E. Deelman, et al., Characterizing and profiling scientific workflows, Futur. Gener. Comp. Syst. 29 (3) (2013) 682–692.

[45] L.C. Canon, E. Jeannot, R. Sakellariou, W. Zheng, Comparative Evaluation of the Robustness of Dag Scheduling Heuristics, Grid Computing: Achievements and Prospects, Springer, Boston, MA, 2008, pp. 73–84.

[46] A. A. Brogi, S. Forti, QoS-aware deployment of IoT applications through the fog, IEEE Internet Things J. 4 (5) (2017) 1185–1192.

[47] S.E. Mahmoodi, R.N. Uma, K.P. Subbalakshmi, Optimal joint scheduling and cloud offloading for mobile applications, IEEE Trans. Cloud Comput. 7 (2) (2019) 301–313.