


RESEARCH

Open Access



Scheduling multi-task jobs with extra utility in data centers

Xiaolin Fang^{1*} , Junzhou Luo¹, Hong Gao², Weiwei Wu¹ and Yingshu Li³

Abstract

This paper investigates the problem of maximizing utility for job scheduling where each job consists of multiple tasks, each task has utility and each job also has extra utility if all tasks of that job are completed. We provide a 2-approximation algorithm for the single-machine case and a 2-approximation algorithm for the multi-machine problem. Both algorithms include two steps. The first step employs the Earliest Deadline First method to compute utility with only extra job utility, and it is proved that it obtains the optimal result for this sub-problem. The second step employs a Dynamic Programming method to compute utility without extra job utility, and it also derives the optimal result. An approximation result can then be obtained by combining the results of the two steps.

Keywords: Multi-task jobs, Extra utility, Scheduling

1 Introduction

Job scheduling is a widely studied topic in computer science. Many systems such as parallel and distributed computing, cloud computing, workforce management, energy management, and network communications require scheduling of jobs [1–5]. There are many studies designing efficient approaches to solve the job scheduling problem so as to improve the resultant performance subject to the resource constraints [6–9].

Many applications prefer to divide large jobs into multiple small tasks to better utilize the limited resources and provide better service quality. As stated in [10], most interactive services such as web search, social networks, online gaming, and financial services now are heavily dependent on computations at data centers because their demands for computing resources are both huge and dynamic. Interactive services are time-sensitive as users expect to receive a complete or possibly partial response within a short period of time. Thus, a job should be preemptive and it can be divided into many small tasks (we call this as multi-task problem) in order to provide interactive services and improve the utilization ratio of the computing resources.

We study the multi-task job scheduling problem in this paper. Usually, the aim of multi-task job scheduling is to maximize the profit or minimize the cost while subject to the resource and deadline constraints. This paper also studies the profit maximization problem for multi-task job scheduling where each job has a starting time and an ending time. The profit is called *utility* in this paper. The utility of a task or job can be obtained only if the task or job is completed. Most state of art works study the problem considering either the utility of the tasks or the utility of the jobs. Few works consider both the utility of the tasks and jobs. In this paper, we study the problem of multi-task job scheduling at a data center with the goal of maximizing the total utility of all the jobs, where each job is decomposed into multiple tasks, and both a job and a task have their own utility. That is, each task has its own utility and each job also has an extra utility which can only be obtained when all its tasks are completed.

The problem investigated in this paper is particularly challenging because it is quite difficult to decide whether it is better to schedule a job as a whole or to schedule the tasks of the job separately. Furthermore, it is difficult to make correct decisions for current jobs because the requirements of the incoming jobs are unknown.

We first study the single-machine problem where there is only one machine that can be used. The single-machine

*Correspondence: xiaolin@seu.edu.cn

¹School of Computer Science and Engineering, Southeast University, Nanjing, China

Full list of author information is available at the end of the article

problem expects a method to schedule the jobs on one machine while satisfying the resource and deadline constraints. We then study the multi-machine problem where multiple machines can be employed. Because of the NP-completeness of the problem, we present two corresponding 2-approximation algorithms for the two problems.

For simplicity, we only consider the problem where every task has uniform resource requirement, but the utility of the tasks could be different. Tasks have arbitrary resource requirements will be studied in the future work. Our contributions are summarized as follows.

- To the best of our knowledge, this is the first work to study the problem considering both task utility and job utility.
- A 2-approximation algorithm is provided for the single-machine problem. This algorithm includes an Earliest Deadline First (EDF) scheduling and a Dynamic Programming (DP) algorithm.
- Another 2-approximation algorithm is provided for the multi-machine problem. Similar to the algorithm for the single-machine problem, this algorithm also employs an EDF scheduling and a DP algorithm.

The rest of the paper is organized as follows. Section 2 introduces the related works. Section 3 presents the problem formulation. Section 4 studies the single-machine problem. The multi-machine problem is studied in Section 5. And Section 6 concludes the paper.

2 Related works

The job scheduling problem can be classified into multiple classes, such as single or multiple tasks, single or multiple machines, and identical or unrelated machines. Usually, the input of the problem involves n jobs and k machines. Each job is associated with a release time, a deadline, a weight, and a processing time on each machine. The goal is to find a non-preemptive schedule that maximizes the weight of the jobs subject to their respective deadlines. Garey and Johnson [11, 12] show that the simplest instance of the decision problem corresponding to this problem is NP-complete.

Bar-Noy et al. [13, 14] study the scheduling problem where each job includes a single task. The authors present a 3-approximation algorithm using the local ratio technique. For arbitrary job weights and a single machine, an LP formulation achieves a 2-approximation for polynomially bounded integral input and a 3-approximation for arbitrary input. For unrelated machines, the factors are 3 and 4, respectively. Because of the high time complexity of the LP-based method, Bar-Noy et al. [13] also provide a combinatorial approximation algorithm whose approximation factor is $3 + 2\sqrt{2}$. Independently, Calinescu et al.

[15] designed a 3-approximation algorithm via rounding linear programming solutions.

The preemptive version of the single-task problem for a single machine was studied by Lawler [16]. For identical job weights, Lawler showed how to apply the dynamic programming techniques to solve the problem in polynomial time. The same techniques are employed to obtain a pseudopolynomial algorithm for the NP-hard variant in which the weights are arbitrary. Lawler [17] also designed polynomial time algorithms that solve the problem in two special cases: (i) the time windows in which jobs can be scheduled are nested, and (ii) the weights and processing times are in opposite order. Kise et al. [18] showed how to solve the special case where the release times and deadlines are similarly ordered.

Some works [19–22] study the problem where each job has multiple tasks, which is called the SplitJob problem. In the SplitJob problem, a task does not have a window within which the tasks can be scheduled. That is, the tasks can only be decided to be scheduled or not. The unit height case of the basic SplitJob problem has been addressed by finding the maximum weight independent sets in interval graphs [19, 20]. Bar-Yehuda et al. [21] present a $(2r)$ -approximation algorithm, where r is the number of the tasks in a job. They also proved a hardness result indicating it is NP-hard to approximate the problem within a factor of $O(r/\log r)$. Thus, their approximation ratio is near-optimal. Bar-Yehuda and Rawitz [22] studied the uniform case of the basic SplitJob problem and derived a $(6r)$ -approximation algorithm by utilizing the fractional local ratio technique.

Venkatesan et al. [23] study the problem of maximizing the throughput of jobs where each job consists of multiple tasks. Different from the SplitJob problem, each task has a window where the task can be scheduled any time within the window subject to the processing length. The algorithm presented in [23] is an LP-based algorithm which gives $8r$ -approximation.

All the above works either consider the utility of tasks or the utility of jobs. In this paper, we study the problem where each job consists of multiple tasks, each task has utility, and each job has extra utility if all its tasks are completed.

A closely related problem is considered by Zheng et al. in [10] which study the problem of scheduling interactive jobs at a data center with the goal of maximizing the total utility of all the jobs. In their problem, the utility of a job is a function of the completed workload of that job. That is, the utility of a job varies when the completed workload of that job increases. The presented function in their work is nonlinear and concave. If the scheduling can be preemptive, then the authors can provide an optimal solution to solve the problem.

3 System model and problem formulation

3.1 System model

Assume there are m physical machines $\{M_1, M_2, \dots, M_m\}$ and n jobs $\{J_1, J_2, \dots, J_n\}$ in the data centers. Each job J_i has a starting time s_i and an ending time e_i , i.e., $J_i = [s_i, e_i]$, which is called the *processing interval*. Each job needs to be completed within its own processing interval. Each job J_i consists of multiple tasks $\{T_{i1}, T_{i2}, \dots, T_{in_i}\}$, where n_i is the number of the tasks involved in job J_i . Each task T_{ij} has a *processing time* of length 1 and a utility u_{ij} , which means it needs 1 machine to take 1 unit of time to complete task T_{ij} , and if the task is completed, then the utility is u_{ij} . With the above assumption, it can be easily found that each task T_{ij} must be completed within the processing interval $[s_i, e_i]$; otherwise, the task is dropped. We define the *assignment* of a task T_{ij} as \emptyset or a sub-interval I_{ij} of 1 unit of length within the processing interval $[s_i, e_i]$, i.e., $I_{ij} = \emptyset$, or $I_{ij} \in [s_i, e_i]$ and $|I_{ij}| = 1$ for some machine M_k . Let $a(T_{ij}) = k$ indicate that task T_{ij} is assign to machine M_k . The empty assignment $I_{ij} = \emptyset$ indicates that the task is dropped. If a task is completed, then it has a non-empty assignment on a certain machine such that the assigned sub-interval does not overlap (or conflict) with any assignment on the same machine. We assume s_i and e_i are integers. One unit of time is called a *slot* in this paper. That is, a task needs to take one slot on a machine to be completed. We only consider the problem where each task T_{ij} has a processing time of length 1. Tasks with arbitrary processing times will be studied in the future work.

We consider a situation where the jobs belong to different users, and the users are always willing to encourage the data centers to complete all their tasks. Therefore, job J_i has an extra utility σ_i in this paper. If all the tasks of job J_i are completed, then the utility gain of this job is the sum of the utility of the tasks included in J_i and the extra utility σ_i , i.e. $u(i) = \sum_{j=1}^{n_i} u_{ij} + \sigma_i$. Otherwise, even one task of a job is not completed, the utility gain is the sum of the utility of the completed tasks, without the extra job utility, i.e. $u(i) = \sum_{j=1, I_{ij} \neq \emptyset}^{n_i} u_{ij}$.

3.2 Problem statement

Our problem is to find an assignment of all the tasks of n jobs to m machines such that the total utility is maximized satisfying that (a) a machine can only process one task at a time, (b) one task can only be processed at one machine, and it cannot be split any more, but the tasks of a job can be assigned to multiple machines. Then, we have

$$\max \sum_{i=1}^n u(i) \tag{1}$$

s.t.

$$I_{ij} \subseteq [s_i, e_i] \text{ for each task} \tag{2}$$

$$|I_{ij}| = \{0, 1\} \text{ for each task} \tag{3}$$

$$a(T_{ij}) = \{1, 2, \dots, m\} \text{ for each task} \tag{4}$$

$$I_{ij} \cap I_{i'j'} = \emptyset \text{ and } a(T_{ij}) = a(T_{i'j'}) \text{ for every two tasks} \tag{5}$$

It is easy to find that this problem is NP-complete. Consider a simple instance where there is only one machine in the problem, the processing interval of each job is $[0, T]$, each job J_i consists of n_i one-unit tasks, each job has extra utility σ_i , and all the tasks have no utility, i.e. $u_{ij} = 0$, then the problem is to find an assignment within the processing interval while maximizing the utility gain, which is equivalent to the well-known Knapsack problem which is NP-complete. For simplicity, the single-machine problem where there is only one machine can be used is first studied and then followed by the general case where there are multiple machines.

4 Algorithm design for single-machine problem

We first consider a simpler instance of this problem where there is only one machine. As stated in the previous section, even the single-machine problem is NP-complete. Therefore, we present a 2-approximation algorithm in this section. The main idea of this algorithm is to solve the problem in two steps. The first step is to solve the single-machine problem without considering extra job utility. The second step is to solve the single-machine problem by only considering the extra job utility. The final result of the single-machine problem can then be obtained by combining the results derived from the two steps.

4.1 Problem without extra job utility

In this step, we do not consider the extra job utility. Therefore, given n jobs, each job J_i has a processing interval $[s_i, e_i]$, each job J_i consists of n_i one-unit-length tasks, and each task T_{ij} has utility u_{ij} . This step is to find an assignment with the maximum utility gain in a single machine.

We first consider a special case where the utility of each task is 1. Then, the problem is to schedule as many tasks as possible. We introduce the earliest ending time first algorithm which is also called Earliest Deadline First (EDF) in other works and show that the EDF algorithm schedules the maximum number of tasks.

The EDF method always schedules the job with the earliest ending time first. Let J_i be the job with the earliest ending time. EDF scans the processing interval of J_i from s_i to e_i and schedules the tasks of J_i one by one to the unused slots. A slot is *unused* if no tasks are scheduled to this slot. If all the slots in $[s_i, e_i]$ are scanned, or all the tasks of J_i are scheduled, EDF begins to schedule the next job J_{i+1} .

Algorithm 1 EDF Algorithm

Input: $[s_i, e_i], n_i, 1 \leq i \leq n$
 Output: Scheduling the maximum number of tasks

- 1: Let J_1, J_2, \dots, J_n be sorted by the ending time in increasing order;
- 2: **for** $i = 1$ to n **do**
- 3: $j = 1$;
- 4: **for** $t = s_i$ to e_i **do**
- 5: **if** $j > n_i$ **then**
- 6: **break**;
- 7: **if** slot t is not used **then**
- 8: Schedule T_{ij} to t ;

Figure 1 shows an example for the EDF algorithm. In this example, there are four jobs. J_1 's processing interval is $[2, 4]$, and it has two tasks. J_2 's processing interval is $[5, 10]$, and it has three tasks. J_3 's processing interval is $[7, 11]$, and it has three tasks. J_4 's processing interval is $[0, 13]$, and it has five tasks. The EDF algorithm schedules the job with the earliest ending time, and the tasks of a job are always scheduled as early as possible in their processing interval. As illustrated in Fig. 1, the gray slots represent the scheduled tasks of the four jobs.

Theorem 1 *The EDF algorithm schedules the maximum number of tasks.*

Proof We prove the theorem by induction. Let $n = 1$, i.e., there is only one job, then it is easy to find that the EDF algorithm schedules the maximum number of tasks.

Assume the EDF algorithm schedules the maximum number of tasks when $n = k$.

Now, we prove that the theorem is correct when $n = k + 1$. If all the tasks of J_{k+1} can be scheduled in its processing interval, then the theorem is correct. Otherwise, not all the tasks of J_{k+1} can be scheduled, then there are two cases as follows.

1) If all the scheduled tasks of J_1 to J_k are within the processing interval $[s_{k+1}, e_{k+1}]$, it indicates that $s_{k+1} \leq \min_{1 \leq i \leq k} \{s_i\}$ and $\max_{1 \leq i \leq k} \{e_i\} \leq e_{k+1}$, then all the slots within $[s_{k+1}, e_{k+1}]$ are used.

2) Otherwise, some tasks of J_1 to J_k are scheduled before s_{k+1} , and some tasks of J_1 to J_k are scheduled within $[s_{k+1}, e_{k+1}]$. No tasks are scheduled after e_{k+1} because the ending time $\max_{1 \leq i \leq k} \{e_i\} \leq e_{k+1}$. We only need to consider whether the scheduled tasks of J_1 to J_k within $[s_{k+1}, e_{k+1}]$ can be moved before s_{k+1} . If we can, then more tasks of J_{k+1} can be scheduled. However, the EDF algorithm always schedules the tasks as earlier as possible; therefore, it is impossible to move some of these scheduled tasks earlier.

It completes the proof. □

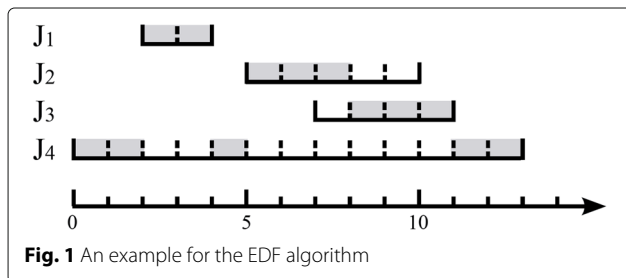
For simplicity of illustration, we explain the meaning of *link* and *reaching* which will be used frequently later. Both link and reach are defined towards the tasks/jobs which are not dropped (the scheduled tasks/jobs). In this paper, the links are directed.

1) The scheduled tasks of the same job link to each other. We call this the task link.

2) A scheduled job J_i links to another scheduled job J_j if there exists a scheduled task T of J_i which is scheduled within the processing interval of J_j . We call this the job link and call T the relay task. Figure 2 shows some job link examples. In this figure, the gray slots represent the scheduled tasks. In Fig. 2a, J_1 links to J_2 since there is a task T belonging to J_1 which is scheduled within the processing interval of J_2 . In Fig. 2b, J_1 and J_2 link to each other. Because jobs are scheduled one by one, there may be no task scheduled for the current job. As shown in Fig. 3, no tasks of J_3 are scheduled; however, J_2 still links to J_3 because task T' of J_2 is scheduled within the processing interval of J_3 .

3) Scheduled task T can reach T' belonging to another job when there exists a job link sequence $\langle J(T), J_{x_1}, J_{x_2}, \dots, J_{x_k}, J(T') \rangle$ where $J(T)$ links to J_{x_1} , J_{x_1} links to J_{x_2} , and J_{x_k} links to $J(T')$. $J(T)$ denotes the job including T . Figure 4 shows that J_1 can reach J_3 . In this example, any scheduled task of J_1 can reach any scheduled task of J_3 . Reaching is defined for task to task, task to job, and job to job.

We now present a EDF-based algorithm (shown in Algorithm 2). Same as the EDF algorithm, Algorithm 2 always schedules the job with the earliest ending time first. Let J_i be the job with the earliest ending time. EDF scans from s_i to e_i and schedules the tasks of J_i one by one to the unused slots. Note that the tasks of J_i are sorted by utility in non-increasing order. Thus, the task of J_i with the largest utility is scheduled first. Let the current task with the largest utility of J_i be T_{ij} . If there is an unused slot within $[s_i, e_i]$, T_{ij} is scheduled to the first unused slot in $[s_i, e_i]$. If there is no unused slot within $[s_i, e_i]$, then find the scheduled task with the least utility and implement a recursive replacement. The replacement is run towards the link sequence from the task with the least utility to the



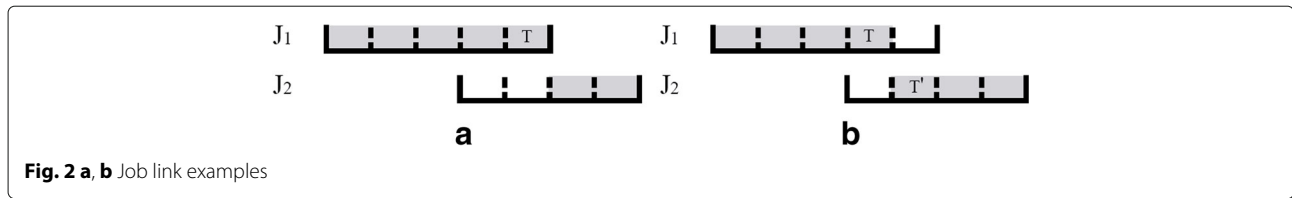


Fig. 2 a, b Job link examples

current job, and vice versa. That is, the replacement can also run in a reversed order of the link sequence. In the recursive replacement towards the link sequence, a prior relay task is always replaced with a later relay task.

Algorithm 2 EDF-based Algorithm

```

Input:  $[s_i, e_i], n_i, u_{ij}, 1 \leq i \leq n, 1 \leq j \leq n_i$ 
Output: Scheduling the tasks to maximize utility gain
1: Let  $J_1, J_2, \dots, J_n$  be sorted by ending time in non-
   decreasing order;
2: Let the tasks of each job be sorted by utility in non-
   increasing order;
3: for  $i = 1$  to  $n$  do
4:   for  $j = 1$  to  $n_i$  do
5:     if There is an unused slot in  $[s_i, e_i]$  then
6:       Find the earliest unused slot  $t$  in  $[s_i, e_i]$  and
       schedule  $T_{ij}$  to  $t$ ;
7:     else
8:       Find the task with the least utility which can
       reach  $J_i$  and let it be  $T$  and its utility be  $u$ ;
9:       if  $u < u_{ij}$  then
10:        Run a recursive replacement to remove  $T$ ;
    
```

Figure 5 shows an example for the EDF-based algorithm. In this example, there are 4 jobs. J_1 's processing interval is $[0, 7]$ and it has 5 tasks with utility 10, 9, 8, 7, and 6, respectively. J_2 's processing interval is $[3, 8]$, and it has 3 tasks with utility 10, 9, and 8, respectively. J_3 's processing interval is $[5, 9]$, and it has 2 tasks with utility 11 and 10, respectively. J_4 's processing interval is $[8, 11]$, and it has 3 tasks with utility 14, 13, and 12, respectively. The EDF-based algorithm always schedules the job with the earliest ending time first. In the first iteration, the EDF-based algorithm schedules the 5 tasks of J_1 to slots 1–5, respectively. In the second iteration, the EDF-based algorithm schedules the 3 tasks of J_2 to slots 6–8, respectively.

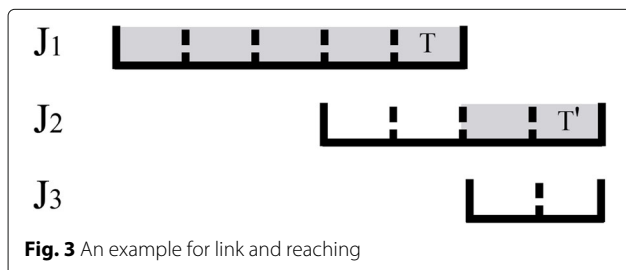


Fig. 3 An example for link and reaching

In the third iteration, the EDF-based algorithm schedules J_3 's task with a larger utility of 11 to slot 9, and then, it finds that there is no slot for J_3 's task with smaller utility of 10. The algorithm then finds the task with the smallest utility which can reach J_3 . The found task is the task of J_1 , and its utility is 6. In Fig. 6, the red links show a link sequence for the reachability from the task with utility 6 of J_1 to J_3 . The tasks pointed by the red arrows are the relay tasks. Every time a relay task is replaced by a later relay task in the link sequence. The algorithm replaces J_1 's task with utility 6 by J_2 's task with utility 8. In the recursive replacement, J_2 's task with utility 8 is replaced by J_3 's task with utility 10.

In the fourth iteration, the EDF-based algorithm schedules J_4 's tasks with utility 14 and 13 respectively, and then, there is no slot for J_4 's task with utility 12. The algorithm then finds the task with the smallest utility which can reach J_4 . The found task is the task of J_1 , and its utility is 7. In Fig. 7, the red links show a link sequence for the reachability from the task with utility 7 of J_1 to J_4 . The tasks pointed by the red arrows are the relay tasks. Every time a relay task is replaced by a later relay task in the link sequence. The algorithm replaces J_1 's task with utility 7 by J_2 's task with utility 9. In the recursive replacement, J_2 's task with utility 9 is replaced by J_3 's task with utility 11, and J_3 's task with utility 11 is replaced by J_3 's task with utility 12.

The final schedule is shown in Fig. 5. The algorithm schedules the maximum number of tasks achieving the maximum total utility. In Fig. 5, the gray slots have been scheduled with tasks, and the numbers on the gray slots represent task utility.

Theorem 2 The EDF-based algorithm is optimal.

Proof Theorem 1 proves that the EDF algorithm schedules the maximum number of tasks. We now only need

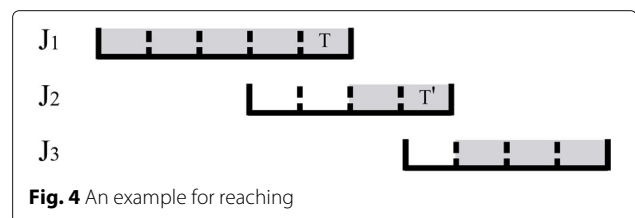
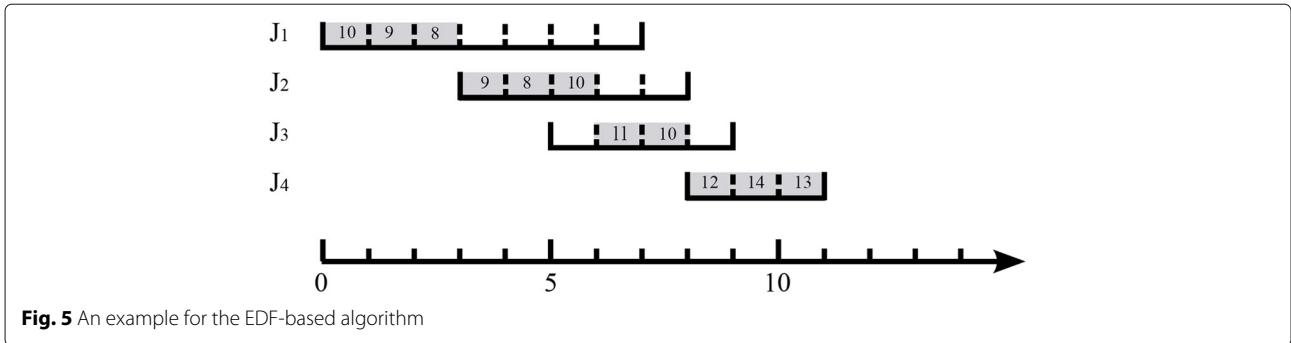


Fig. 4 An example for reaching



to prove that the EDF-based algorithm maximizes the total utility of the scheduled tasks. We also prove it by induction.

Let $n = 1$, that is, there is only one job. It is easy to find that the EDF-based algorithm maximizes the total utility of the scheduled task because the EDF-based algorithm always schedules the task with the largest utility first.

Assume the EDF-based algorithm maximizes the total utility of the scheduled tasks when there are k jobs, i.e., $n = k$.

Next, we prove that the theorem is correct when $n = k + 1$. Then, we need to prove the EDF-based algorithm obtains the optimal result for J_{k+1} .

If all the tasks of J_{k+1} can be scheduled in its processing interval, then the theorem is correct.

Otherwise, if not all the tasks of J_{k+1} can be scheduled, then the EDF-based algorithm first schedules as many tasks with the largest utility as possible to the unused slots. As stated in Theorem 1, the EDF algorithm cannot schedule more tasks. Therefore, it should determine whether to schedule the remaining tasks of J_{k+1} or not. For the remaining tasks, the EDF-based algorithm always uses them to replace the tasks with smaller utility. Without loss of generality, let T' be the task with the largest utility in the remaining tasks and its utility be u' . Let T be the task with the least utility that has been scheduled, and its utility be u . If $u < u'$, then the largest utility gain is achieved if T is replaced by T' . The utility gain is $u' - u_1 + u_1 - u_2 + u_2 - \dots - u_k + u_k - u = u' - u$, where $\{u_k, u_{k-1}, \dots, u_1\}$ are the utility of the relay tasks in the link sequence from T to T' . Because u is the minimum, $u' - u$ is maximum. This property holds for the remaining tasks. It completes the proof. \square

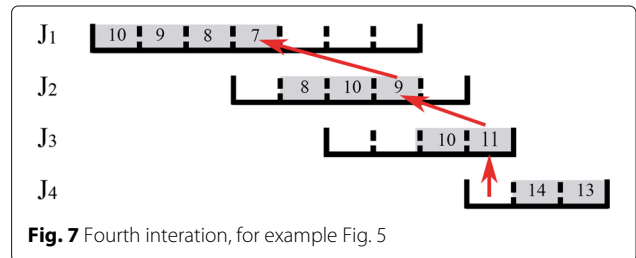
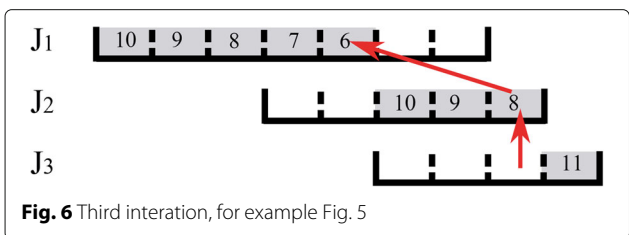
The time complexity of the EDF-based algorithm is $O(n^2r^2)$, where n is the number of jobs and r is the maximized number of the tasks of a job. In the EDF-based algorithm, the jobs are scheduled according to the ending time one by one. For the tasks of each job, the algorithm needs to find a task with the smallest utility that has already been scheduled. It needs $O(nr)$ time to search the task in a directed graph constructed by the link relation for tasks to tasks, tasks to jobs, and jobs to jobs. It also needs $O(n)$ time to update the graph any time the graph is changed, i.e. a replacement is implemented.

$$C_i(s, \sigma) = \min \begin{cases} C_{i-1}(s, \sigma) \\ \max\{s_i, C_{i-1}(s, \sigma - \sigma_i)\} + n_i \\ \min_{s', \sigma'} \{C_{i-1}(s', \sigma') + \max\{0, n_i - s' + s_i + P_{i-1}(s', \sigma - \sigma_i - \sigma')\}\} \end{cases} \quad (6)$$

$$P_{i-1}(s, s', \sigma'') = \min \begin{cases} P_{i-1}(s^+, s', \sigma'') \\ \min_{0 < \sigma' \leq \sigma''} \{\max\{0, C_{i-1}(s, \sigma') - s_i + P_{i-1}(s'', s', \sigma'' - \sigma')\}\} \end{cases} \quad (7)$$

4.2 Problem with only extra job utility

Now, we address the problem where each job has only extra utility and every task does not have utility. If all the tasks of a job are scheduled, then the job obtains the extra utility. Even one task of a job is not scheduled, the job loses the extra utility. This problem is similar to the one studied in [16] where given n jobs with arbitrary processing time, release dates and due dates, and the job can be scheduled



preemptively; the objective is to minimize the sum of the weights of the later jobs. The scheduling of our problem is not preemptive, but the processing time of each task is one unit of time. The preemptive scheduling and a task with one unit of processing time in our problem are similar. Therefore, minimizing the sum of the weights of the later jobs is the same as maximizing the sum of the utility of the scheduled jobs in our problem. The authors in [16] give a pseudo polynomial time Dynamic Programming (DP) algorithm. We also adopt this algorithm to solve the problem. The DP formulations are represented by Eqs. (6) and (7).

Given a job set J , let $s(J) = \min_{J_i \in J} \{s_i\}$ be the minimum starting time of J , $p(J) = \sum_{J_i \in J} n_i$ be the total processing time of J , $\sigma(J) = \sum_{J_i \in J} \sigma_i$ be the total extra utility of J , and $c(J)$ be the time the last job in J is completed in an EDF schedule.

One can refer to article [16] for the detailed algorithm. As stated in that work, assume the jobs are ordered by the ending time in non-decreasing order. Let s be a starting time, and σ be an integer representing utility. $C_i(s, \sigma)$ is defined as the minimum value of $c(J)$ with respect to feasible set $J \subseteq \{J_1, J_2, \dots, J_i\}$, with $s(J) \geq s$ and $\sigma(J) \geq \sigma$. If there is no such feasible set J , then $C_i(s, \sigma) = +\infty$. Accordingly, the final result that maximizing the weight of a feasible set is given by the largest value of σ such that $C_n(s_{\min}, \sigma)$ is finite, where $s_{\min} = \min_{1 \leq i \leq n} \{s_i\}$.

If job J_i cannot be contained in a feasible set J , i.e. $s_i < s(J)$, then $C_i(s, \sigma) = C_{i-1}(s, \sigma)$.

Otherwise, if job J_i can be contained in the feasible set J , there exists two cases.

In the first case, job J_i starts after $c(J - \{J_i\})$. Either $c(J - \{J_i\}) \leq s_i$, then $C_i(s, \sigma) = s_i + n_i$; or $c(J - \{J_i\}) > s_i$ and the scheduled tasks in the interval $[s_i, c(J - \{J_i\})]$ are continuous for $J - \{J_i\}$, then $C_i(s, \sigma) = C_{i-1}(s, \sigma - \sigma_i) + n_i$. Thus, $C_i(s, \sigma) = \max\{s_i, C_{i-1}(s, \sigma - \sigma_i)\} + n_i$.

In the second case, job J_i starts before $c(J - \{J_i\})$, which indicates there is an idle time between s_i and $c(J - \{J_i\})$. Let J' be the last set of jobs scheduled continuously before $c(J - \{J_i\})$ for $J - \{J_i\}$. Then, $c(J') = C_{i-1}(s(J'), \sigma(J'))$. Let it be $c(J') = C_{i-1}(s', \sigma')$ for simplicity.

Let $P_{i-1}(s, s', \sigma'')$ be the minimum number of tasks scheduled between s_i and s' , with respect to feasible set $J'' \subseteq \{J_1, J_2, \dots, J_{i-1}\}$ with $s(J'') \geq s$, $c(J'') \leq s'$, and $\sigma(J'') \geq \sigma''$. Note that it is the minimum number of tasks scheduled in interval $[s_i, s']$, rather than $[s, s']$. Then, the number of slots available for job J_i between s_i and s' can be represented as $s' - s_i - P_{i-1}(s, s', \sigma - \sigma_i - \sigma')$. Thus, the completing time $C_i(s, \sigma) = C_{i-1}(s', \sigma') + \max\{0, n_i - s' + s_i + P_{i-1}(s, s', \sigma - \sigma_i - \sigma')\}$.

Enumerate every s' and σ' . We can get $C_i(s, \sigma) = \min_{s' > s, \sigma' < \sigma} \{C_{i-1}(s', \sigma') + \max\{0, n_i - s' + s_i + P_{i-1}(s, s', \sigma - \sigma_i - \sigma')\}\}$. The enumeration of s is among all the starting times of the jobs, rather than among all the possible

times, which can drastically reduce the computation complexity.

The computation of $P_{i-1}(s, s', \sigma'')$ is as follows. Let $J'' \subseteq \{J_1, J_2, \dots, J_{i-1}\}$ be the set of jobs which realize $P_{i-1}(s, s', \sigma'')$. Then, there exists two cases.

If $s(J'') > s$, then $P_{i-1}(s, s', \sigma'') = P_{i-1}(s(J''), s', \sigma'')$. Enumerate every $s^+ > s$ and find the minimum one, then $P_{i-1}(s, s', \sigma'') = \min_{s^+ > s} \{P_{i-1}(s^+, s', \sigma'')\}$.

Otherwise, if $s(J'') = s$ and the scheduling of J'' is not continuous, let J' be the first set of jobs which run continuously and $s(J') = s$, then the total number of tasks scheduled within $[s_i, C_{i-1}(s, \sigma(J'))]$ is $\max\{0, C_{i-1}(s, \sigma(J')) - s_i\}$. We now need to compute the number of tasks which can be scheduled within $[C_{i-1}(s, \sigma(J')), s']$. It is easy to find that $P_{i-1}(s, s', \sigma'')$ can be represented as $\max\{0, C_{i-1}(s, \sigma(J')) - s_i\} + P_{i-1}(s', s', \sigma'' - \sigma(J'))$, where s' is the minimum starting time greater than or equal to $C_{i-1}(s, \sigma(J'))$. For simplicity, let $\sigma' = \sigma(J')$. Enumerate every σ' . We have $P_{i-1}(s, s', \sigma'') = \min_{0 < \sigma' \leq \sigma''} \{\max\{0, C_{i-1}(s, \sigma') - s_i\} + P_{i-1}(s', s', \sigma'' - \sigma')\}$.

The initial conditions are shown in Eqs. (8) to (11).

$$C_0(s, 0) = s, \quad \text{for all starting time } s \quad (8)$$

$$C_0(s, w) = +\infty, \text{ for all starting time } s \text{ and } w > 0 \quad (9)$$

$$P_{j-1}(s, s', 0) = 0, \quad \text{for } j = 1, 2, \dots, n \quad (10)$$

$$P_0(s, s', \sigma'') = +\infty, \quad \text{for } \sigma'' > 0 \quad (11)$$

The time and space complexities for this DP algorithm are $O(n^3 \sigma^2)$ and $O(n^2 \sigma)$, respectively, where n is the number of the jobs and σ is the sum of the utility of the jobs. It can be easily found that the time complexity of the DP algorithm is pseudo polynomial because the DP formula includes an integer input σ which can be extremely large in real systems. Therefore, we provide a theoretical approximation solution for this problem.

4.3 Approximation algorithm for single-machine problem

The main idea of the approximation algorithm is to get two intermediate results using the two algorithms independently and then combine the two results. First, it uses the EDF-based algorithm to solve the problem without extra job utility. Second, it uses the DP algorithm to solve the problem with only extra job utility. Then, it selects a larger one from the two scheduling results. Algorithm APPX1 shows the detailed approximation algorithm.

Theorem 3 *The APPX1 algorithm is a 2-approximation algorithm.*

Proof Let OPT be the utility obtained by an optimal solution and ALG be the utility obtained by the APPX1

Algorithm 3 APPX1 algorithmInput: $[s_i, e_i], \sigma_i, n_i, u_{ij}, 1 \leq i \leq n, 1 \leq j \leq n_i$

Output: Scheduling the tasks to maximize utility gain

- 1: Use the EDF-based algorithm to solve the problem without extra utility;
- 2: Use the DP algorithm to solve the problem with only extra utility;
- 3: Let \mathcal{T} be the set of tasks scheduled by the EDF-based algorithm;
- 4: Let \mathcal{J} be the set of jobs scheduled by the DP algorithm;
- 5: Let $u(\mathcal{T})$ be the total utility of \mathcal{T} including the utility of the tasks in \mathcal{T} and the extra utility of the completed jobs among \mathcal{T} ;
- 6: Let $u(\mathcal{J})$ be the total utility of \mathcal{J} including the extra utility of the jobs in \mathcal{J} and the utility of all the tasks of \mathcal{J} ;
- 7: **if** $u(\mathcal{T}) < u(\mathcal{J})$ **then**
- 8: Use the scheduling result of the DP algorithm;
- 9: **else**
- 10: Use the scheduling result of the EDF-based algorithm;

algorithm. It is easy to find that OPT can be represented as $\text{OPT} = u + \sigma$, where u is the total utility of the scheduled tasks and σ is the total extra utility of the entirely scheduled jobs in an optimal solution. Let u' be the total utility obtained by the EDF-based algorithm, and σ' be the total utility obtained by the DP algorithm. From the earlier analysis, both the EDF-based algorithm for the problem without extra utility and the DP algorithm for the problem with only extra utility are optimal. Thus, $u \leq u'$ and $\sigma \leq \sigma'$, and then, we have $\text{OPT} \leq u' + \sigma'$. ALG actually can be represented as $\text{ALG} \geq \max\{u', \sigma'\}$. Therefore, $\text{OPT} \leq 2\text{ALG}$. It completes the proof. \square

4.4 An improvement for the DP algorithm

Recall the definition of $C_i(s, \sigma)$, it is the minimum value of $c(J)$ with respect to feasible set $J \subseteq \{J_1, J_2, \dots, J_i\}$, with $s(J) \geq s$ and $\sigma(J) \geq \sigma$. It can be found that, in the DP recursion formula $C_i(s, \sigma)$, the parameter σ is build only on the extra utility of the scheduled jobs, but does not consider the utility of the tasks of the scheduled jobs. This can be improved by computing $C_i(s, u)$ where the parameter is build on the total utility of the scheduled jobs. Let $u_i = \sum_{j=1}^{n_i} u_{ij} + \sigma_i$. Use u_i to replace σ_i in the DP formula, then $C_i(s, u)$ represents the minimum value of $c(J)$ with respect to feasible set $J \subseteq \{J_1, J_2, \dots, J_i\}$, with $s(J) \geq s$ and $u(J) \geq u$, where $u(J)$ includes the utility of all the tasks of the jobs in J and the extra utility of the jobs in J . Such modification consider the extra utility of the scheduled jobs and also the utility of the tasks of the scheduled jobs. It

can improve the result derived by the DP algorithm when the utility of the tasks takes a large proportion comparing with the extra utility. However, when the extra utility takes a large proportion (for example, in a worst case where all the tasks have no utility, the jobs have only the extra utility), the improvement is little. Algorithm APPX1's use of such modification of the DP algorithm cannot improve the approximation ratio, but may improve the results in many scenarios.

5 Algorithm design for multi-machine problem

The solution for the multi-machine problem is similar to the single-machine problem. It also includes two steps. The first step is to schedule the tasks without considering the extra utility of the jobs. The second step is to schedule the jobs only considering the extra utility of the jobs. And finally, select a better schedule from the two steps.

5.1 Problem without extra utility

The EDF-multi-algorithm is similar to the one for the single-machine problem. The difference is that every time the algorithm needs to schedule a task, it finds the earliest unused slot in the task's processing interval among all the machines, while the algorithm for the single-machine problem just needs to search in one machine. If there is no unused slot, the replacement function also finds the task with the smallest utility that has been scheduled among all the machines. The detailed algorithm is shown in Algorithm 4. The obtained result is optimal for the problem without extra utility.

Algorithm 4 EDF-Multi algorithmInput: $[s_i, e_i], n_i, u_{ij}, 1 \leq i \leq n, 1 \leq j \leq n_i$

Output: Scheduling the tasks to maximize utility gain

- 1: Let J_1, J_2, \dots, J_n be sorted by ending time in non-decreasing order;
- 2: Let the tasks of each job be sorted by utility in non-increasing order;
- 3: **for** $i = 1$ to n **do**
- 4: **for** $j = 1$ to n_i **do**
- 5: **if** There is an unused slot within $[s_i, e_i]$ in some machine **then**
- 6: Find the earliest unused slot t in $[s_i, e_i]$ among all the machines, and let it be machine M_k . Schedule T_{ij} to t at machine M_k ;
- 7: **else**
- 8: Find the task with the smallest utility that has been scheduled among all the machines and let it be T and its utility be u ;
- 9: **if** $u < u_{ij}$ **then**
- 10: Run a recursive replacement to remove T ;

5.2 Problem with only extra utility

We design an algorithm for the multi-machine problem with only extra utility by adopting the idea of the DP algorithm for the single-machine problem. Given job set J , let $s(J) = \min_{J_i \in J} \{s_i\}$ be the minimum starting time of J , $p(J) = \sum_{J_i \in J} n_i$ be the total processing time of J , and $\sigma(J) = \sum_{J_i \in J} \sigma_i$ be the total extra utility of J .

Define $c(J) = \langle t, j \rangle$ as a 2-tuple where t is the time and j is the number of used machines at t that the last job in J is completed in an EDF schedule.

Define $\langle t, j \rangle + p = \left\langle t + \left\lfloor \frac{j+p+1}{m} \right\rfloor, (j+p) \bmod m \right\rangle$ which represents scheduling p tasks from time t and after machine M_j continuously. Because there are m machines, m tasks can be scheduled in each slot. We focus on how many machines are used in slot t rather than which machines are used. Without loss of generality, we assume $\langle t, j \rangle$ represent in slot t , machine M_1 to M_j are used. Therefore, the ending time after scheduling p tasks from time t and after machine M_j continuously is $t + \left\lfloor \frac{j+p+1}{m} \right\rfloor$, and at the ending time, $(j+p) \bmod m$ machines are used. In this paper, $\langle t+1, 0 \rangle$ equals to $\langle t, m \rangle$ representing in slot t , all the m machines are used.

Define $\langle t, j \rangle < \langle t', j' \rangle$ if $t < t'$ or $t = t'$ and $j < j'$. It represents that $\langle t, j \rangle$ is earlier than $\langle t', j' \rangle$. $\langle t, j \rangle = \langle t', j' \rangle$ only if $t = t'$ and $j = j'$.

We can regard the scheduling process as putting tasks to a 2-dimensional array from top to bottom and from left to right. Given a starting place $\langle s, j \rangle$, the tasks can be scheduled $\langle s, j+1 \rangle$ to $\langle s, m \rangle$, $\langle s+1, 1 \rangle$ to $\langle s+1, m \rangle$, $\langle s+2, 1 \rangle$ to $\langle s+2, m \rangle$, and so on. Therefore, $\langle t', j' \rangle - \langle t, j \rangle$ can be regarded as how many tasks can be put from $\langle t, j \rangle + 1$ to $\langle t', j' \rangle$.

Let s be a starting time, j be the number of used machines in s , and σ be an integer representing utility. Similarly to but different from the single-machine problem, $C_i(\langle s, j \rangle, \sigma)$ is defined as the minimum value of $c(J)$ with respect to feasible set $J \subseteq \{J_1, J_2, \dots, J_i\}$, with $s(J) \geq s$, $\sigma(J) \geq \sigma$ and j machines are used in s . If there is no such feasible set J , then $C_i(\langle s, j \rangle, \sigma) = \langle +\infty, +\infty \rangle$. Accordingly, the final result maximizing the utility of a feasible set is given by the largest value of σ such that $C_n(\langle s_{\min}, 0 \rangle, \sigma)$ is finite, where $s_{\min} = \min_{1 \leq i \leq n} \{s_i\}$.

$$C_i(\langle s, 0 \rangle, \sigma) = \min$$

$$\begin{cases} C_{i-1}(\langle s, 0 \rangle, \sigma) \\ \max\{\langle s_i, 0 \rangle, C_{i-1}(\langle s, 0 \rangle, \sigma - \sigma_i) + n_i \\ \min_{\langle s', 0 \rangle > \langle s, j \rangle, \sigma' < \sigma} \{C_{i-1}(\langle s', 0 \rangle, \sigma') + \max\{0, n_i - (\langle s', 0 \rangle - \langle s_i, 0 \rangle) \\ + P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma - \sigma_i - \sigma')\}\} \end{cases} \quad (12)$$

$$P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'') = \min$$

$$\begin{cases} P_{i-1}(\langle s^+, 0 \rangle, \langle s', 0 \rangle, \sigma'') \\ \min_{0 < \sigma' \leq \sigma''} \{\max\{0, C_{i-1}(\langle s, 0 \rangle, \sigma') - \langle s_i, 0 \rangle + P_{i-1}(\langle s'', 0 \rangle, \langle s', 0 \rangle, \sigma'' - \sigma')\} \end{cases} \quad (13)$$

The dynamic programming recursion formula is shown in Eqs. (12) and (13). We now introduce the recursion formula in details. The definition of $C_i(\langle s, 0 \rangle, \sigma)$ whose value is a 2-tuple represent the smallest ending place $\langle t, x \rangle$ where a feasible set $J \subseteq \{J_1, J_2, \dots, J_i\}$ can complete, while satisfying $u(J) \geq \sigma$, $s(J) \geq s$ and j machines are used in s .

If $J_i \notin J$, i.e., J_i cannot be contained in a feasible set J satisfying the constraint, $C_i(\langle s, 0 \rangle, \sigma) = C_{i-1}(\langle s, 0 \rangle, \sigma)$.

Let us consider the situation $J_i \in J$, where J_i is contained in a feasible set J satisfying the constraint. There are two cases as follows.

Case 1: Job J_i is apparent to start after $\langle s_i, 0 \rangle$. If $c(J - \{J_i\}) \leq \langle s_i, 0 \rangle$, or else $c(J - \{J_i\}) > \langle s_i, 0 \rangle$ and $J - \{J_i\}$ are scheduled continuously from $\langle s_i, 0 \rangle$ to $c(J - \{J_i\})$, then $C_i(\langle s, 0 \rangle, \sigma) = \max\{\langle s_i, 0 \rangle, C_{i-1}(\langle s, 0 \rangle, \sigma - \sigma_i) + n_i$.

Case 2: Job J_i is apparent to start after $\langle s_i, 0 \rangle$. But the tasks scheduled between $\langle s_i, 0 \rangle$ and $c(J - \{J_i\})$ are not continuous. That is, some tasks of job J_i can be scattered between $\langle s_i, 0 \rangle$ and $c(J - \{J_i\})$ rather than after $c(J - \{J_i\})$. As stated in [16], the EDF method schedules the tasks as the form of periods of continuous processing. A period of continuous processing is called a block. We consider the scheduling of $J - \{J_i\}$ in the DP algorithm. Let the starting time of the last block in $J - \{J_i\}$ be $\langle s', 0 \rangle$, and the utility of the last block in $J - \{J_i\}$ be σ' , then the ending time of the last block is $C_{i-1}(\langle s', 0 \rangle, \sigma')$. Let $P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'')$ be the minimum amount of processing done between $\langle s_i, 0 \rangle$ and $\langle s', 0 \rangle$, with respect to feasible set $J'' \subseteq \{J_1, J_2, \dots, J_{i-1}\}$ with $s(J'') \geq s$, $c(J'') \leq s'$, and $\sigma(J'') \geq \sigma''$, then the number of slots available for job J_i between $\langle s_i, 0 \rangle$ and $\langle s', 0 \rangle$ is

$$\langle s', 0 \rangle - \langle s_i, 0 \rangle - P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma - \sigma_i - \sigma').$$

Then, the completing time $C_i(\langle s, 0 \rangle, \sigma)$ can be represented as

$$C_{i-1}(\langle s', 0 \rangle, \sigma') + \max\{0, n_i - \langle s', 0 \rangle + \langle s_i, 0 \rangle + P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma - \sigma_i - \sigma')\}$$

Enumerating every s' and σ' , we can get

$$C_i(\langle s, 0 \rangle, \sigma) = \min_{\langle s', 0 \rangle > \langle s, 0 \rangle, \sigma' < \sigma} \{C_{i-1}(\langle s', 0 \rangle, \sigma') + \max\{0, n_i - \langle s', 0 \rangle + \langle s_i, 0 \rangle + P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma - \sigma_i - \sigma')\}\}.$$

We now introduce how to realize the computation of $P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'')$. Recall the definition of $P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'')$. It is the minimum number of tasks scheduled between $\langle s_i, 0 \rangle$ and $\langle s', 0 \rangle$ satisfying the utility constraint. Assume $P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'')$ is achieved by a non-empty set $J'' \subseteq \{J_1, J_2, \dots, J_{i-1}\}$.

If $\langle s(J''), 0 \rangle > \langle s, j \rangle$, then $P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'') = P_{i-1}(\langle s(J''), 0 \rangle, \langle s', 0 \rangle, \sigma'')$. Therefore, we can enumerate every $s^+ > s$ and find the result. The result is the minimum one among $\min_{s^+ > s} \{P_{i-1}(\langle s^+, 0 \rangle, \langle s', 0 \rangle, \sigma'')\}$.

Otherwise, $\langle s(J''), 0 \rangle = \langle s, 0 \rangle$. Let the first block in the solution be J' and the total extra utility of the first block be σ' , then the ending time of the first block is $C_{i-1}(\langle s, 0 \rangle, \sigma')$. Therefore, $P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'')$ can

be represented as $\max\{0, C_{i-1}(\langle s, 0 \rangle, \sigma') - \langle s_i, 0 \rangle\} + P_{i-1}(\langle s'', 0 \rangle, \langle s', 0 \rangle, \sigma'' - \sigma')$, where s'' is the smallest starting time $\langle s'', 0 \rangle \geq C_{i-1}(\langle s, 0 \rangle, \sigma')$. Enumerating every σ' , we can obtain $P_{i-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'') = \min_{0 < \sigma' \leq \sigma''} \{\max\{0, C_{i-1}(\langle s, 0 \rangle, \sigma') - \langle s_i, 0 \rangle\} + P_{i-1}(\langle s'', 0 \rangle, \langle s', 0 \rangle, \sigma'' - \sigma')\}$.

The initial conditions are shown in Eqs. (14) to (17).

$$C_0(\langle s, 0 \rangle, 0) = \left\lceil \frac{s+1}{m} \right\rceil, s \bmod m, \text{ for all starting time } s \quad (14)$$

$$C_0(\langle s, 0 \rangle, w) = +\infty, \text{ for all starting time } s \text{ and } w > 0 \quad (15)$$

$$P_{j-1}(\langle s, 0 \rangle, \langle s', 0 \rangle, 0) = 0, \text{ for } j = 1, 2, \dots, n \quad (16)$$

$$P_0(\langle s, 0 \rangle, \langle s', 0 \rangle, \sigma'') = +\infty, \text{ for } \sigma'' > 0 \quad (17)$$

An improvement of the DP formula for the single-machine problem can also be used in the multi-machine problem. It cannot improve the result in a worst case where the tasks have no utility, but it can improve the result for many inputs.

5.3 Approximation algorithm for multi-machine problem

The same as single-machine problem, the main idea of the approximation algorithm for the multi-machine problem is to get two intermediate results using the EDF-multi-algorithms and the DP algorithm for the multi-machine problem independently and then combine the two results of the two algorithms. First, it uses the EDF-multi-algorithm to solve the problem without extra job utility. Second, it uses the DP algorithm to solve the problem with only extra job utility. Finally, it selects a larger one from the two scheduling results. Algorithm APPXm shows the detailed approximation algorithm.

Theorem 4 *The APPXm algorithm is a 2-approximation algorithm.*

Proof Let OPT be the utility obtained by an optimal solution and ALG be the utility obtained by the APPXm algorithm. It is easy to find that OPT can be represented as $\text{OPT} = u + \sigma$, where u is the total utility of the scheduled tasks and σ is the total extra utility of the entirely scheduled jobs in an optimal solution. Let u' be the total utility derived by the EDF-multi-algorithm and σ' be the total utility obtained by the dynamic programming algorithm. From the earlier analysis, both the EDF-multi-algorithm for the problem without extra utility and the dynamic programming algorithm for the problem with only an extra utility are optimal. Thus, $u \leq u'$ and $\sigma \leq \sigma'$, then we have $\text{OPT} \leq u' + \sigma'$. ALG actually can be represented as

Algorithm 5 APPXm algorithm

Input: $m, n, [s_i, e_i], \sigma_i, n_i, u_{ij}, 1 \leq i \leq n, 1 \leq j \leq n_i$

Output: Scheduling the tasks to m machines to maximize utility gain.

- 1: Use the EDF-Multi algorithm to solve the problem without extra utility;
- 2: Use the dynamic programming algorithm to schedule jobs with only the extra utility to m machines;
- 3: Use the EDF-Multi algorithm to solve the problem without extra utility;
- 4: Use the DP algorithm to solve the problem with only extra utility;
- 5: Let \mathcal{T} be the set of tasks scheduled by the EDF-Multi algorithm;
- 6: Let \mathcal{J} be the set of jobs scheduled by the DP algorithm;
- 7: Let $u(\mathcal{T})$ be the total utility of \mathcal{T} including the utility of the tasks in \mathcal{T} and the extra utility of the completed jobs among \mathcal{T} ;
- 8: Let $u(\mathcal{J})$ be the total utility of \mathcal{J} including the extra utility of the jobs in \mathcal{J} and the utility of all the tasks of \mathcal{J} ;
- 9: **if** $u(\mathcal{T}) < u(\mathcal{J})$ **then**
- 10: Use the scheduling of the DP algorithm;
- 11: **else**
- 12: Use the scheduling of the EDF-Multi algorithm;

$\text{ALG} \geq \max\{u', \sigma'\}$. Therefore, $\text{OPT} \leq 2\text{ALG}$. It completes the proof. \square

6 Simulation result

The simulation result is shown in this section. As the computation complexity of our algorithm is high, especially the DP algorithm; thus, the input of our simulation is set to not too large. The number of machines used in our simulation is at most 5, the number of applications is at most 100, and the number of tasks for each application is at most 5. The utility of each task is a random value. The starting time and the ending time and the extra utility of each application is also randomly generated.

As the optimal result is hard to compute; thus, we use an upper bound result to represent the optimal result. The upper bound is computed by dividing the extra utility of each application into its tasks in proportion, that is task with high utility will be assigned with a high extra utility.

The scheduling result in single machine is shown in Fig. 8. As shown in Fig. 8, the utility that the approximation algorithm get increases as the number of applications increases. However, when the number of application increases to a certain degree, the utility that the approximation algorithm can get increases slower; this is because

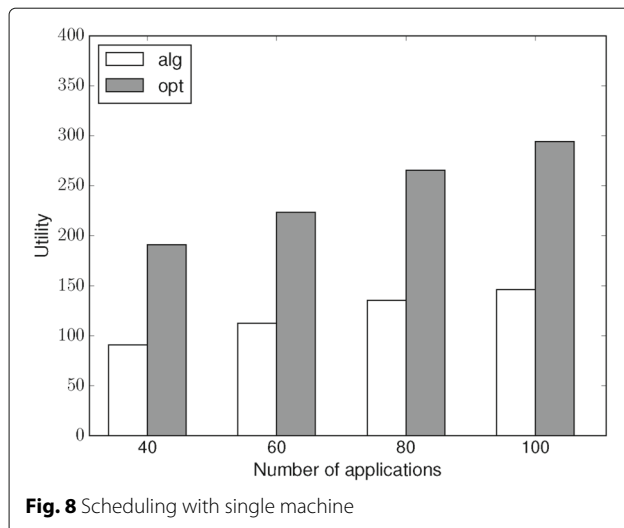


Fig. 8 Scheduling with single machine

the single machine is approaching to its maximum computing load. Figure 9 illustrates the scheduling result in five machines. Similar as the single-machine case, the utility that the approximation algorithm can get increase as the number of applications increases. And the increasing rate gets slower as the multiple machines are getting to their maximum load. Because the upper bound we used to represent the optimal result is higher than the real optimal result, therefore, the utility that the approximation algorithm gets can get closer to the optimal result and the difference is much less than two times, which confirms the approximation ratio; thus, the performance is acceptable in our simulation.

7 Conclusions

This paper proposes a class of algorithms to solve the problem of maximizing utility for job scheduling where each job consists of multiple tasks. Different from the

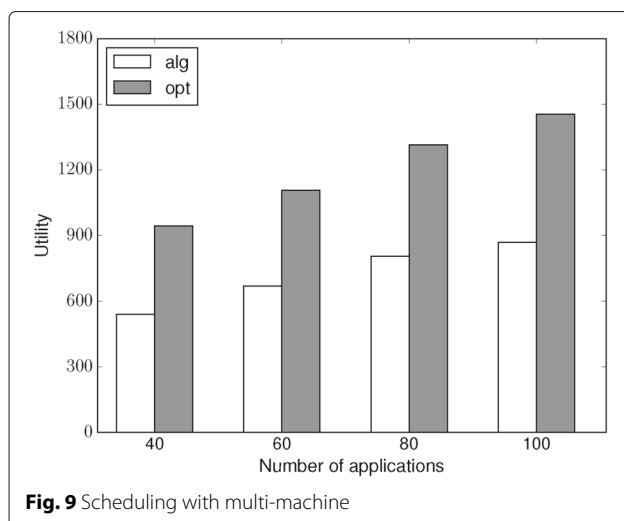


Fig. 9 Scheduling with multi-machine

existing works which either consider job utility or task utility individually, this paper considers both job utility and task utility simultaneously by introducing extra utility for every job. We analyze the complexity of the problem and discuss two sub-problems of scheduling jobs in a single machine and scheduling jobs in multiple machines. We design two 2-approximation algorithms for the sub-problems, and the approximation proofs are also presented. Although the time complexity is pseudo polynomial, we provide a theoretical insight into this problem.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under grant nos. 61502099, 61632008, 61320106007, 61502100, 61532013, 61602084, and 61672154, Jiangsu Provincial Natural Science Foundation of China under grant no. BK20150637, Jiangsu Provincial Key Laboratory of Network and Information Security under grant no. BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under grant no. 93K-9, and Collaborative Innovation Center of Novel Software Technology and Industrialization.

Authors' contributions

XF and WW conceived and designed the study. XF performed the experiments. XF and YL wrote the paper. JL, HG, and YL reviewed and edited the manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

¹School of Computer Science and Engineering, Southeast University, Nanjing, China. ²School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. ³Department of Computer Science, Georgia State University, Atlanta, USA.

Received: 10 August 2017 Accepted: 13 November 2017

Published online: 25 November 2017

References

- SH Bokhari, *Assignment Problems in Parallel and Distributed Computing*, vol. 32. (Springer US, New York, 2012)
- M Armbrust, A Fox, R Griffith, AD Joseph, R Katz, A Konwinski, et al., A view of cloud computing. *Commun. ACM.* **53**(4), 50–58 (2010). Available from: URL <http://doi.acm.org/10.1145/1721654.1721672>
- Q Zhang, L Cheng, R Boutaba, Cloud computing: state-of-the-art and research challenges. *J. Int. Serv. Appl.* **1**(1), 7–18 (2010). Available from: <http://dx.doi.org/10.1007/s13174-010-0007-6>
- V Sharma, U Mukherji, V Joseph, S. Gupta, Optimal energy management policies for energy harvesting sensor nodes. *IEEE Trans. Wirel. Commun.* **9**(4), 1326–1336 (2010)
- C Lefurgy, K Rajamani, F Rawson, W Felten, M Kistler, TW Keller, Energy management for commercial servers. *Computer.* **36**(12), 39–48 (2003)
- RL Graham, EL Lawler, JK Lenstra, AHGR Kan, in *Discrete Optimization III Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver. vol. 5 of Annals of Discrete Mathematics*, ed. by ELJ P L Hammer, BH Korte. Optimization and approximation in deterministic sequencing and scheduling: a survey (Elsevier, 1979), pp. 287–326. Available from: <http://www.sciencedirect.com/science/article/pii/S016750600870356X>. Accessed 29 Apr 2008
- D Applegate, W Cook, A computational study of the job-shop scheduling problem. *ORSA J. Comput.* **3**(2), 149–156

8. EL Lawler, JK Lenstra, AHGR Kan, DB Shmoys, in *Logistics of Production and Inventory*, vol. 4 of *Handbooks in Operations Research and Management Science*. Chapter 9. Sequencing and scheduling: algorithms and complexity (Elsevier, 1993), pp. 445–522. Available from: <http://www.sciencedirect.com/science/article/pii/S0927050705801896>
9. J Blazewicz, JK Lenstra, AHGR Kan, Scheduling subject to resource constraints: classification and complexity. *Discret. Appl. Math.* **5**(1), 11–24 (1983). Available from: <http://www.sciencedirect.com/science/article/pii/0166218X83900124>. Accessed 9 Sept 2002
10. Y Zheng, B Ji, N Shroff, P Sinha, in *2015 IEEE 8th International Conference on Cloud Computing*. Forget the deadline: scheduling interactive applications in data centers (IEEE, New York, 2015), pp. 293–300
11. MR Garey, DS Johnson, Two-processor scheduling with start-times and deadlines. *SIAM J. Comput.* **6**(3), 416–426 (1977)
12. MR Garey, DS Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. (W.H. Freeman & Co, New York, 1979)
13. A Bar-Noy, S Guha, JS Naor, B Schieber, in *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing. STOC '99*. Approximating the throughput of multiple machines under real-time scheduling (ACM, New York, 1999), pp. 622–631. Available from: <http://doi.acm.org/10.1145/301250.301420>
14. A Bar-Noy, R Bar-Yehuda, A Freund, J (Seffi) Naor, B Schieber, A unified approach to approximating resource allocation and scheduling. *J. ACM.* **48**(5), 1069–1090 (2001). Available from: <http://doi.acm.org/10.1145/502102.502107>
15. G Calinescu, A Chakrabarti, H Karloff, Y Rabani, An improved approximation algorithm for resource allocation. *ACM Trans. Algorithms.* **7**(4), 48:1–48:7 (2011). Available from: <http://doi.acm.org/10.1145/2000807.2000816>
16. EL Lawler, A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Ann. Oper. Res.* **26**(1-4), 125–133 (1991). Available from: <http://dx.doi.org/10.1007/BF02248588>
17. G Steiner, Models and algorithms for planning and scheduling problems minimizing the number of tardy jobs with precedence constraints and agreeable due dates. *Discret. Appl. Math.* **72**(1), 167–177 (1997). Available from: <http://www.sciencedirect.com/science/article/pii/S0166218X96000431>. Accessed 10 Jan
18. H Kise, T Ibaraki, H Mine, A solvable case of the one-machine scheduling problem with ready and due times. *Oper. Res.* **26**(1), 121–126 (1978). Available from: <http://dx.doi.org/10.1287/opre.26.1.121>
19. V Bafna, B Narayanan, R6 Ravi, *Nonoverlapping Local Alignments, Weighted Independent Sets of Axis Parallel Rectangles*. (Center for Discrete Mathematics & Theoretical Computer Science, Princeton, 1995)
20. P Berman, B DasGupta, S Muthukrishnan, in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '02*. Simple approximation algorithm for nonoverlapping local alignments (Society for Industrial and Applied Mathematics, Philadelphia, 2002), pp. 677–678. Available from: <http://dl.acm.org/citation.cfm?id=545381.545471>
21. R Bar-Yehuda, MM Halldórsson, JS Naor, H Shachnai, I Shapira, in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '02*. Scheduling split intervals (Society for Industrial and Applied Mathematics, Philadelphia, 2002), pp. 732–741. Available from: <http://dl.acm.org/citation.cfm?id=545381.545479>
22. R Bar-Yehuda, D Rawitz, Using fractional primal-dual to schedule split intervals with demands. *Discret. Optim.* **3**(4), 275–287 (2006). Available from: <http://dx.doi.org/10.1016/j.disopt.2006.05.010>
23. VT Chakaravarthy, A Roy Choudhury, S Roy, Y Sabharwal, in *Proceedings of the 19th International Conference on Parallel Processing. Euro-Par'13*. Scheduling jobs with multiple non-uniform tasks (Springer-Verlag, Berlin, Heidelberg, 2013), pp. 90–101. Available from: http://dx.doi.org/10.1007/978-3-642-40047-6_12

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com
