Task Scheduling with the Adaptive Branch and Bound Algorithm

Jue-Jun Wu[a] and Chang-Biau Yang[a*]and Kuo-Tsung Tseng[b]
[a]Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
[b]Department of Shipping and Transportation Management
National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan

Abstract

Efficient task scheduling is crucial in heterogeneous distributed computing systems. In previous researches, some list-based scheduling algorithms have been proposed to solve the problem. Those algorithms perform the task-processor assignment according to a specific task order. Such algorithms might fall into the local optima. In this paper, we apply an adaptive branch and bound algorithm to search for more task-processor feasible assignments. An estimation function for evaluating the future makespan is proposed to decide the goodness of each feasible branch. The estimation function is adjustable by changing the α value, which determines the branching degree. To evaluate the performance of our algorithm, we use the improvement percentage metric with the heterogeneous earliest finish time (HEFT) algorithm's makespan as the baseline. Within an acceptable execution time, our algorithm achieves approximately 10% improvement when CCR, graph level, number of tasks, and number of processors are large.

Keywords: task scheduling, task-processor assignment, heterogeneous network graph, list-based scheduling, branch and bound

1 Introduction

The efficiency of parallel processing on a heterogeneous network depends deeply on the arrangement of tasks and processors. The task scheduling on a heterogeneous network is represented by a *directed acyclic graph* (DAG). Each task is represented by a node, and each arc (directed edge), associated with a communication cost, represents the execution order of the two tasks. The task scheduling problem [4, 7, 8] aims to assign available processors to the tasks and to determine the start and end time of each task for minimizing the *makespan*, the total elapsed time. There are many variants for the task scheduling problem so that it might focus on different optimization goals. With the rise of cloud computing, energy consumption, and currency costs may become crucial optimization criteria.

The *list-based task scheduling* [1–3, 5, 6, 9] has now become more popular because there are many deformation methods to rank and allocate tasks. There are two stages in the list-based task scheduling, including task ranking and task allocation. The task ranking stage calculates the priority order of each task. In the task allocation stage, we assign a processor to each task according to the priority order. These algorithms usually have the advantage of low computational complexity, but their scheduling results might fall into a locally optimal state.

In this paper, we solve the static task scheduling problem in the heterogeneous environment by using the adaptive branch and bound approach. In our approach, there is no predetermined task order. Both task order and processor assignment are decided in the branching stage. Due to the possible long execution time, we attempt to avoid some branches by using an estimation function. Thus, the estimation function of the future makespan plays an important role. The parameters of the estimation function in our algorithm can be adjusted to fit different criteria according to different situations.

The experimental results show that the makespan obtained by our algorithm is better than those obtained by the previous heuristic algorithms. It also reveals the advantages of our al-

*Corresponding author. E-mail: cbyang@cse.nsysu.edu.tw (Chang-Biau Yang).

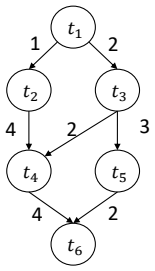

Figure 1: An example of a simple DAG.

gorithm according to various CCRs with different numbers of tasks and processors, where CCR is an approximate ratio of the communication time to the computation time for one task. When CCR is large, our algorithm improves makespan by about 10%, compared to HEFT [9].

The rest of this paper is organized as follows. Section 2 introduces the previous works in task scheduling methods. Section 3 presents our estimation function of the future makespan. Section 4 presents the experimental results for evaluating the proposed algorithm. Finally, Section 5 summarizes and concludes this paper.

2 Preliminaries

Definition 1. (makespan and task scheduling) *In the task scheduling problem on a heterogeneous network, we are given a* directed acyclic graph *(DAG)* $G = (V, E)$ *and a computation cost (time) table. Each task* t_i *is represented by a node* v_i *in* V*, and each arc* $(t_i, t_j) \in E$ *represents that* t_i *should be executed before* t_j*. In addition, each arc* (t_i, t_j) *is associated with a communication cost (time), which is required if the two tasks are assigned different processors. In the computation cost table, each entry represents the time required by processor* p_i *assigned to task* t_j*. The* makespan *is the elapsed time from the start of the first task to the end of the last task. The* task scheduling *problem aims to minimize the the makespan by assigning the available processors to the tasks.*

Figure 1 shows the DAG graph of six tasks. Table 1 shows the computation cost of each task on two processors.

Based on the list-based scheduling approach, Wu *et al.* proposed the *heuristic branch-and-bound with hill climbing* (BnB_HC) [10] to assign tasks to processors in the task allocation stage. In the first stage, it adopts the HEFT [9] ranking methods to get the order of each task. In the second stage, it invokes the hill climbing in the branch-and-bound method to do the task-processor assignment.

Table 1: The computation costs of two available processors for the DAG in Figure 1.

task \ processor	p_1	p_2
t_1	3	2
t_2	2	5
t_3	4	3
t_4	2	2
t_5	2	4
t_6	3	5

The hill climbing method is a variant of the depth first search (DFS). The method maintains a current minimum makespan. During searching the path, if the earliest finish time of task t_i is larger than the current minimum makespan, then the route need not be traversed. That is, it stops searching the path earlier in this situation. BnB_HC designs an estimation function to predict the required time for the undone tasks in the path.

The estimation function is calculated recursively by Equation 1. $E(H_{1..k-1},\ t_i^k, p_j, \alpha)$ represents an estimated value, where $H_{1..k-1}$ denotes a list of tasks and processors which have been already assigned from level 1 to level $k-1$; t_i^k and p_j are the chosen task and processor on level k, respectively. $AFT(\ )$ represents the available finish time of the task, and $A(\ )$ is the evaluated cost of the undone tasks in the future. The value of α decides the pruning degree.

$$E(H_{1..k-1}, t_i^k, p_j, \alpha) = \max_{1 \le j \le m} \{ AFT(H_{1..k-1}, t_i^k, p_j^k) + A(H_{1..k+1}, t_i^k, \alpha) \}. \quad (1)$$

The algorithm might miss the optimal solution, but it can shorten the time to get a better makespan than previous heuristic algorithms. However, unchanged task order may not be good enough for searching the optimal solutions.

3 Our Algorithm

We apply the adaptive branch-and-bound approach to task-processor assignment for all routes in the searching tree. Every level considers all unassigned tasks (not a specific task of a specific order) on all available processors. In this situation, the variety of paths cannot be predicted in advance. We present a simple example to present how we search all feasible paths.

We design an estimation function, expanded from the previous research [10]. The estimation function is used to evaluate the future makespan and then to decide whether we will branch the 2-tuple of task-processor assignment or not. We do not use a fixed order to choose the task. Instead, we try all possible routes, but it takes too much time to visit all routes. To reduce the solution searching space, our estimation function, shown in Equation 2, overestimates the future cost before starting a potentially feasible extension. If the estimated value is smaller than the current minimum makespan, we branch the path. Otherwise, we discard the path.

$$E(H_{1..k-1}, t_i^k, p_j, \alpha) = \max_{1\le j\le m}\{M(H_{1..k-1}, t_i^k, p_j) + \frac{1}{m}\times \sum_{i\notin(H_{1..k.task})}(\frac{w_{i,j}}{2-\alpha}+ \frac{\alpha\times(w^{\mathrm{o}}\times\delta_i^{\mathrm{MO}}+w^{\mathrm{i}}\times\delta_i^{\mathrm{MI}}}{m-\alpha})\}. \quad (2)$$

$M(\ )$ computes the makespan when processor j is assigned to task i on level k, and it has the known cost $H_{1..k-1}$ of the existing assignment from levels 1 through $k-1$. The remaining part evaluates the cost of the undone tasks, considering four parameters: (1) the weight of the maximal outgoing communication cost of the undone tasks, denoted by w^{o}; (2) the weight of average incoming communication costs of the undone task, denoted by w^{i}; (3) the adjustable α value at a specific time; (4) the variance of the incoming communication cost of the undone task t_i, denoted by $\sigma_{t_i}^2$.

Consider the first and the second parameters together. δ_j^{MO} is calculated by Equation 3, which means that we might spend the worst cost if we are going to choose task t_j at the next level. δ_j^{MI} is calculated by Equation 4, where d_j^{in} denotes the indegree of t_j. If task t_j is a critical node and it has a small δ_j^{MI}, then we should pick it earlier. They both consider the communication costs of the same task, and we do not want it to be double counted. Thus, we give them a weight and set $w^{\mathrm{o}}+w^{\mathrm{i}}=1$. As the experimental results show, we can get the better makespan for the setting $w^{\mathrm{o}}=0.8$ and $w^{\mathrm{i}}=0.2$.

$$\delta_j^{\mathrm{MO}} = \max_{t_i\in succ(t_j)}\{c_{j,i}\} \quad (3)$$

$$\delta_j^{\mathrm{MI}} = \frac{\sum_{t_i\in pred(t_j)} c_{i,j}}{d_j^{\mathrm{in}}} \quad (4)$$

If we set $\alpha \ge 1$, the estimation function $E(\ )$ tends to overestimate the cost frequently. Thus, most branches are discarded, though the search runs very fast. In this situation, we lose many better solutions. In contrast, if $\alpha = 0$, $E(\ )$ underestimates the cost, and it visits most of the feasible paths. So the search cannot end in an acceptable time. Thus, we narrow the range of α to 0.2 through 0.8.

Note that in the extreme case, if $\alpha = -\infty$, the right part of Equation 2 becomes zero, and then the existing makespan $M(\ )$ becomes the lower bound of the solution. When $\alpha = -\infty$, our algorithm becomes the traditional branch and bound. Thus, the optimal solution can be obtained.

In BnB_HC algorithm [10], the α value is fixed in every single experiment. Our experimental results show that changing the α value more dynamically can find the better makespan. We think the key point is the communication cost. If we assign the predecessors and successors of task t_i on the same processor, then the communication cost is 0, and we may decrease the makespan. We find out that the variance of incoming communication cost, $\sigma_{t_i}^2$, can accelerate searching paths and dynamically adjust the α value.

According to some primitive experimental results, we set $\alpha \in \{0.2, 0.3, 0.5, 0.8\}$ in Equation 5. $\sigma_{t_i}^2$ denotes the variance of incoming communication costs of t_i, and σ_{avg}^2 denotes the variance of average incoming communication costs of all tasks (excluding the entry task).

$$\alpha = \begin{cases} 0.2 & \text{if } 1.5\sigma_{\mathrm{avg}}^2 < \sigma_{t_i}^2 \\ 0.3 & \text{if } \sigma_{\mathrm{avg}}^2 < \sigma_{t_i}^2 \le 1.5\sigma_{\mathrm{avg}}^2 \\ 0.5 & \text{if } 0.5\sigma_{\mathrm{avg}}^2 < \sigma_{t_i}^2 \le \sigma_{\mathrm{avg}}^2 \\ 0.8 & \text{if } \sigma_{t_i}^2 \le 0.5\sigma_{\mathrm{avg}}^2. \end{cases} \quad (5)$$

The pseudo code of our algorithm is presented in Algorithm 1. The implementation of the algorithm is done recursively.

With the example in Figure 1, we illustrate our algorithm in Figure 2. In every node, we show the task-processor assignment 2-tuple $\langle t_i, p_j\rangle$, meaning that processor p_j is assigned to task t_i, and the estimated value is given below. We demonstrate how the estimation function $E(\ )$, defined in Equation 2, works as follows.

σ_{avg}^2 is calculated as 0.64. The $\sigma_{t_i}^2$ values of tasks $\{t_1, t_2, t_3, t_4, t_5, t_6\}$ are $\{0, 0.25, 0.25, 0, 0, 0\}$, respectively. By the Equation 5, the α values of the tasks are obtained as 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, respectively. We set $w^{\mathrm{o}}=0.8$ and $w^{\mathrm{i}}=0.2$.
$E(\langle\ \rangle, 1, 1, 0.8) = M(\langle\ \rangle, 1, 1) + \frac{1}{2}\times(\frac{2+4+2+2+3}{2-0.8}+$

Algorithm 1 Adaptive Branch and Bound Algorithm

Input: A task scheduling network G with n tasks and m processors, the DAG of task dependency, the task computation cost, and the adjustment weight α.
Output: The near minimal makespan CM.

```
1: Initialize H[1..n], k ← 0, CM ← ∞
2: ABnB(H[1..k], k, D^k[1..n]){
3: Use H[1..k] to update D^k[1..n], w ← 0.
4: for each (undone task t_i with D^k[t_i] = 0) do
5:     for j = 1 to m do
6:         B[w].task ← t_i
7:         B[w].proc ← j
8:         B[w].E ← E(H[1..k] , t_i^k, p_j, α)
9:         w = w + 1
10: Sort B[·].E increasingly
11: if k + 1 = n then
12:     if B[1].E < CM then
13:         CM ← B[1].E
14:         H[k + 1] ← B[1]
15:     return CM
16: for i = 1 to w do
17:     if B[i].E < CM then
18:         H[k + 1] ← B[i]
19:         D_{k+1}[n] ← H[1..k + 1]
20:         ABnB(H[1..k + 1], k + 1, D_{k+1}[1..n])
21:     else
22:         return CM
23: }
```

$\frac{0.8\times(0.8\times(4+3+4+2+0)+0.2\times(1+2+3+3+3))}{2-0.8}$)
$= 3 + 9.68 = 12.68.$
$E(\langle\ \rangle, 1, 2, 0.8) = M(\langle\ \rangle, 1, 2) + \frac{1}{2} \times (\frac{5+3+2+4+5}{2-0.8} +$
$\frac{0.8\times(0.8\times(4+3+4+2+0)+0.2\times(1+2+3+3+3))}{2-0.8}$)
$= 2 + 12.18 = 14.18.$

The value of $E(\langle\ \rangle, 1, 1, 0.8)$ is the smallest, so we start branching from $\langle t_1, p_1\rangle$. The rest of the searching paths are shown in Figure 2. In node d, $E(\langle\langle 1,2\rangle\langle 2,1\rangle\langle 3,2\rangle\langle 4,1\rangle\langle 5,1\rangle\rangle, 6, 1, 0.8) = M(\langle\langle 1,2\rangle\langle 2,1\rangle\langle 3,2\rangle\langle 4,1\rangle\langle 5,1\rangle\rangle, 6, 1) + \frac{1}{2} \times (\frac{0}{2-0.8} +$ $\frac{0.8\times(0)+0.2\times(0))}{2-0.8}) = 14.$

Since node d is a leaf node, the makespan of the path (assignment) is 14.

4 Experimental Results

The experimental datasets are generated randomly with various numbers of tasks, numbers of processors, and CCR values. Each generated DAG is a multi-stage graph, whose parameters are set as follows: number of tasks $n \in \{10, 20, 40, 60, 80, 100\}$, number of processors $m \in \{2, 3, 5, 7, 10\}$, ratio of communication time to computation time $CCR \in \{0.1, 0.5, 1, 5, 10\}$. There are $6 \times 5 \times 5 = 150$ experimental datasets. Every dataset considers various shape β and ratio of computation time γ, which are set as $\beta \in \{0.5, 1, 2.0\}$ and $\gamma \in \{0.125, 0.25\}$. We generate 10 DAGs for every type.

Table 2: The makespan improvement to HEFT with number of processors $m \in \{7, 10\}$, number of tasks $n \in \{60, 80, 100\}$, $\beta \in \{0.5, 1, 2\}$ and $CCR = 10$, where the cumulative improvement percentage is evaluated at that second.

(a) $m = 7$

n	β	Algo	1s	5s	10s	50s	100s	300s	600s
60	0.5	r_o	0.0865	0.0875	0.088	0.089	0.09	0.0915	**0.092**
		r_w	-0.0355	-0.0185	-0.0155	-0.0065	-0.005	0.003	0.0095
	1	r_o	0.058	0.061	0.063	0.064	0.0645	0.0675	0.0675
		r_w	-0.044	-0.037	-0.036	-0.032	-0.031	-0.022	-0.02
	2	r_o	0.0445	0.0485	0.0495	0.055	0.0555	0.0565	0.0868
		r_w	-0.019	-0.0175	-0.016	-0.0135	-0.0075	-0.006	0.004
80	0.5	r_o	0.0485	0.055	0.055	0.056	0.0565	0.0575	**0.0585**
		r_w	-0.037	-0.029	-0.0225	-0.0145	-0.011	-0.0025	0.0005
	1	r_o	0.0495	0.0525	0.053	0.0545	0.055	0.0555	0.0555
		r_w	-0.0445	-0.037	-0.0315	-0.0275	-0.027	-0.0235	-0.023
	2	r_o	0.019	0.0472727	0.025	0.028	0.029	0.029	0.029
		r_w	-0.0145	-0.0135	-0.0135	-0.0135	-0.0135	-0.0125	-0.0125
100	0.5	r_o	0.07	0.0735	0.0755	0.0755	0.0775	0.0775	**0.0775**
		r_w	-0.0445	-0.0365	-0.035	-0.0305	-0.0285	-0.0265	-0.0235
	1	r_o	0.0445	0.0485	0.05	0.0525	0.053	0.053	0.055
		r_w	-0.0405	-0.0395	-0.038	-0.033	-0.0315	-0.029	-0.028
	2	r_o	0.024	0.0704545	0.0255	0.029	0.029	0.0295	0.0295
		r_w	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215	0.0215

(b) $m = 10$

n	β	Algo	1s	5s	10s	50s	100s	300s	600s
60	0.5	r_o	0.057	0.0625	0.0625	0.0665	0.067	0.067	**0.0675**
		r_w	-0.0195	-0.0065	-0.002	0.012	0.0135	0.0235	0.028
	1	r_o	0.033	0.0395	0.041	0.046	0.046	0.048	0.0485
		r_w	-0.051	-0.048	-0.0455	-0.0325	-0.03	-0.02	-0.0165
	2	r_o	0.03	0.0365	0.0375	0.044	0.045	0.049	0.0609
		r_w	-0.019	-0.0175	-0.0175	-0.0145	-0.0145	-0.0105	-0.0095
80	0.5	r_o	0.0865	0.087	0.0875	0.09	0.0905	0.0905	**0.091**
		r_w	-0.019	-0.015	-0.007	0.0025	0.01	0.0175	0.0205
	1	r_o	0.0345	0.037	0.0405	0.0425	0.043	0.045	0.046
		r_w	-0.062	-0.056	-0.0525	-0.0425	-0.0425	-0.0395	-0.038
	2	r_o	0.001	0.003	0.0065	0.009	0.0105	0.0115	0.0868
		r_w	-0.008	-0.008	-0.008	-0.008	-0.008	-0.008	-0.008
100	0.5	r_o	0.063	0.0635	0.0645	0.065	0.065	0.065	**0.065**
		r_w	-0.065	-0.0625	-0.061	-0.0545	-0.052	-0.049	-0.0455
	1	r_o	0.031	0.0335	0.034	0.0355	0.036	0.037	0.0385
		r_w	-0.0485	-0.0475	-0.0475	-0.0465	-0.0425	-0.0345	-0.0345
	2	r_o	-0.005	-0.0015	-0.0015	0.0035	0.0035	0.005	0.0609
		r_w	-0.0095	-0.0095	-0.0095	-0.0095	-0.0095	-0.0095	-0.0095

We compare the execution time of our algorithm and the makespan produced by our algorithm to HEFT [9] and BnB_HC [10] (hereinafter referred to as Wu). For simplicity, the makespans of HEFT, Wu's, and our algorithms are represented as M_h, M_w, and M_o, respectively. In our algorithm, we attempt to find a better solution, so we have to expand more paths to find the better solution. We set that every single experiment records the makespan at every second and stops when 600 seconds elapse.

In Table 2, every row shows the average of cumulative improvement percentage for the same shape ratio, where β is divided into $\{0.5, 1, 2.0\}$, separately. We figure out that β is the obvious feature. As the experimental results show, our algorithm performs well when $\beta = 0.5$, representing narrow graphs. Our algorithm is better than Wu's because we can adjust the α value dynamically.

In the following figures, we use (CCR, n, m) to

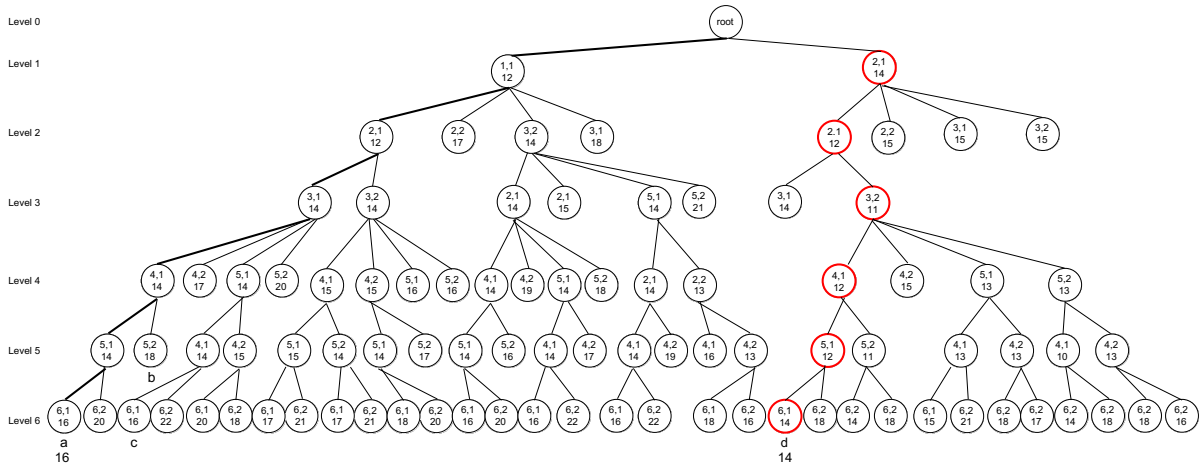

Figure 2: A demonstration of our algorithm with Figure 1.

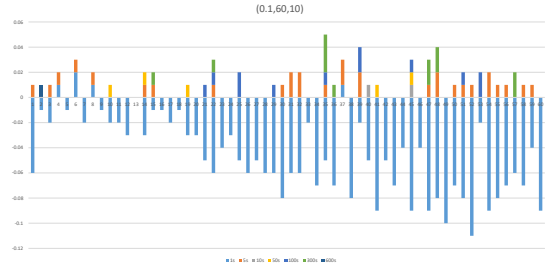

Figure 3: The cumulative improvement percentage of our makespan to HEFT with $CCR = 0.1$, $n = 60$ and $m = 10$.

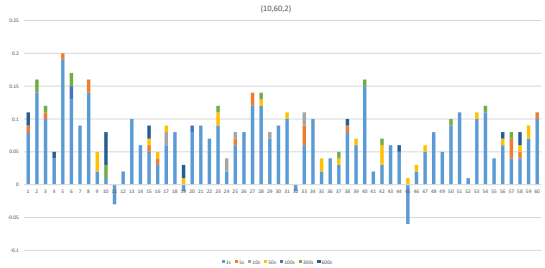

Figure 4: The cumulative improvement percentage of our makespan to HEFT with $CCR = 10$, $n = 60$ and $m = 2$.

identify various parameters for illustrating the experimental results. The datasets from 1 to 20, 21 to 40, and 41 to 60 have different values of $\beta \in \{0.5, 1, 2\}$, respectively. The cumulative improvement percentage of our makespan to HEFT makespan is shown with a different color at the $\{1, 5, 10, 50, 100, 300, 600\}$ second, respectively.

Our algorithm does not perform well in Figure 3. When our computation cost is fixed between 20 to 50, the communication cost is set between 1 through 20 if the CCR is small. The small value of communication cost is too close to distinguish the α value because our algorithm adjusts the α value based on the variance of communication costs.

In Figures 4, and 5, our algorithm is better than HEFT. In these datasets, we can get a better solution within one second. Due to the variety of communication costs, the value is randomly set from 100 through 200. The $\sigma_{t_i}^2$ value can specifically decide a proper α value.

For example, with 60 tasks, the tasks can be divided into 12 or 6 levels in the DAG. With 12

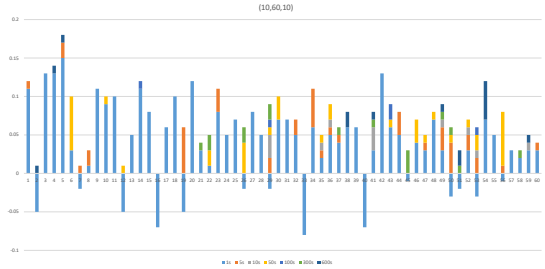

Figure 5: The cumulative improvement percentage of our makespan to HEFT with $CCR = 10$, $n = 60$ and $m = 10$.

levels, changing the α value can visit the feasible paths more precisely. With 6 levels, the tasks on the same level share the same α value, and the difference between tasks is not apparent. In this situation, our algorithm makes only a slight improvement.

When the α value is small, the gap in the communication cost at this layer is large. Thus, we want to see more details to branch out more feasible tasks at this level. In contrast, when the α value is large, the gap of the communication cost at this layer is small. No matter which feasible path we choose, there is only a little difference. In this situation, we can discard the rest of the feasible paths at this level to speed up the search.

5 Conclusion

We conducted experiments by testing various combinations of parameters, including CCRs, the number n of tasks, the number m of processors, the shape of the graph, and the α value. According to the experimental results, our algorithm performs better when $CCR \in \{5, 10\}$, $n \in \{60, 80, 100\}$, $m \in \{7, 10\}$ and the levels are the more, the better. In these datasets, our algorithm achieves a cumulative improvement of about 10% in the average makespan compared to HEFT. It is because the communication cost between the tasks plays a pivotal role in our algorithm and determines the precise weight of the parameter.

Every DAG has its unique feature, such as incoming edge, outgoing edge, shape, and the number of levels. In the future, we may adjust the search paths more precisely to find a better solution. By taking these DAG features into account, we can set more appropriate parameters to achieve our goal and find answers more efficiently.

References

[1] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 25, No. 3, pp. 682–694, 2014.

[2] L. F. Bittencourt, R. Sakellariou, and E. R. Madeira, "DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," *The 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pp. 27–34, Pisa, Italy, 2010.

[3] Y. Dai and X. Zhang, "A synthesized heuristic task scheduling algorithm," *The Scientific World Journal*, Vol. 2014, pp. 1–9, 2014.

[4] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, Vol. 13, No. 3, pp. 219–237, 2005.

[5] E. Ilavarasan and P. Thambidurai, "Low complexity performance effective task scheduling algorithm for heterogeneous computing environments," *Journal of Computer Sciences*, Vol. 3, No. 2, pp. 94–103, 2007.

[6] M. A. Khan, "Scheduling for heterogeneous systems using constrained critical paths," *Parallel Computing*, Vol. 38, No. 4-5, pp. 175–193, 2012.

[7] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, and D. Okaya, "SCEC cybershake workflow-automating probabilistic seismic hazard analysis calculations," *Workflows for e-Science*, pp. 143–163, Springer, 2007.

[8] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, and M. Samidi, "Scheduling data-intensiveworkflows onto storage-constrained distributed resources," *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, pp. 401–409, Rio de Janeiro, Brazil, 2007.

[9] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 3, pp. 260–274, 2002.

[10] C.-K. Wu, C.-B. Yang, and K.-T. Tseng, "The task scheduling with adaptive branch and bound," *Proceedings of the National Computer Symposium 2021 (NCS 2021)*, pp. 478–483, Taichung, Taiwan, 2021.