Efficient Genetic Algorithms for Task Scheduling with the Deadline Constraint

Chih-Hsuan Chien
Dept. Computer Science & Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan

Chang-Biau Yang
Dept. Computer Science & Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
cbyang@cse.nsysu.edu.tw
(Corresponding author)

Kuo-Tsung Tseng
Dept. of Shipping and Transportation Management
National Kaohsiung Univ. of Science and Technology
Kaohsiung, Taiwan

Abstract

Most research of task scheduling focuses on minimizing the makespan or the cost to fulfill user requirements. Some studies utilized genetic algorithms (GA) to solve the task scheduling problem. In these studies, the information of scheduling order, task-service assignment, and task-processor assignment is encoded together in the chromosome. However, such an encoding scheme may decrease the efficiency of the algorithm due to the large solution space it generates. This paper integrates a heuristic algorithm into the GA framework. In the first phase, only scheduling order and task-service assignment are encoded into the chromosome. And in the second phase, the task-processor assignment is determined by a heuristic algorithm, then served as the fitness score of the GA. Our two-phase approach significantly reduces the solution space. As experimental results show, the solutions obtained by our two-phase algorithm are superior to those of the previous studies.

Keywords: genetic algorithms, task scheduling, task ordering, task-processor assignment, heterogeneous network graph

I. Introduction

With the development of networks and computers, the field of cloud computing [1] has also emerged as the times require. The cloud computing provides high-performance computing resources to the users for the execution of huge and complex missions. There are several advantages of cloud computing, such as to provide unlimited resources on users' demand, or to pay only for the time period used [2].

In general, the task scheduling problem has many different versions, most of which are NP-hard [3]. The problem needs to assign appropriate resources to each task in the workflow to minimize or meet quality of service (QoS) constraints, such as deadlines and budgets. Among the various QoS constraints, makespan and cost are two noteworthy evaluation criteria.

Several algorithms have been developed for the task scheduling problem with deadline constraint. They are classified into heuristic and meta-heuristic algorithms. *IaaS cloud partial critical path* (ICPCP) algorithm [4] and *just-in-time for cloud* (JITC) [5] are two examples of heuristics algorithms. Modern meta-heuristic algorithms try to find a near-optimal answer in an acceptable computation time, such as *cost-effective firefly algorithm* (CEFA) [6], *particle swarm optimization* (PSO) [7], and *cost effective genetic algorithm* (CEGA) [8].

In this paper, we focus on solving the problem with the objective of minimizing the cost of the scheduling that fulfills the deadline constraint. Our proposed algorithm adopts a two-phase scheme, integrated into the genetic algorithm framework. We refer to our algorithm as the *two-phase genetic algorithm* (2pGA). In the first phase of 2pGA, we encode solely the scheduling order and the task-service assignment into the chromosome, excluding task-processor assignment. After a chromosome is generated in the first phase, we design a heuristic algorithm to determine the task-processor assignment in the second phase.

To evaluate the performance of our 2pGA, we employ the one-phase genetic algorithm (1pGA), known as CEGA [8], as the baseline. The experimental results demonstrate that the solutions obtained by 2pGA outperform those obtained by 1pGA when CCR is large, where CCR represents an approximate ratio of the communication time to the computation time for an individual task. The average improvement is

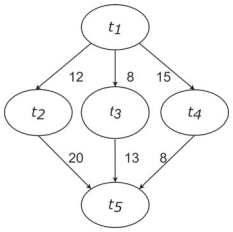


Fig. 1: An example of a workflow represented by a DAG.

approximately 17%. We also show that our heuristic algorithm does well for task-processor assignment. Alternatively, we can employ a genetic algorithm for the second phase to determine task-processor assignment. Experimental results show that GA in the second phase improves only about 3%, but the required execution time significantly increases.

The rest of this paper is organized as follows. In Section II, the related works for solving the task scheduling problem are presented. In Sections III and IV, we present algorithm in detail. In Section V, our proposed algorithms are evaluated experimentally. Finally, Section VI concludes the paper.

II. Preliminaries

Numerous studies have been proposed on the task scheduling problem, which is considered as a NP-hard problem in general. A *directed acyclic graph* (DAG) is usually applied to describe the structure of a workflow. A task which is executed on different service/processor may have different processing cost/time. Figure 1 shows a workflow with five tasks.

Definition 1 (task scheduling problem [4])**.** *In the task scheduling problem, a workflow G, a task-service computation time table S, a service cost table C, a billing unit time constant β, and a deadline δ along with a penalty function $pen(\ )$ are given. S is of size $n \times m$, where n and m denote the number of tasks and services, respectively. $S_{i,j}$ in S represents the computation time required for task t_i executed on one processor of service j. The continuously used time for a processor is rounded up to β. The* task scheduling *problem aims to assign the tasks to the processors in the service such that the cost is minimized. The number of available processors in one service is assumed to be unlimited. In addition, $pen(\ )$, a penalty function, is added to the cost if the makespan of the resulted schedule exceeds δ.*

The optimization formula is shown in Equation 1

TABLE I: The computation time and cost in different services for the DAG of Figure 1. (a) The computation time. (b) The costs of the services.

	s_1	s_2	s_3
t_1	11	12	15
t_2	13	14	18
t_3	9	10	12
t_4	14	18	21
t_5	8	11	14
deadline=70			

(a)

	s_1	s_2	s_3
cost	5	3	2
billing unit $\beta = 10$			

(b)

	t_1	t_2	t_3	t_4
scheduling order	1	3	2	4
assigned service	1	1	2	2
assigned processor	1	2	1	1

Fig. 2: An example of encoding in CEGA [8].

[4] as follows.

$$\begin{aligned}&\text{Minimize} \quad Cost + pen(makespan).\\ &Cost = \Sigma_{k=1}^{|resourcepool|} C_{type}(p_{s,k}) \times \psi.\\ &pen(makespan) = \begin{cases} \frac{makespan-\delta}{\beta} \times 5 \times C_\delta & \text{if } makespan > \delta;\\ 0 & \text{otherwise.}\end{cases}\end{aligned} \tag{1}$$

In the above equation, $C_{type}(p_{s,k})$ represents the unit cost of the processor $p_{s,k}$ in the service type, and ψ denotes the total number of billing units. However, there may be some solutions exceeding the deadline, so we use a penalty function $pen(\ )$ to add the cost. In such a way, we transform the constrained problem with deadline into an unconstrained one without deadline, but with extra penalty.

Table I shows the computation time and the cost of each service, where β is the billing unit of leased time.

The *cost effective genetic algorithm* (CEGA) [8] is a meta-heuristic algorithm that solves the problem of minimizing makespan while meeting the deadline. It is based on the genetic algorithms (GA). CEGA encodes the scheduling order, task-service assignment and task-processor assignment together into a chromosome.

Figure 2 shows an example of encoding in CEGA. For example, t_2 is the third task to be executed, and it is assigned to the second processor of the first service.

III. The Two-phase Genetic Algorithm

Our algorithm is based on the two-phase concept, and we refer to it as the *two-phase genetic algorithm*

(2pGA). In the first phase of 2pGA, we only encode the scheduling order and task-service assignment into a chromosome, excluding task-processor assignment. After a chromosome is generated in the first phase, we design a heuristic algorithm to determine task-processor assignment in the second phase. Our two-phase concept can be applied to other meta-heuristic algorithms as well.

The main steps of our algorithm are developed as follows.

Stage 1: Enhance the initial population with a small dataset
The initial solutions are generated by (1) Randomly generating initial solutions; (2) Generating initial solutions with the JITC algorithm [5]; (3) Generating initial solutions with the ICPCP algorithm [4]; (4) Employing both ICPCP and JITC algorithms as parts of the initial solutions; (5) First utilizing upward ranking to generate a scheduling order, then taking this scheduling order as input of JITC algorithm to derive a solution.

Stage 2: Perform the preliminary experiments of 5 algorithms with a small dataset
We incorporate our two-phase approach into 5 meta-heuristic algorithms, including genetic algorithm (GA), differential evolution (DE) [9], teaching-learning-based optimization (TLBO) [10], grey wolf optimizer (GWO) [11, 12] and search economics algorithm (SE) [13].

Stage 3: Design the hybrid algorithms
Based on the experimental results of stage 2, we find out that 2pGA has the best performance, while 2pDE performs the worst. Consequently, we exclude 2pDE from further consideration. Then we mix the remaining algorithms with some selected steps of 2pGA

Stage 4: Refine the two-phase GA (2pGA)
In the crossover step, we first select some elites for the next generation, then perform the tournament selection and employ the two-point crossover method [8]. In the mutation step, two possible operations are performed. (1) Select two random positions, and then they are swapped [8]. (2) Randomly select one task, and change its assigned service randomly.

Stage 5: Design a heuristic algorithm for task-processor assignment
In a chromosome of 2pGA, the service assigned to each task has been decided. Subsequently, we develop a heuristic algorithm to determine task-processor assignments, which are used to calculate the fitness score of 2pGA. The fitness of a chromosome is the total cost of a schedule, which is the sum of the costs across all processors.

Stage 6: Design a task-processor assignment algorithm with GA
After obtaining the scheduling solution using 2pGA, we enhance the solution further by designing a GA algorithm for task-processor assignment (tpGA).

Stage 7: Compare the performance of 2pGA and 1pGA
The one-phase GA (1pGA) is an implementation based on CEGA algorithm [8], and it serves as a baseline for comparing to our 2pGA. In our main experiment, we compare the performance of 2pGA to 1pGA.

Since 2pGA exhibits the best performance in Stage 3, we concentrate to improve its performance in Stages 4 to 6, including both scheduling results and execution time of 2pGA.

IV. The Algorithm for Task-processor Assignment

To accomplish the task-processor assignment, we propose two strategies as follows.

(1) To minimize the additional leased time. A processor is suitable if it finishes the task before its latest finish time and has the minimum additional leased time. If no processor meets these criteria, we allocate a new processor of the assigned service to the task.

(2) To maximize the utilization of the remaining leased time. We choose a processor that completes the task before its latest finish time and utilizes the maximum remaining leased time. If there is no such processor, we assign a new processor of the assigned service to the task.

Since these two strategies may select two different processors, we select the processor with a lower cost.

We use an example to explain our task-assignment method. Figure 3 shows a simple DAG, in which the computation time and the cost of each service are shown in Tables IIa and IIb, respectively. The latest finish time (`lft`) can be calculated with a bottom-up scheme, where the `lft` of an exit node is set to the deadline. Table IIc shows the `lft` of each task. t_4 is an exit note, so its `lft` is set the deadline 30. The possible communication time from t_2 to t_4 is 8, and the fast execution time of t_4 is 7 in service. Thus, the `lft` of t_2 is $30 - (8 + 7) = 15$.

Suppose that the scheduling order and assigned service have been decided in the first phase of 2pGA, as shown in Figure 4. The scheduling order is t_1,

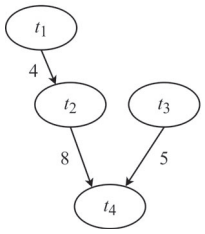


Fig. 3: A DAG for explaining our task-processor assignment.

TABLE II: The computation time and cost in different services for the DAG of Figure 1. (a) The computation time. (b) The cost of each service.(c) The latest finish time of each task.

	s_1	s_2
t_1	10	12
t_2	1	9
t_3	6	3
t_4	7	11
deadline = 30		

(a)

	s_1	s_2
cost	5	3
billing unit = 10		

(b)

	t_1	t_2	t_3	t_4
lft	10	15	18	30

(c)

t_3, t_2, t_4 and the tasks are all assigned to service 1. Again, suppose that the processors of t_1, t_3 and t_2 have been already assigned to $p_{1,1}$ (from time 1 to 10), $p_{1,1}$ (from time 11 to 16) and $p_{1,2}$ (from time 15 to 15), respectively.

Now, the last task to be scheduled is task t_4. The scheduling results of t_4 on $p_{1,2}$ and $p_{1,1}$ are shown in Figures 5a and 5b, respectively. Figure 5a tries to assign $p_{1,2}$ to t_4. Since the billing unit is 10, $p_{1,1}$ has been leased from 1 to 20 for executing t_1 and t_3 (from time 1 to 16), and $p_{1,2}$ has been leased from 15 to 24 for executing t_2 (from time 15 to 15). Then, there is only 1 additional leased time unit at 21 on $p_{1,1}$ and 4 additional leased time units from 25 to 28 on $p_{1,2}$, totally 5 additional leased time units.

Figure 5b tries to assign $p_{1,1}$ to t_4. There are

	t_1	t_2	t_3	t_4
scheduling order	1	3	2	4
assigned service	1	1	1	1

Fig. 4: An encoding of Figures 3, decided by the first phase of 2pGA.

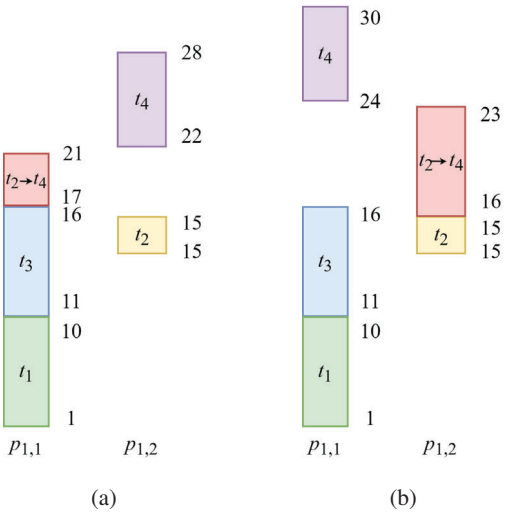


Fig. 5: The scheduling results. (a) minimizing the additional leased (b) maximizing the utilization of the remaining leased time.

totally 10 additional leased time unit from 21 to 30 on $p_{1,1}$. Since there is less additional leased time unit for assigning t_4 on $p_{1,2}$, following the strategy of minimizing the additional leased time, we assign t_4 on $p_{1,2}$.

In Figure 5a, if $p_{1,2}$ is assigned to t_4, it uses 4 remaining leased time units from 17 to 20 on $p_{1,1}$ (communication time from 17 to 21) and 3 remaining leased time units from 22 to 24 on $p_{1,2}$ (execution time from 22 to 28), totally 7 remaining leased time units. In Figure 5b, if $p_{1,1}$ is assigned to t_4, it uses 8 remaining leased time units from 16 to 23 on $p_{1,2}$ (communication time from 16 to 23). Since assigning t_4 to $p_{1,1}$ utilizes more of the remaining leased time units, following the strategy of maximizing the utilization of the remaining leased time, we assign t_4 to $p_{1,1}$.

Finally, t_4 is assigned on $p_{1,2}$, since the scheduling result has a lower cost from the strategy of minimizing the additional leased time.

V. Experimental Results

Our test datasets are randomly generated with various parameters, including number of tasks, number of services, computation to communication ratios (CCR), shape factor, and heterogeneity factor. The parameters are described as follows.

- n: The number of tasks (nodes) in the DAG.
- m: The number of services.
- CCR: The approximate ratio of the communication time to the computation time for a task.

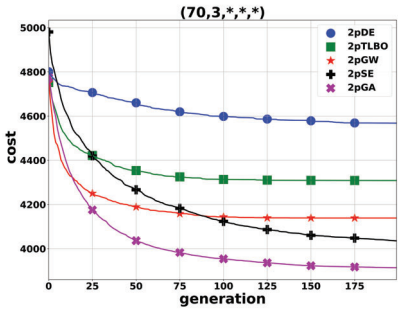


Fig. 6: The performance comparison of five algorithms by the average of 10 independent runs with $n = 70$ and $m = 3$.

While CCR is low, it means the task is computationally intensive, rather than communication intensive.

- α: Shape factor α decides that there are l levels in the DAG, where $l = \frac{\sqrt{n}}{\alpha}$. When α is large, the parallelism of the DAG tends to be high. Conversely, when α is small, the parallelism of the DAG tends to be low.
- γ: Heterogeneity factor γ determines the variation of the computation on all services. The computation time is randomly generated in the range of $W \times (1-\gamma)$ and $W \times (1+\gamma)$, where W is a random positive integer. When γ is large, there is a significant variation in computation time among different services.

In the following figures and tables, we represent the settings of our experimental parameters in the form of (n, m, CCR, α, γ). Taking (50, 6, *, *, *) as an example, the number n of tasks is set to 50, the number m of services is set to 6, the ratio of communication time to computation time $CCR \in \{0.1, 0.5, 1, 5, 10\}$, the shape factor $\alpha \in \{0.5, 1, 1.5\}$, and the heterogeneity factor $\gamma \in \{0.1, 0.5, 0.9\}$.

The results of preliminary experiments for GA, DE, TLBO, GWO and SE are shown in Figure 6. Since all of them are based on the two-phase concept, they are prefixed with '2p.' As one can see, 2pGA has the best performance.

For task-processor assignment, we compare our heuristic algorithm to tpGA. There is a significant difference in required time between the two approaches, as shown in Table IIIb. However, the improvement achieved by tpGA is only 0.037, which is not significant, as shown in Table IIIba

In our main experiment, we use the results obtained by the 1pGA as a benchmark to calculate the improvement achieved by 2pGA. The experimental results are shown in Figures 7.

TABLE III: Comparison of tpGA and 2pGA. (a) The cost improvement of tpGA compared to 2pGA. (b) The computation time of tpGA and our heuristic algorithm for task-processor assignment.

Improved cases	Improvement
269/400	0.037

(a)

tpGA	heuristic algorithm
12806.4 Sec	7.634 Sec

(b)

Among all parameters, we find that CCR and shape factor have a greater impact on the results. As the CCR value is increased, our improvement ratio also rises. We believe this is because as the communication time increases, the proportion of communication cost to the total cost also grows. Additionally, when the shape factor decreases, the number of layers in the DAG increases, leading to a decrease in parallelism. In such cases, it becomes more important to place tasks with precedence relationships on the same processor.

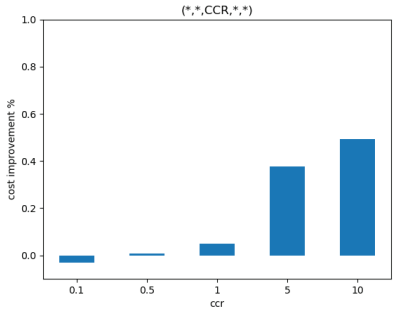
Our 2pGA effectively reduces the solution space. Furthermore, our task-processor assignment algorithm can find a good assignment. Thus, the overall results are improved.

VI. Conclusion

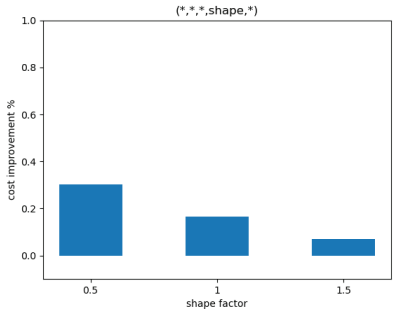
To solve the task scheduling problem, the previous approaches usually encoded scheduling order, task-service assignment and task-processor assignment together into the chromosome. In this paper, we propose a two-phase approach that integrates a heuristic method into GA. During the first phase, only scheduling order and task-service assignment are encoded. In the second phase, we utilize heuristic algorithms to decide the task-processor assignment. Our two-phase approach can significantly reduce the effort required for searching solution space.

As the experimental results show, our algorithm significantly improves upon the traditional GA when the DAG graph has large communication time and many layers. Conversely, when the communication time is small, our algorithm may yield slightly inferior performance. This outcome is because in such cases, the significance of placing tasks with precedence relationships on the same processor is small.

In the future, we may incorporate more efficient search methods into the meta-heuristic algorithms to



(a)



(b)

Fig. 7: The improvement of our 2pGA to 1pGA with various parameters. (a) CCR. (b) Shape factor.

improve the search efficiency. These advancements may further enhance the efficiency and effectiveness of our approach.

References

[1] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.

[2] Christina Hoffa, Gaurang Mehta, Tim Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman, and John Good. On the use of cloud computing for scientific workflows. In *2008 IEEE Fourth International Conference on eScience*, pages 640–645, Indianapolis, USA, 2008.

[3] Oliver Sinnen. *Task Scheduling*, chapter 4, pages 74–107. John Wiley & Sons, Ltd, 2007.

[4] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H.J. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158–169, 2013.

[5] Jyoti Sahni and Deo Prakash Vidyarthi. A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment. *IEEE Transactions on Cloud Computing*, 6(1):2–18, 2018.

[6] Koneti Kalyan Chakravarthi, L. Shyamala, and V. Vaidehi. Cost-effective workflow scheduling approach on cloud under deadline constraint using firefly algorithm. *Applied Intelligence*, 51(3):1629–1644, 2021.

[7] Maria Alejandra Rodriguez and Rajkumar Buyya. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 2(2):222–235, 2014.

[8] Jasraj Meena, Malay Kumar, and Manu Vardhan. Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. *IEEE Access*, 4:5065–5082, 2016.

[9] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

[10] R Venkata Rao, Vimal J Savsani, and DP Vakharia. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3):303–315, 2011.

[11] Tianhua Jiang and Chao Zhang. Application of grey wolf optimization for solving combinatorial problems: Job shop and flexible job shop scheduling cases. *IEEE Access*, 6:26231–26240, 2018.

[12] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61, 2014.

[13] Chun-Wei Tsai. Search economics: A solution space and computing resource aware search method. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2555–2560, Hong Kong, China, 2015.