Optimal Schedule for Periodic Jobs with Discretely Controllable Processing Times on Two Machines

Zizhao Wang[a,*], Wei Bao[a], Dong Yuan[b], Albert Y. Zomaya[a]

[a] *The University of Sydney, Building J12, 1 Cleveland Street, Darlington, Sydney, 2008, NSW, Australia*
[b] *The University of Sydney, J03 Electrical Engineering Building, Maze Cres, Darlington, Sydney, 2008, NSW, Australia*

Abstract

In many real-world situations, the processing time of computational jobs can be shortened by lowering the processing quality. This is referred to as discretely controllable processing time, where the original processing time can be shortened to a number of levels with lower processing qualities. In this paper, we study the scheduling problem of periodic jobs with discretely controllable processing times on two machines. The problem is NP-hard, as directly solving it through dynamic programming leads to exponential computational complexity. This is because we need to memorise a set of processed jobs to avoid reprocessing. In order to address this issue, we prove the Ordered Scheduling Structure (OSS) Property and the Consecutive Decision Making (CDM) Property. The OSS Property allows us to search for an optimal solution in which jobs on the same machine are orderly started. The CDM Property allows us to memorise only two jobs to completely avoid the job reprocessing. These two properties greatly decrease the searching space, and the resultant dynamic programming solution to find an optimal solution is with pseudo-polynomial computational complexity.

Keywords: Scheduling; periodic job; controllable processing time

*Corresponding author
Email addresses: zwan5430@uni.sydney.edu.au (Zizhao Wang), wei.bao@sydney.edu.au (Wei Bao), dong.yuan@sydney.edu.au (Dong Yuan), albert.zomaya@sydney.edu.au (Albert Y. Zomaya)

1. Introduction

With the advances of smart devices and Internet of Things (IoT), computational jobs can directly be run in the devices, facilitating a wide range of smart applications. However, due to the limited computational capacity and energy, the devices may not be able to complete computational-intensive jobs. There are two evident solutions: (1) We reduce the processing time/workload by sacrificing processing quality (i.e., controllable processing time). (2) We offload the jobs to a nearby resource-rich computing node.

In this paper, we study the scheduling problem in two machines: one low-speed machine and one high-speed machine. The jobs arrive periodically and each can be processed in either machine. Each job has multiple levels of controllable processing times: If a job is fully completed, it will obtain a full utility; If a job is completed with a lower processing level, a partial utility will be obtained. Also, each job has a due time. If the completion time of a job breaches its own due time, it will obtain 0 utility. We also assume that each job can be processed by one machine and is non-preemptive. Our objective is to maximise the overall accumulated utility. The problem is referred to as the Periodic jobs with discretely Controllable Processing Times on two parallel machines (PCPT) Problem.

There are many real-world examples of the PCPT Problem:

Example 1. Real-time video processing through deep neural networks with early exits. A real-time video is generated at a smart device (e.g., VR glasses), and each video frame is processed by a deep neural network (DNN) for inference (e,g., video analytics). Each frame can be processed at the local device or offloaded to a nearby computer for DNN inference. The DNN model has multiple exit points. Earlier exit points lead to smaller inference delay but lower processing accuracy [1, 2, 3, 4, 5].

Example 2. Scalable video transcoding for live streaming. A live video is generated from a smart phone, and each video chuck is transcoded to be broadcasted

[6][1]. The video transcoding jobs can be processed at the phone or offloaded to a nearby server. Scalable video transcoding is able to encode the original video into multiple layers. While the base layer is mandatory, enhancement layers can be skipped at the cost of video quality [7, 8, 9, 10, 11].

Example 3. Smart building controller. A building management computer periodically collects sensed data in a building to control mechanical and electrical equipment to save energy costs and increase living comfort. However, optimal control is computing intensive as there are a huge amount of data collected. The computer can offload the computing job to its associated server. The computer and the server may only process a subset of collected data, which will lead to a sub-optimal decision with a smaller delay [12, 13, 14].

Solving PCPT Problem is challenging. A general problem to schedule jobs in multiple parallel machines to minimise the number of tardy jobs is NP-Hard (even without controllable processing times). We proved the NP-hardness of the PCPT Problem later in this work. The PCPT Problem has a special form because (1) jobs arrive periodically and (2) there are two parallel machines. By characterising these two new settings, we have opportunities to reduce the searching space to solve the PCPT Problem in pseudo-polynomial computational complexity.

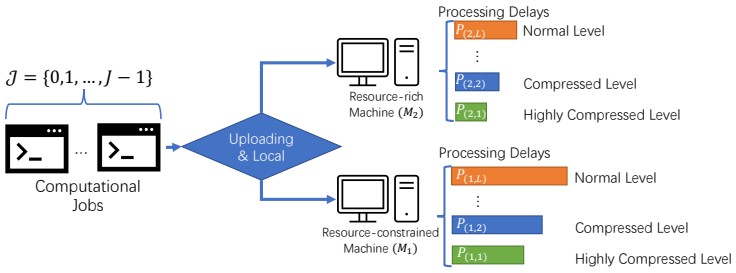


Figure 1: Overview of System Setting.

In this paper, we propose PCPT-solver to solve the PCPT Problem through

[1]In a video live streaming system, the video is segmented into chunks of equal size for transcoding. The transcoded chunks are generated, to be transmitted to devices of viewers.

dynamic programming in pseudo-polynomial computational complexity. Direct utilising dynamic programming leads to a large state space and thus exponential computational complexity. This is because we need to memorise which jobs have been done so that we do not reprocess the jobs again. We need to memorise a number of jobs and the state space is exponential with respect to the number of jobs in the memory. In order to address this issue, we first characterise the Ordered Scheduling Structure (OSS) Property. It allows us to search for an optimal solution only from the situations that jobs are started in order on the same machine. OSS helps reduce the memory size, but there are still $\frac{T_{\max}}{T}$ jobs in the memory, where $T_{\max}$ is the relative deadline of the jobs and T is the inter-arrival time. The complexity is still in the order of $2^{\frac{T_{\max}}{T}}$. In the next step, we prove the Consecutive Decision Making (CDM) Property, so that it is sufficient to memorise only 2 jobs. As a result, the state space is substantially reduced and the resultant computational complexity of the dynamic programming solver is pseudo-polynomial.

The rest of the paper is organised as follows: The related work is summarised in Section 2. The problem formulation is outlined in Section 3. A dynamic programming based solution is presented in Section 4, and it is further simplified in Section 5 for the purpose of pseudo-polynomial complexity. The optimisation algorithm PCPT-solver and its complexity analysis are presented in Section 6. In Section 7, we conduct trace-driven simulation to demonstrate the effectiveness of the PCPT-solver. Finally, we draw the conclusion in Section 8.

2. Related Work

In many real-world situations, the processing time of computational jobs can be shortened by sacrificing an acceptable level of processing quality. This concept was introduced by [15, 16] and it is called as the controllable processing time. Since then, scheduling problems with controllable processing time has received more attention [17, 18, 19]. A number of works focus on the single machine scenario. The works in [20, 21, 22] study the problems of minimising the

sum of cost incurred in compressing job processing times and the cost associated with the number of tardy jobs. The studies [23, 24, 25] consider the same problem under the discrete controllable processing times, i.e. the possible processing times of a job can only be controlled to be a number of specified lengths. [1] describes a DNN inference application with early exits as an example of discretely controllable processing times, where jobs can be exited earlier to shorten the inference delay with lower accuracy. The aforementioned works focus on the single machine, which is substantially different from our two-machine scenario. The works [26, 27, 28, 29] focused on controllable processing times among multiple parallel machines. However, they aim to minimise makespan (the completion time of the last job) or the total tardiness (the sum earliness/lateness of jobs). [30] considers the objective of total weighted completion time on parallel machines and proposes a $\frac{3}{2}$-approximation algorithm. Those are different from our system setting (we have 0-1 utility if a job misses/meets its deadline). To the best of our knowledge, this is the first work that optimally solves the periodic jobs with discretely controllable processing times on two parallel machines in pseudo-polynomial complexity.

Without considering the controllable processing time, our problem can be reduced to a traditional scheduling problem with the objective of minimising the number of tardy jobs [31]. This type of problems have been extensively studied in the literature [32, 33, 34, 35, 36]. The problem of minimising the number of tardy jobs on a single machine ($1||\Sigma U_j$) was solved by [37] with an $\mathcal{O}(n \log n)$ complexity. Another category of works consider to schedule jobs under the parallel machine environment [38, 39]. Generally, minimizing the number of tardy jobs under such environments is known as NP-Hard [40].

3. Problem Formulation

3.1. System Setting

As shown in Fig. 1, the system operates in discrete time slots, starting from 0 and ending at T_c. We consider a job set $\mathcal{J}$ containing J jobs, denoted by

$\mathcal{J} = \{0, 1, 2, \ldots, J-1\}$. All jobs in $\mathcal{J}$ are independent, identical, and non-preemptive. Jobs arrive at the system with a fixed time interval T. The release time (i.e., arrival time) of each job $j \in \mathcal{J}$ is $r_j = j \cdot T$; T_{max} is the relative deadline for all jobs and $d_j = r_j + T_{max}$ is the due time of job j. T_c is the common deadline for all jobs, indicating that the system will be shut down at T_c and all jobs need to be completed on or before T_c. We let $(J-1)T \leq T_c < JT$, indicating that the system is shut down after all jobs arrived. Any job j can only be started on or after its release time r_j, and it should be completed on or before its due time and the common deadline $\min(d_j, T_c)$. We define $\min(d_j, T_c)$ as the absolute deadline of job j. Zero utility will be received if job j breaches its absolute deadline.

We employ one low-speed machine M_1 and one high-speed machine M_2 in our system. The released job can be allocated to either machine M_1 or M_2 to process. Please note, each job can only be allocated once, or rejected by both machines.

We allow discretely controllable processing times, indicating that machines M_1 and M_2 have multiple processing levels. A compressed processing level will shorten the processing time and lead to a partial utility. There are L processing levels for both M_1 and M_2, denoted as $\mathcal{L} = \{1, \ldots, L\}$, and level L indicates the standard processing level with no compression.

All jobs are identical, and the processing time of one compressed processing level for any job on the same machine is identical. Let $\mathcal{P}_1 = \{p_{(1,1)}, p_{(1,2)}, \ldots, p_{(1,L)}\}$ denote the controllable processing time set for machine M_1, and let $\mathcal{P}_2 = \{p_{(2,1)}, p_{(2,2)}, \ldots, p_{(2,L)}\}$ denote controllable processing time set for machine M_2. In other words, the processing time of a job with processing level l is $p_{(1,l)}$ (resp. $p_{(2,l)}$) on M_1 (resp. M_2). We have $p_{(1,l)} < p_{(1,l')}$ and $p_{(2,l)} < p_{(2,l')}$, for all $l < l' \leq L$, as the processing time for a lower processing level is shorter. Please note, the aforementioned parameters $T, T_{max}, T_c, p_{(1,l)}$, and $p_{(2,l)}$ are integers.

The system will obtain the same utility if any job is completed with the same processing level on either machine on time. The utility set is denoted as $\mathcal{U} = \{u_1, u_2, \ldots, u_L\}$. u_l utility is received if a job is processed by level l. Please

note, we have $u_l < u_{l'}$ for all $l < l', l, l' \in \mathcal{L}$.

Let $O_{(j,m,l)}, j \in \mathcal{J}, m \in \{1,2\}, l \in \mathcal{L}$ denote an *active operation*, which indicates that the machine M_m starts the job j with the processing level of l. Such an operation will obtain a utility of u_l. We also allow the machine M_m not to start any job but to wait for one time unit during time $[t, t+1]$, where $t \in [0, T_c)$. We call such operation the *waiting operation*, denoted as $W_{(m,t)}$. The waiting operation will gain 0 utility. Let $o = O_{(j,m,l)}$ or $o = W_{(m,t)}$ denote an arbitrary operation.

Our objective is to find an optimal sequence that maximises the overall utility (which will be formally defined in the next section).

3.2. Decision Variables

Let the 0-1 decision variable $x_{jml} = 1$ represent an active operation $O_{(j,m,l)}$ is performed, otherwise $x_{jml} = 0$. All jobs are non-preemptive, and each job can be started at most once or never be started. We have $\sum_{m\in\{1,2\}, l\in\mathcal{L}} x_{jml} \leq 1, \forall j \in \mathcal{J}$.

Let $b_{jm} (j \in \mathcal{J}, m \in \{1,2\})$ denote the start time of the operation $O_{(j,m,l)}$ on machine M_m. If a job j is never started by machine M_m, we let $b_{jm} = +\infty$. All jobs can only be started on or after their release time, and then we have $b_{jm} \geq r_j, \forall j \in \mathcal{J}, m \in \{1,2\}$. We have only one start time $b_j = \min\{b_{j1}, b_{j2}\}$ for job j.

Let c_{jm} denote the completion time of job j on machine M_m. If job j is started by machine M_m with the operation $O_{(j,m,l)}$, then the completion time of j on machine M_m is $c_{jm} = b_{jm} + p_{(m,l)}$. If job j is not started by machine M_m, we let $c_{jm} = +\infty$. Then, the completion time of j is $c_j = \min\{c_{j1}, c_{j2}\}$.

Let 0-1 decision variable $y_{mt} = 1$ indicate the wait operation $W_{(m,t)}$ is performed, otherwise $y_{mt} = 0$.

A mixed integer formulation can be derived by introducing the decision variables x_{jml}, y_{mt}, and b_{jm} for each operation, indicating whether and when the operation $O_{(j,m,l)}$ or $W_{(m,t)}$ should be performed.

3.3. Optimisation Problem Formulation

We formulate this optimisation problem, which will be referred to as **PCPT** Problem, as follows

$$\max_{\substack{x_{jml},y_{mt},b_{jm},\\ \forall j\in\mathcal{J},m\in\{1,2\},\\ l\in\mathcal{L},t\in[0,T_c)}} \sum_{\substack{\forall j\in\mathcal{J},m\in\{1,2\},\\ l\in\mathcal{L}}} u_l x_{jml}, \tag{1a}$$

$$\text{s. t. } x_{jml}, y_{mt} \in \{0,1\}, \forall j \in \mathcal{J}, m \in \{1,2\}, l \in \mathcal{L}, t \in [0,T_c); \tag{1b}$$

$$\sum_{m\in\{1,2\},l\in\mathcal{L}} x_{jml} \leq 1, \forall j \in \mathcal{J}; \tag{1c}$$

$$b_{jm} \geq r_j, \forall j \in \mathcal{J}, m \in \{1,2\}; \tag{1d}$$

$$b_{jm} + p_{(m,l)} \leq \min(d_j, T_c), \forall j \in \mathcal{J}, m \in \{1,2\}, l \in \mathcal{L}, x_{jml} = 1; \tag{1e}$$

$$b_{jm} + p_{(m,l)} \leq b_{km} \text{ OR } b_{km} + p_{(m,l')} \leq b_{jm},$$
$$\forall k, j \in \mathcal{J}, k \neq j, m \in \{1,2\}, l, l' \in \mathcal{L}, x_{jml} = x_{kml'} = 1; \tag{1f}$$

$$b_{jm} + p_{(m,l)} \leq t \text{ OR } t+1 \leq b_{jm},$$
$$\forall j \in \mathcal{J}, m \in \{1,2\}, l \in \mathcal{L}, t \in [0,T_c), x_{jml} = y_{mt} = 1. \tag{1g}$$

(1a) is the overall utility to be maximised. (1b) and (1c) indicate that each job can be started by one operation at most once. (1d) indicates job j can only be started on or after its release time. (1e) indicates that if job j is started, it should be completed on or before its absolute deadline $\min(d_j, T_c)$. (1f) indicates that the processing duration of any two jobs cannot be overlapped on the same machine. (1g) indicates that an active operation and a waiting operation cannot be performed at the same time on one machine.

Note that we can also employ the three-field notation $\alpha|\beta|\gamma$ [32] to describe PCPT Problem. We have Q_2 in the α field, indicating two machines are parallel with different speeds. In the β field, we have $r_j = j \cdot T$ and $d_j = r_j + T_{max}$, which indicate the release times and the due times of each jobs. All operations must be completed on or before T_c, then we have $\overline{d} = T_c$ where $\overline{d}$ is the common deadline of all the jobs. We allow the controllable processing times, so that *contr* is also in the β field. In the γ field, our objective is to minimise the negative overall

utility $-\sum_{j\in\mathcal{J},m\in\{1,2\},l\in\mathcal{L}} u_l x_{jml}$. Thus, the PCPT Problem can be presented as $Q_2|r_j = j\cdot T, d_j = r_j + T_{max}, \overline{d} = T_c, contr| - \sum_{j\in\mathcal{J},m\in\{1,2\},l\in\mathcal{L}} u_l x_{jml}$.

3.4. The NP-harness of the PCPT Problem

Theorem 1. *The PCPT Problem is an NP-Hard problem.*

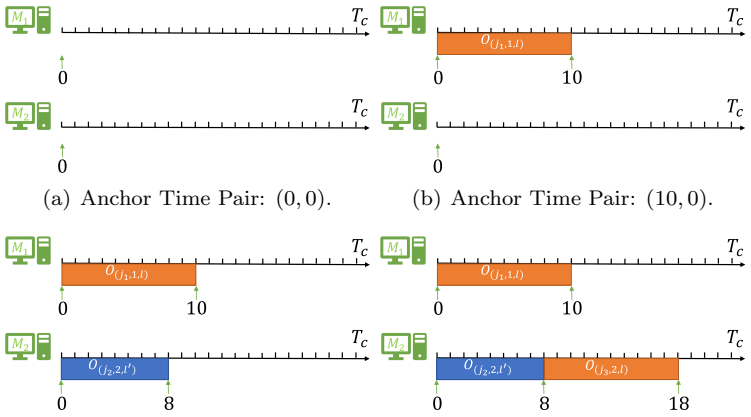
Detailed proof of this theorem is provided in Appendix A. It is very challenging to solve such a problem using standard methods directly. In what follows, we introduce the OSS Property and the CDM Property to reduce the state space of the PCPT Problem. Based on these two properties, we are able to obtain an optimal schedule through a dynamic programming approach at a pseudo-polynomial time complexity.

3.5. Sequence of Operations

Let $S_1 = (s_{(1,1)}, \ldots, s_{(1,N_1)})$ denote the sequence of operations of machine M_1, which records the N_1 operations (in order) completed by machine M_1 throughout the whole system period, as well as the start time of these operations. Let $S_2 = (s_{(2,1)}, \ldots, s_{(2,N_2)})$ represent the sequence of operations of machine M_2.

We define $s_{(1,n_1)}$ as the *operation entry*, which is a tuple representing the n_1th operation completed by machine M_1 and the start time of this operation. We have two types of operation entries. For an active operation entry, we define $s_{(1,n_1)} = (t_{(1,n_1)}, O_{(j_{(1,n_1)},1,l_{(1,n_1)})}), j_{(1,n_1)} \in \mathcal{J}, l_{(1,n_1)} \in \mathcal{L}, n_1 \in [0, N_1]$, where $O_{(j_{(1,n_1)},1,l_{(1,n_1)})}$) represents the n_1th operation completed by machine M_1, and $t_{(1,n_1)}$ is the start time of this active operation. For a waiting operation entry, we define $s_{(1,n_1)} = (t_{(1,n_1)}, W_{(1,t_{(1,n_1)})}), n_1 \in [0, N_1]$, where $W_{(1,t_{(1,n_1)})}$ represents the n_1th operation completed by machine M_1, and $t_{(1,n_1)}$ is the start time of this waiting operation. We define the operation entry $s_{(2,n_2)}$ for machine M_2 in the same way.

Let $S = (S_1, S_2)$ denote the valid sequence of the PCPT Problem. A valid sequence represents a valid solution to the PCPT Problem. It includes all information to carry out the solution. A valid sequence S should never include



(a) Anchor Time Pair: $(0, 0)$. (b) Anchor Time Pair: $(10, 0)$.

(c) Anchor Time Pair: $(10, 8)$. (d) Anchor Time Pair: $(10, 18)$.

Figure 2: An example of anchor time pair.

an active operation which generates 0 utility (otherwise we just delete this active operation). We aim to find an optimal valid sequence in order to solve PCPT Problem.

4. Problem Solution through Dynamic Programming

In this section, we target to find an optimal valid sequence via dynamic programming. We first introduce how the system evolves. We define the decision maker and decision making time. Next, we investigate an important structure (Theorem 2) which allows us to search for an optimal solution only from ordered sequences. Then, we define the system state, action, state transition, and reward, and formulate the Bellman equation. However, even we utilised the structure in Theorem 2, the resultant state space is still large. We leave our further solutions to reduce the state space to Section 5. Please note that we do need to formulate the dynamic programming in the form in Section 4 in the first stage, as it is the prerequisite for the discussion on the possibility of space reduction in Section 5. The forms of dynamic programming are called original form and simplified form in Sections 4 and 5 respectively.

4.1. System Evolution by Anchor Time Pair

There are two machines in our system, and each of them can make a decision to start a new operation when it just completed an active or waiting operation. At any time instant $t \in [0, T_c)$, if one machine is ready to make a decision to start a new operation (i.e., action operation or waiting operation), then we define the time instant t as the *decision making time* for this machine and we define this machine as the *decision maker*.

On the contrary, at any time instant $t \in [0, T_c)$, if one machine is working on a previously started operation and will be ready to make the next decision later, then we define the completion time of the current operation as the *committed time* for this machine, and we define this busy machine as the *committed machine*. Then, given any time instant t, the two machines have the following 4 possibilities: (a) M_1: decision maker; M_2: committed machine, (b) M_1: decision maker; M_2: decision maker. (c) M_1: committed machine; M_2: decision maker. (d) M_1: committed machine; M_2: committed machine.

We adopt the *anchor time pair* (τ_1, τ_2) to label the situation when at least one machine is a decision maker. At t, for case (a), we have the anchor time pair (t, t_C), where $t_C > t$ is the committed time for machine M_2 and t is the decision making time for M_1. In case (b), the anchor time pair is (t, t), indicating that both machines are ready to make the decision at t. In this case, without loss of generality, we let M_1 make the decision first. In case (c), the anchor time pair is (t_C, t), where machine M_2 makes a decision at t and machine M_1 can make its next decision at t_C. We do not consider the case (d), as none of the machines makes a decision at time t and we can just skip this situation. Note that, given an anchor time pair (τ_1, τ_2), the decision making time is $\min(\tau_1, \tau_2)$; the decision maker is M_1 if $\tau_1 \leq \tau_2$, or M_2 if $\tau_1 > \tau_2$.

Anchor time pair is the reference of time for the system evolution. The system evolution can be demonstrated through an example. In Fig. 2(a), at the time instant 0, both M_1 and M_2 are ready to make decisions. The anchor time pair is $(0, 0)$ and M_1 will start a new operation. Next, as shown in Fig. 2(b), M_1 completes this operation at 10. Then, the new anchor time pair is $(10, 0)$

and M_2 is the decision maker. As shown in Fig. 2(c), M_2 starts an operation at time 0 and completes it at 8. Then, the new anchor time pair is $(10, 8)$, and M_2 is still the decision maker. In Fig. 2(d), M_2 completes an operation at 18 and the anchor time pair becomes $(10, 18)$. M_1 is the decision maker at this anchor time pair. Therefore, at one anchor time pair, we make a decision (for one machine), and then we move to the next anchor time pair to make the next decision. The anchor time pair is used to describe the system evaluation by flagging the decisive moments.

4.2. Ordered Scheduling Structure (OSS) Property

In order to characterise the system evolution, it is not sufficient to only flag the decision moments through the anchor time pairs. We also need to record which jobs have already been done so that we do not reprocess them. One straightforward way is to record all such jobs in a set, but this will obviously lead to large state space. Therefore, in what follows, we present the Ordered Scheduling Structure (OSS) Property. It allows us to search for an optimal solution only from the situations where jobs are started in order on the same machine. Therefore, we only need to record a smaller number of jobs.

Theorem 2. *There exists an optimal sequence $S^* = (S_1^*, S_2^*)$, satisfying the following conditions:*
(1) For any two operation entries $s_{(1,n_1)}, s_{(1,n_1')} \in S_1^*$ and $n_1 \neq n_1'$, if $j_{(1,n_1)} < j_{(1,n_1')}$, we have $n_1 < n_1'$.
(2) For any two operation entries $s_{(2,n_2)}, s_{(2,n_2')} \in S_2^*$ and $n_2 \neq n_2'$, if $j_{(2,n_2)} < j_{(2,n_2')}$, we have $n_2 < n_2'$.
In other words, if we have two jobs, and we process both of them on the same machine, then the earlier released job should be started earlier.

The detailed proof can be found in Appendix B. The OSS Property is a very important property to facilitate the search for an optimal sequence. It allows us to search for the optimal sequence only from the sequences with the jobs that are started in order on the same machine. When one machine is making a

decision, it can only consider the jobs that are released after the last completed job on that machine. Therefore, for one machine, if we know job j is already done at this machine, we can ignore all jobs before j for this machine and we do not need to record jobs before j for it.

4.3. System State (Original Form)

We now define the system state. We adopt the anchor time pair to show the decision time. We also employ two *job exclusion sets* to record the completed jobs, so that these jobs should not be considered anymore. Let $\Psi = (\tau_1, \tau_2, F_1, F_2)$ denote the system state (original form), where (τ_1, τ_2) is the anchor time pair, and F_1 and F_2 are the job exclusion sets which record the jobs that have been done by M_1 and M_2 that can no longer be done again. F_1 records jobs that have been done by M_1 and these jobs cannot be done by M_1 or M_2 later. F_2 records jobs that have been done by M_2 and these jobs cannot be done by M_1 or M_2 later. We use two sets F_1 and F_2 instead of one set because we need to know which machine has done these jobs to facilitate the updates of F_1 and F_2 as the system evolves (Section 4.5). We can record a fewer number of jobs (instead of recording all jobs that have been done) in F_1 and F_2 by considering the following factors: (1) Even if we immediately start to process the job with the smallest processing time but its absolute deadline is still missed, it is not necessary to record this job (or any earlier jobs), because the decision maker will automatically avoid processing a job missing the absolute deadline. (2) According to the OSS Property, if a job j_0 has been done by M_1 (or M_2), all the jobs before j_0 will no longer be considered by that machine.

Given any anchor time pair (τ_1, τ_2). Let $\mathcal{R}_1$ (resp. $\mathcal{R}_2$) be the historical set which records all the jobs that have been completed by machine M_1 (resp. M_2) on or before time τ_1 (resp. τ_2). Let $R_1 = \max \mathcal{R}_1$ (resp. $R_2 = \max \mathcal{R}_2$) denote the last job that is completed by M_1 (resp. M_2) on or before time τ_1 (resp. τ_2). F_1 does not need to include all entries in $\mathcal{R}_1$ and $\mathcal{R}_2$, but it still should record the following two types of jobs.

For F_1, first, we need to record the jobs that are completed by machine M_1

on or before τ_1, so that M_2 will no longer work on them. These jobs should satisfy three conditions: (1) The jobs should be completed by machine M_1 on or before τ_1; (2) The jobs should be able to be started by M_2 at time τ_2 with the smallest processing time $p_{(2,1)}$ to meet their absolute deadlines; (3) The jobs should be released after the last job that is completed by machine M_2 on or before τ_2. (3) is because machine M_2 follows the OSS Property so that if the job is before that last job, we can ignore it in M_2. Therefore, the above job set can be defined as $G_1 = \{j | j \in \mathcal{R}_1 \wedge j \geq \lceil \frac{\tau_2 + p_{(2,1)} - T_{max}}{T} \rceil \wedge j > R_2\}$.

Second, for F_1, we need to record the jobs that may affect the future decisions of machine M_1. There is at most one job to be considered here: Due to the OSS Property, machine M_1 should process all jobs in order. We only need to record the last completed job, i.e., R_1, for machine M_1. Please note that if R_1 does not meet its absolute deadline even if we process it with the smallest processing time, we can further ignore it as the decision maker will automatically avoid processing a job missing its absolute deadline. We use H_1 to denote the set of jobs influencing M_1, where $H_1 = \begin{cases} \{R_1\} & \text{if } R_1 \geq \lceil \frac{\tau_1 + p_{(1,1)} - T_{max}}{T} \rceil; \\ \emptyset & \text{otherwise.} \end{cases}$

In summary, the job exclusion set F_1 for machine M_1 is the union of G_1 and H_1

$$F_1 = G_1 \bigcup H_1. \tag{2}$$

We define the job exclusion set F_2 for machine M_2 in a symmetrical manner. $F_2 = G_2 \bigcup H_2$, and G_2 and H_2 are also defined symmetrically as G_1 and H_1.

The *initial system state* is $(0, 0, \varnothing, \varnothing)$. Any original system state with the anchor time pair of (T_c, T_c) is an original *stop state*.

4.4. Action (Original Form)

At any system state $\Psi = (\tau_1, \tau_2, F_1, F_2)$, and Ψ is not a stop state, the decision maker can take a valid operation to transit to a resultant system state. We define $\mathcal{A}(\Psi)$ as the *action set* of the system state at Ψ. Given $\Psi = (\tau_1, \tau_2, F_1, F_2)$, and $(\tau_1, \tau_2) \neq (T_c, T_c)$ (not a stop state), the action set

$\mathcal{A}(\Psi)$ lists all valid operations as follows.

① For any state, the waiting operation is a valid operation.

② A valid active operation can only start a job that is released on or before the decision making time.

③ A valid active operation, due to the OSS Property, can only start a job that is released after the last completed job by the decision maker. Suppose M_1 is the decision maker, it can only start a job j_0 such that $j_0 > \max F_1$.

④ A valid active operation can only start a job that is not completed by the committed machine on or before the committed time. Suppose M_1 is the decision maker, it can only start a job j_0 such that $j_0 \notin F_2$.

⑤ A valid active operation can start a job with a certain processing level so that the absolute deadline is not breached.

Therefore, given a system state Ψ, any operations that satisfy the conditions ①–⑤ will be included in the action set $\mathcal{A}(\Psi)$, the formal definition of $\mathcal{A}(\Psi)$ is shown as below.

If $\tau_1 \leq \tau_2$,

$$\begin{aligned}\mathcal{A}(\Psi) =& W_{(1,\tau_1)} \bigcup \Big\{ O_{(j,1,l)} | j \in \Big[0, \Big\lfloor \frac{\tau_1}{T} \Big\rfloor \Big] \wedge j \notin F_2 \wedge j > \max F_1 \wedge l \in \mathcal{L} \wedge \\ & \tau_1 + p_{(1,l)} \leq \min(r_j + T_{max}, T_c) \Big\}.\end{aligned} \tag{3}$$

Otherwise,

$$\begin{aligned}\mathcal{A}(\Psi) =& W_{(2,\tau_2)} \bigcup \Big\{ O_{(j,2,l)} | j \in \Big[0, \Big\lfloor \frac{\tau_2}{T} \Big\rfloor \Big] \wedge j \notin F_1 \wedge j > \max F_2 \wedge l \in \mathcal{L} \wedge \\ & \tau_2 + p_{(2,l)} \leq \min(r_j + T_{max}, T_c) \Big\}.\end{aligned} \tag{4}$$

Please note, the action set for any stop state is empty, where $\mathcal{A}((T_c, T_c, F_1, F_2)) = \varnothing, \forall F_1, F_2$.

4.5. State Transition (Original Form)

We now further discuss the resultant state given the state and action. Let $r(\Psi, o_a)$ denote the resultant system state of performing a valid operation o_a at system state Ψ.

At a system state $\Psi = (\tau_1, \tau_2, F_1, F_2)$, w.l.o.g., suppose $\tau_1 \leq \tau_2$, indicating machine M_1 is the decision maker. M_1 can take one valid operation o_a from the action set of Ψ, where $o_a \in \mathcal{A}(\Psi)$, to transit to a resultant state $\Psi' = (\tau_1', \tau_2', F_1', F_2')$.

When o_a is an active operation, where $o_a = O_{(j_0,1,l_0)}, O_{(j_0,1,l_0)} \in \mathcal{A}(\Psi)$, M_1 will start the job j_0 with the processing level of l_0, then the resultant anchor time pair is $(\tau_1', \tau_2') \Leftarrow (\tau_1 + p_{(1,l_0)}, \tau_2)$. We first add the newly started job j_0 to the resultant job exclusion set F_1'. Next, F_1' and F_2' are updated from F_1 and F_2. In short, if a job k_0 is not necessarily to be recorded for machine M_1 (i.e., $k_0 < \max(j_0, \lceil \frac{\tau_1' + p_{(1,1)} - T_{max}}{T} \rceil)$), and it is not necessarily to be recorded for machine M_2 (i.e., $k_0 < \max(\max F_2, \lceil \frac{\tau_2' + p_{(2,1)} - T_{max}}{T} \rceil)$), then we can delete k_0 from F_1 to construct the resultant job exclusion set F_1'. For F_2', we follow the symmetrical manner to update the jobs.

When o_a is a waiting operation $W_{(1,\tau_1)}$, we have the resultant anchor time pair $(\tau_1', \tau_2') \Leftarrow (\tau_1 + 1, \tau_2)$. No job is started in such operation, then we only need to delete stale jobs (if there are any) in $\left[0, \lceil \frac{\min(\tau_1' + p_{(1,1)} - T_{max}, \tau_2' + p_{(2,1)} - T_{max})}{T} \rceil\right)$ which will definitely miss their absolute deadlines when τ_1 is increased.

Therefore, at the system state Ψ, M_1 can take a valid operation o_a to reach the resultant state $\Psi' = (\tau_1', \tau_2', F_1', F_2')$, where $(\tau_1', \tau_2', F_1', F_2') \Leftarrow r(\Psi, o_a)$. The formal derivation of the state transition is shown below.

$$\Psi' = (\tau_1', \tau_2', F_1', F_2') \Leftarrow r(\Psi, o_a). \tag{5}$$

If $o_a = O_{(j_0,1,l_0)}, O_{(j_0,1,l_0)} \in \mathcal{A}(\Psi)$, we obtain the resultant anchor time pair from $(\tau_1', \tau_2') \Leftarrow (\tau_1 + p_{(1,l_0)}, \tau_2)$, and update the resultant job exclusion sets F_1'

and F_2' from F_1 and F_2 as below.

$$F_1' \Leftarrow F_1 \bigcup j_0 \setminus \Bigg\{ j | j \in \mathcal{J} \wedge 0 \leq j < \min \Bigg(\max \Big(j_0, \Big\lceil \frac{\tau_1' + p_{(1,1)} - T_{max}}{T} \Big\rceil \Big), \max \Big(\max F_2, \Big\lceil \frac{\tau_2' + p_{(2,1)} - T_{max}}{T} \Big\rceil \Big) \Bigg) \Bigg\}. \tag{6a}$$

$$F_2' \Leftarrow F_2 \setminus \Bigg\{ j | j \in \mathcal{J} \wedge 0 \leq j < \min \Bigg(\max \Big(j_0, \Big\lceil \frac{\tau_1' + p_{(1,1)} - T_{max}}{T} \Big\rceil \Big), \max \Big(\max F_2, \Big\lceil \frac{\tau_2' + p_{(2,1)} - T_{max}}{T} \Big\rceil \Big) \Bigg) \Bigg\}. \tag{6b}$$

If $o_a = W_{(1,\tau_1)}, W_{(1,\tau_1)} \in \mathcal{A}(\Psi)$, the resultant anchor time pair is $(\tau_1', \tau_2') \Leftarrow (\tau_1 + 1, \tau_2)$, and the resultant job exclusion sets F_1' and F_2' are updated from F_1 and F_2 as below.

$$F_1' \Leftarrow F_1 \setminus \Bigg\{ j | j \in \mathcal{J} \wedge 0 \leq j < \min \Bigg(\Big\lceil \frac{\tau_1' + p_{(1,1)} - T_{max}}{T} \Big\rceil, \Big\lceil \frac{\tau_2' + p_{(2,1)} - T_{max}}{T} \Big\rceil \Bigg) \Bigg\}. \tag{7a}$$

$$F_2' \Leftarrow F_2 \setminus \Bigg\{ j | j \in \mathcal{J} \wedge 0 \leq j < \min \Bigg(\Big\lceil \frac{\tau_1' + p_{(1,1)} - T_{max}}{T} \Big\rceil, \Big\lceil \frac{\tau_2' + p_{(2,1)} - T_{max}}{T} \Big\rceil \Bigg) \Bigg\}. \tag{7b}$$

When $\tau_1 > \tau_2$, M_2 is the decision maker, and the state transition is symmetric.

4.6. Reward (Original Form)

When the decision maker takes an operation at a state, the system receives a reward, denoted as $\theta(\Psi, o_a)$, indicating the one-step utility that we received from taking operation o_a at system state Ψ. The active operation will generate a positive utility according to its processing level. Note that we have ensured that an active action will not generate a 0 utility (e.g., missing absolute deadlines or repeated processing) by removing those actions in the action set. The waiting operation will generate 0 utility. Suppose we are at system state $\Psi = (\tau_1, \tau_2, F_1, F_2)$, w.l.o.g., we have $\tau_1 \leq \tau_2$. M_1 starts the operation

$o_a \in \mathcal{A}(\Psi)$ at time τ_1 to obtain the one-step utility $\theta(\Psi, o_a)$. Then, we have

$$\theta(\Psi, o_a) \Leftarrow \begin{cases} u_l, & \text{if } o_a = O_{(j,1,l)}, O_{(j,1,l)} \in \mathcal{A}(\Psi); \\ 0, & \text{if } o_a = W_{(1,\tau_1)}, W_{(1,\tau_1)} \in \mathcal{A}(\Psi). \end{cases} \tag{8}$$

If $\tau_1 > \tau_2$, we have the symmetric formulation.

4.7. Bellman Equation (Original Form)

We can establish the Bellman equation based on the previous discussions on system state, action, state transition, and reward. We define $\Theta((\tau_1, \tau_2, F_1, F_2))$ as the accumulated utility at state $\Psi = (\tau_1, \tau_2, F_1, F_2)$. Given the system state $\Psi = (\tau_1, \tau_2, F_1, F_2)$ and a valid operation $o_a \in \mathcal{A}(\Psi)$, the accumulated utility of its resultant state Ψ' is defined as

$$\Theta(\Psi') = \Theta(\Psi) + \theta(\Psi, o_a). \tag{9}$$

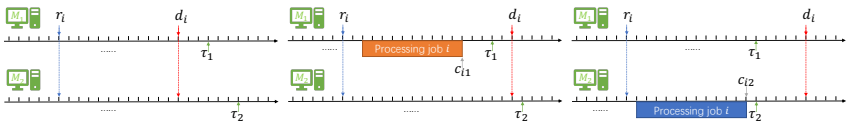
Let $\Theta^*(\Psi')$ denote the optimal accumulated utility at state Ψ', we have

$$\begin{aligned} \Theta^*(\Psi') = \max_{\forall \Psi, o_a,} & \{\Theta^*(\Psi) + \theta(\Psi, o_a)\}. \\ \text{s. t. } & r(\Psi, o_a) = \Psi'. \end{aligned} \tag{10}$$

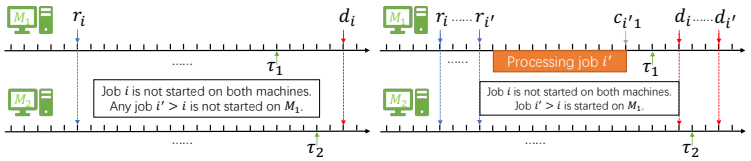
For the initial state $(0, 0, \varnothing, \varnothing)$, we set $\Theta^*((0, 0, \varnothing, \varnothing)) = 0$. The system ends when we reach (T_c, T_c, F_1, F_2), for some F_1, F_2. Then, the optimal accumulated utility for PCPT Problem is

$$\max_{F_1, F_2} \Theta^*((T_c, T_c, F_1, F_2)). \tag{11}$$

Through this formulation, a stop state with the highest accumulated utility is an optimal stop state. We can obtain the optimal sequence by recursively tracing back previous system states from the optimal stop state.



(a) Job i is in `OVERDUE 1`. (b) Job i is in `COMPLETED 1`. (c) Job i is in `RESIGNED 1`.



(d) Job i is in `UNCOMMITTED 1`. (e) Job i is in `IGNORED 1`.

Figure 3: Illustration of Job Conditions in M_1.

4.8. Size of State Space

In this section, even if we have utilised the OSS Property, the complexity so far is still non-polynomial. Jobs released in time $[\tau_1 - T_{max}, \tau_1]$ can be recorded in F_1, and jobs released in time $[\tau_1 - T_{max}, \tau_1 + p_{(2,L)})$ can be recorded in F_2. Therefore, we have at most $2^{\lceil \frac{T_{max}}{T} \rceil} \cdot 2^{\lceil \frac{T_{max}+p_{(2,L)}}{T} \rceil}$ possibilities for F_1 and F_2. The state space is not polynomial with respect to $\frac{T_{max}}{T}$. If we directly employ dynamic programming at this stage, the computational complexity is not polynomial. In what follows, we target to further simplify the state space.

5. Bellman Formula Re-formulation with Pseudo-polynomial Complexity

In this section, we target to reduce the state space to pseudo-polynomial complexity. We first explore the *Consecutive Decision Making (CDM) Property*, so that we do not need to record a number of jobs in F_1 and F_2, but to record only two jobs. Through this, we can reformulate the state, action, state transition, and reward, so that the resultant dynamic programming is in pseudo-polynomial complexity.

5.1. The Consecutive Decision Making (CDM) Property

5.1.1. Definition of Job Conditions

In order to further characterise which jobs a machine can or cannot process, we firstly define *job conditions*[2] given an anchor time pair.

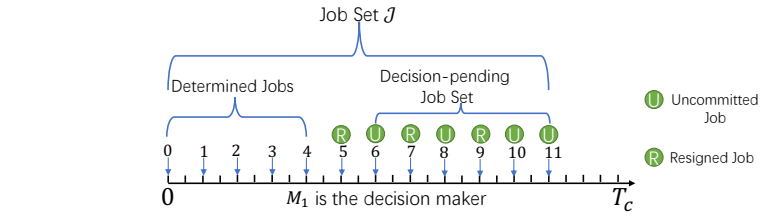
Given any (τ_1, τ_2), any job $i \in \mathcal{J}$ has one job condition in M_1 and in M_2 respectively. The job conditions in two machines are denoted as (σ_1, σ_2). In M_1, σ_1 belongs to one of the following job conditions: `OVERDUE 1`: Job i is overdue for machine M_1 at τ_1 (no matter if it is completed or not), i.e., $d_i < \tau_1$ (Fig. 3(a)); `COMPLETED 1`: Job i is completed by machine M_1 on or before τ_1 and not overdue for machine M_1 at τ_1, i.e., $c_{i1} \leq \tau_1 \leq d_i$ (Fig. 3(b)); `RESIGNED 1`: Job i is not overdue for machine M_1 at τ_1. However, job i is taken by the machine M_2 (to be completed on or before τ_2). In other words, we have $c_{i2} \leq \tau_2 \wedge d_i \geq \tau_1$ (Fig. 3(c)); `UNCOMMITTED 1`: Job i is not overdue for machine M_1 at τ_1, and it is not taken by any machine on or before $\max(\tau_1, \tau_2)$, and there exists no such job i', where $i' > i$, that i' is completed by machine M_1 on or before τ_1. In other words, we have $d_i \geq \tau_1 \wedge c_{i1} > \tau_1 \wedge c_{i2} > \tau_2 \wedge \nexists i', i' > i, c_{i'1} \leq \tau_1$ (Fig. 3(d)); `IGNORED 1`: Job i is not overdue for machine M_1 at τ_1, and it is not taken by any machine before $\max(\tau_1, \tau_2)$, but there exists a job i', where $i' > i$, such that i' is completed by machine M_1 on or before τ_1. (Due to the OSS Property, job i will never be done by M_1 after $c_{i'1}$.) In other words, we have $d_i \geq \tau_1 \wedge c_{i1} > \tau_1 \wedge c_{i2} > \tau_2 \wedge \exists i', i' > i, c_{i'1} \leq \tau_1$ (Fig. 3(e)). For jobs under `OVERDUE 1`, `COMPLETED 1`, and `IGNORED 1`, we know the fate of jobs through M_1 only, without checking the status of M_2. For jobs under `RESIGNED 1` and `UNCOMMITTED 1`, we do not know the fate of jobs without checking with M_2. Therefore, to avoid re-processing, we should find a way to flag if a job is `RESIGNED 1` or `UNCOMMITTED 1`. In the next, we avoid flagging each job individually (causing a large stage space), but to use two flags only.

In M_2, σ_2 belongs to one of the following job conditions: `OVERDUE 2`, `COMPLETED 2`, `RESIGNED 2`, `UNCOMMITTED 2`, and `IGNORED 2`, which can be defined symmetri-

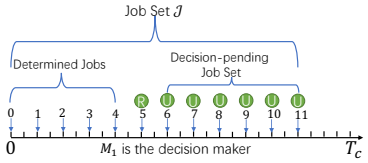
[2] We do not use "state" to avoid confusion with "system state".

cally.

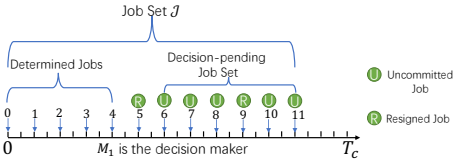
Given any anchor time pair (τ_1, τ_2), w.l.o.g., $\tau_1 \leq \tau_2$ (M_1 is the decision maker), if a job is under `UNCOMMITTED 1`, it is called an *uncommitted job*. If a job is under `RESIGNED 1`, it is called a *resigned job*. All other jobs are called *determined jobs*. If job i is an uncommitted job, then any job i' with $i' > i$ can only be an uncommitted job or a resigned job. Job i' cannot be `OVERDUE 1` in M_1, because job i has an earlier absolute deadline than job i'. i' cannot be `COMPLETED 1` or `IGNORED 1`, because job i will be `IGNORED 1` if any job released after i is completed by M_1. Therefore, from the earliest uncommitted job till the last job, there exist only uncommitted and resigned jobs, which is called the *decision-pending job set*, as shown in Fig. 4(a). Note that for the case $\tau_1 < \tau_2$, we have the symmetric property, and the discussion on that case is omitted.



(a) Definition of determined jobs and decision-pending job set. (Due to Theorem 3, the above situation shown in (a) is not possible. Only (b) and (c) shown below are possible.)



(b) By Theorem 3, the decision-pending job set without any resigned job.



(c) By Theorem 3, the decision-pending job set with one resigned job.

Figure 4: Illustration of decision-pending job set.

5.1.2. Job State Based Interpretation of the CDM Property

In what follows, we present the CDM Property, so that we can flag two values to distinguish resigned jobs from uncommitted jobs in the decision-pending job set.

Theorem 3. *There exists an optimal sequence $S^* = (S_1^*, S_2^*)$ to this PCPT Problem, satisfying the OSS Property and the following conditions:*
(1.1). For any $s_{(1,n_1)} \in S_1^*$, for any job $j' \in \mathcal{J}$ where $b_{j'} \geq t_{(1,n_1)}$, and $r_{j'} \leq t_{(1,n_1)} < t_{(1,n_1)} + p_{(1,l_{(1,n_1)})} \leq d_{j'}$, we have $j_{(1,n_1)} < j'$;
(1.2) [Symmetric to (1.1)]. For any $s_{(2,n_2)} \in S_2^*$, for any job $j' \in \mathcal{J}$ where $b_{j'} > t_{(2,n_2)}$, and $r_{j'} \leq t_{(2,n_2)} < t_{(2,n_2)} + p_{(2,l_{(2,n_2)})} \leq d_{j'}$, we have $j_{(2,n_2)} < j'$.
(2.1). For any anchor time pair (τ_1, τ_2) of the sequence S^*, when $\tau_1 \leq \tau_2$, if we have a job j that is under `UNCOMMITTED 1` in M_1, then we have at most one job j', where $j < j'$, such that j' is under `RESIGNED 1` in M_1;
(2.2) [Symmetric to (2.1)]. For any anchor time pair (τ_1, τ_2) of the sequence S^*, when $\tau_1 > \tau_2$, if we have a job j that is under `UNCOMMITTED 2` in M_2, then we have at most one job j', where $j < j'$, such that j' is under `RESIGNED 2` in M_2.

(1.1) and (1.2) show that if we have jobs $j < j'$, and it is feasible to process both of them at the decision making time with the same processing level to meet their absolute deadlines, then we should process the earlier released job j. (2.1) and (2.2) show that there exists at most one resigned job in decision-pending job set.

The detailed proof can be found in Appendix C. The CDM Property proves that flagging two jobs is sufficient. At any anchor time pair (τ_1, τ_2), w.l.o.g., $\tau_1 \leq \tau_2$, we consider two cases: (1) There exists no resigned job in decision-pending job set in M_1, as shown in Fig. 4(b). In this case, all uncommitted jobs are consecutive. We can flag one job (job 5 in Fig. 4(b)) to indicate where the decision-pending job set begins; (2) There exists one resigned job in decision-pending job set in M_1, as shown in Fig. 4(c). In this case, the uncommitted jobs are divided into two groups by the latest resigned job. We flag one job (job 5 in Fig. 4(c)) to indicate the beginning of the decision-pending job set, and flag the latest resigned job (job 9 in Fig. 4(c)) to avoid reprocessing it. There is at most one resigned job in the middle of the decision-pending job set so that we only need one more flag here.

Note that $\tau_1 > \tau_2$ case has the symmetrical property. Therefore, we only need to flag two jobs to construct the new system state.

5.2. System State (Simplified Form)

Based on the CDM Property, we can simplify the form of dynamic programming. Let $\Pi = (\tau_1, \tau_2, f_1, f_2)$ denote the simplified system state, where (τ_1, τ_2) is the anchor time pair, and f_1 and f_2 are the *Flagged Job Indices*.

Let $e_1 = \lceil \frac{\tau_1 + p_{(1,1)} - T_{max}}{T} \rceil$ (resp. $e_2 = \lceil \frac{\tau_2 + p_{(2,1)} - T_{max}}{T} \rceil$) denote the earliest job that can be possibly started by machine M_1 (resp. M_2) at time τ_1 (resp. τ_2) to meet the absolute deadline. Recall the definitions that we have the historical sets $\mathcal{R}_1$ and $\mathcal{R}_2$, and the last completed jobs R_1 and R_2 (for machines M_1 and M_2 respectively). The flagged job index f_1 considers four conditions A, B, C, and D as below. A: $R_1 \geq e_1$; B: $R_1 \geq \max(R_2, e_2)$; C: $R_1 + 1 \notin \mathcal{R}_2$; D: $\min\big(e_1, \max(R_2, e_2)\big) \notin \mathcal{R}_2$.

Then, we flag f_1 by following one of cases below: (a) Under $(A \vee B) \wedge C$, we flag $f_1 = R_1$; (b) Under $(A \vee B) \wedge \bar{C}$, we flag $f_1 = \min\{j | j > R_1 + 1, j \notin \mathcal{R}_2\} - 1$; (c) Under $(\overline{A \vee B}) \wedge D$, we flag $f_1 = \min\big(e_1, \max(R_2, e_2)\big) - 1$; (d) Under $(\overline{A \vee B}) \wedge \bar{D}$, we flag $f_1 = \min\Big\{k | k > \min\big(e_1, \max(R_2, e_2)\big), k \notin \mathcal{R}_2\Big\} - 1$.

The flagged job index f_2 also considers four conditions E, F, G, and H as below. E: $R_2 \geq e_2$; F: $R_2 \geq \max(R_1, e_1)$; G: $R_2 + 1 \notin \mathcal{R}_1$; H: $\min\big(e_2, \max(R_1, e_1)\big) \notin \mathcal{R}_1$. Then, f_2 can be characterised as below: (a) Under $(E \vee F) \wedge G$, we flag $f_2 = R_2$; (b) Under $(E \vee F) \wedge \bar{G}$, we flag $f_2 = \min\{j | j > R_2 + 1, j \notin \mathcal{R}_1\} - 1$; (c) Under $(\overline{E \vee F}) \wedge H$, we flag $f_2 = \min\big(e_2, \max(R_1, e_1)\big) - 1$; (d) Under $(\overline{E \vee F}) \wedge \bar{H}$, we flag $f_2 = \min\Big\{k | k > \min\big(e_2, \max(R_1, e_1)\big), k \notin \mathcal{R}_1\Big\} - 1$.

5.3. Action (Simplified Form)

For each system state $\Pi = (\tau_1, \tau_2, f_1, f_2)$, we redefine $\mathcal{G}(\Pi)$ as the action set (simplified form), which follows all the conditions ①–⑤ in Section 4.4. Due to the CDM Property, we add one more condition

⑥ Given any system state, for any processing level, we only start the earliest job which satisfies all the previous five conditions.

Then, for $\Pi = (\tau_1, \tau_2, f_1, f_2)$ (not a stop state), the formal definition of the action set $\mathcal{G}(\Pi)$ can be defined as below.

If $\tau_1 \leq \tau_2$,

$$\mathcal{G}(\Pi) = W_{(1,\tau_1)} \bigcup \Big\{ O_{(j,1,l)} | l \in \mathcal{L}, j = \min \Big\{ j' | j' \in \Big[0, \Big\lfloor \frac{\tau_1}{T} \Big\rfloor \Big] \wedge j' \neq f_2 \wedge j' > f_1 \wedge \tau_1 + p_{(1,l)} \leq \min\{r_{j'} + T_{max}, T_c\} \Big\} \Big\}. \tag{12}$$

Otherwise,

$$\mathcal{G}(\Pi) = W_{(2,\tau_2)} \bigcup \Big\{ O_{(j,2,l)} | l \in \mathcal{L}, j = \min \Big\{ j' | j' \in \Big[0, \Big\lfloor \frac{\tau_2}{T} \Big\rfloor \Big] \wedge j' \neq f_1 \wedge j' > f_2 \wedge \tau_2 + p_{(2,l)} \leq \min\{r_{j'} + T_{max}, T_c\} \Big\} \Big\}. \tag{13}$$

The action set for any stop state is empty, where $\mathcal{G}((T_c, T_c, f_1, f_2)) = \varnothing$, for any f_1, f_2.

5.4. State Transition (Simplified Form)

With the system state and the action set in simplified form, we can redefine the state transition (simplified form). Let $v(\Pi, o_g)$ denote the resultant state that is transited from Π by taking an operation $o_g \in \mathcal{G}(\Pi)$.

Suppose we are at $\Pi = (\tau_1, \tau_2, f_1, f_2)$, w.l.o.g., we have $\tau_1 \leq \tau_2$ indicating that machine M_1 is the decision maker. Machine M_1 can take a valid operation $o_g \in \mathcal{G}(\Pi)$ to transit to the resultant state $\Pi' = (\tau_1', \tau_2', f_1', f_2')$, which can be defined as below.

$$\Pi' = (\tau_1', \tau_2', f_1', f_2') \Leftarrow v(\Pi, o_g). \tag{14}$$

If o_g is an active operation $O_{(k_1,1,l_1)}$, where $o_g = O_{(k_1,1,l_1)}, O_{(k_1,1,l_1)} \in \mathcal{G}(\Pi)$, we have the resultant anchor time pair $(\tau_1', \tau_2') \Leftarrow (\tau_1 + p_{(1,l_1)}, \tau_2)$. We have $e_1' = \lceil \frac{\tau_1' + p_{(1,1)} - T_{max}}{T} \rceil$ and $e_2' = \lceil \frac{\tau_2' + p_{(2,1)} - T_{max}}{T} \rceil$. Then, we update the resultant

flagged jobs f_1' and f_2' from f_1 and f_2 by following the equations as below.

$$f_1' \Leftarrow \begin{cases} k_1, & \text{if } k_1 \geq \min\left(e_1', \max(f_2, e_2')\right) \wedge k_1 + 1 \neq f_2; \\ f_2, & \text{if } k_1 \geq \min\left(e_1', \max(f_2, e_2')\right) \wedge k_1 + 1 = f_2; \\ \min\left(e_1', \max(f_2, e_2')\right) - 1, & \text{if } k_1 < \min\left(e_1', \max(f_2, e_2')\right) \\ & \quad \wedge \min\left(e_1', \max(f_2, e_2')\right) \neq f_2; \\ f_2, & \text{otherwise.} \end{cases} \tag{15}$$

$$f_2' \Leftarrow \begin{cases} f_2, & \text{if } f_2 \geq \min\left(e_2', \max(k_1, e_1')\right) \wedge f_2 + 1 \neq k_1; \\ k_1, & \text{if } f_2 \geq \min\left(e_2', \max(k_1, e_1')\right) \wedge f_2 + 1 = k_1; \\ \min\left(e_2', \max(k_1, e_1')\right) - 1, & \text{if } f_2 < \min\left(e_2', \max(k_1, e_1')\right) \\ & \quad \wedge \min\left(e_2', \max(k_1, e_1')\right) \neq k_1; \\ k_1, & \text{otherwise.} \end{cases} \tag{16}$$

If o_g is a waiting operation $W_{(1,\tau_1)}$, where $o_g = W_{(1,\tau_1)}, W_{(1,\tau_1)} \in \mathcal{G}(\Pi)$, the resultant anchor time pair is $(\tau_1', \tau_2') \Leftarrow (\tau_1 + 1, \tau_2)$. We have $e_1' = \lceil \frac{\tau_1' + p_{(1,1)} - T_{max}}{T} \rceil$ and $e_2' = \lceil \frac{\tau_2' + p_{(2,1)} - T_{max}}{T} \rceil$. Then, we update the resultant flagged jobs f_1' and f_2' from f_1 and f_2 as below.

$$f_1' \Leftarrow \begin{cases} \min(e_1', e_2') - 1, & \text{if } f_1 < \min\left(e_1', e_2'\right) \wedge \min(e_1', e_2') \neq f_2; \\ f_2, & \text{if } f_1 < \min\left(e_1', e_2'\right) \wedge \min(e_1', e_2') = f_2; \\ f_1, & \text{otherwise.} \end{cases} \tag{17}$$

$$f_2' \Leftarrow \begin{cases} \min(e_1', e_2') - 1, & \text{if } f_2 < \min(e_1', e_2') \wedge \min(e_1', e_2') \neq f_1; \\ f_1, & \text{if } f_2 < \min(e_1', e_2') \wedge \min(e_1', e_2') = f_1; \\ f_2, & \text{otherwise.} \end{cases} \tag{18}$$

For the $\tau_1 > \tau_2$ case, M_2 is the decision maker, the simplified state transition is symmetric.

5.5. Reward (Simplified Form)

The reward (simplified form) is as follows. We define $\omega(\Pi, o)$ as the one-step utility of taking an action o_g at system state Π. Suppose we are at the $\Pi = (\tau_1, \tau_2, f_1, f_2)$, w.l.o.g., we have $\tau_1 \leq \tau_2$. Machine M_1 takes an action $o_g \in \mathcal{G}(\Pi)$ to obtain its one-step utility $\omega(\Pi, o_g)$. Then, the one-step utility equation is

$$\omega(\Pi, o_g) \Leftarrow \begin{cases} u_l, & \text{for } o_g = O_{(j,1,l)}, O_{(j,1,l)} \in \mathcal{G}(\Pi); \\ 0, & \text{for } o_g = W_{(1,\tau_1)}, W_{(1,\tau_1)} \in \mathcal{G}(\Pi). \end{cases} \tag{19}$$

Please note, in $\tau_1 > \tau_2$ case, M_2 is the decision maker, the one-step utility can be formulated symmetrically.

5.6. Bellman Equation (Simplified Form)

After we simplify the system state and action set, and redefine the system transitions, we can reformulate the Bellman equation in simplified form.

We define $\Omega(\Pi)$ as the accumulated utility of the system state Π. Given a system state $\Pi = (\tau_1, \tau_2, f_1, f_2)$ and an action o_g, where $o_g \in \mathcal{G}(\Pi)$, the accumulated utility at $\Pi' = v(\Pi, o_g)$ is

$$\Omega(\Pi') = \Omega(\Pi) + \omega(\Pi, o_g). \tag{20}$$

Let $\Omega^*(\Pi)$ denote the optimal accumulated utility for system state Π. The bellman equation (simplified form) can be redefined as

$$\begin{aligned} \Omega^*(\Pi') = & \max_{\forall \Pi, o_g,} \{\Omega^*(\Pi) + \omega(\Pi, o_g)\}. \\ & \text{s. t. } v(\Pi, o_g) = \Pi'. \end{aligned} \tag{21}$$

We define $\Omega^*((0, 0, -1, -1)) = 0$ as the accumulated utility for the initial

state. The optimal accumulated utility for the PCPT Problem is

$$\max_{f_1,f_2} \Omega^*((T_c, T_c, f_1, f_2)). \tag{22}$$

Any stop state with the highest accumulated utility is an optimal stop state. Finally, we trace back all the intermediate system states between the initial state and the optimal stop state to obtain the optimal sequence.

5.7. Size of State Space (Simplified Form)

In Section 5, we proposed the CDM Property and the system state (simplified form), leading to a pseudo-polynomial state space. Given any anchor time pair (τ_1, τ_2), w.l.o.g., suppose $\tau_1 \leq \tau_2$. Jobs released in time $[\tau_1 - T_{max}, \tau_1]$ can be flagged by f_1, and jobs released in time $[\tau_1 - T_{max}, \tau_1 + p_{(2,L)})$ can be flagged by f_2. Therefore, we have at most $\lceil \frac{T_{max}}{T} \rceil$ possibilities for f_1, and $\lceil \frac{T_{max}+p_{(2,L)}}{T} \rceil$ possibilities for f_2. If $\tau_1 > \tau_2$, we have the symmetric property. Let $p_{max} = \max\{p_{(1,L)}, p_{(2,L)}\}$ denote the maximum possible processing time. Thus, there are at most $\lceil \frac{T_{max}}{T} \rceil \cdot \lceil \frac{T_{max}+p_{max}}{T} \rceil$ possibilities for the flags.

6. PCPT Solver

6.1. PCPT-Solver Design

Following the conclusions in Section 5, as in Algorithm 1, we propose an optimal PCPT solver (PCPT-solver) by dynamic programming to optimally solve the PCPT Problem.

We employ the simplified form $\Pi = (\tau_1, \tau_2, f_1, f_2)$ in the algorithm. We maintain a dynamic dictionary `DICT` to indicate which $(\tau_1, \tau_2, f_1, f_2)$ has been visited. Whenever $(\tau_1, \tau_2, f_1, f_2)$ is visited, it is added to the dictionary by `DICT.add`$\big((\tau_1, \tau_2, f_1, f_2)\big)$. Also, given (τ_1, τ_2), we are able to search all visited f_1 and f_2 values in the dictionary. In other words, `DICT.query`(τ_1, τ_2) returns the set of existing states $(\tau_1, \tau_2, f_1, f_2)$ in `DICT`. We employ another dynamic dictionary `FS` to memorise the current optimal father state for any existing state. In other words, if we take an action to transit the state from Π to Π',

so that Π' obtains the highest accumulated utility, then we add (if Π' is not in `FS`) or update (if Π' is in `FS`) Π as the current optimal father state of Π' by `FS.update`$(\Pi') = \Pi$. Among all existing stop states, we have a stop state that has the highest accumulated utility. We employ `OS` to memorise this stop state, and this accumulated utility value will be memorised by `Optimal`. Finally, we employ a list `OptimalPath` to memorise the optimal path (i.e., optimal historical decisions). We can track all the father states from the optimal stop state to the initial state by `OptimalPath.append(OS)`.

In the Algorithm, Lines 1–3 show the initialisation of the algorithm. In Lines 4–5, we aim to traverse all the anchor time pairs. At each anchor time pair, we have multiple possible flagged jobs (Line 6), leading to different system states (Line 7). Following (12)–(13), we consider all possible actions for the current system state in Line 8. As shown in Lines 10–11, for each action taken, we can follow (14)–(19) to reach the resultant state and calculate the reward. If the resultant state does not exist, it is added in dictionary `DICT` and its accumulated utility is initialised as $-\infty$ (Lines 13–14). If the new state transition can generate a greater accumulated utility, according to (20)–(21), we memorise this value for this resultant state and update the father state (Lines 15–17). In Lines 18–21, if we reach a new stop state, we compare its accumulated utility with `Optimal`; if the new value is larger, we update the optimal state `OS` and `Optimal` accordingly. In Lines 22–24, we trace back the optimal father states from the optimal stop state to obtain the optimal path.

6.2. Complexity Analysis of the PCPT-Solver

In Section 6.1, we proposed Algorithm 1 to optimally solve PCPT Problem. The computational complexity of this algorithm can be analysed through 3 levels. In the first level, as shown in Lines 4–5, we consider anchor time pairs (τ_1, τ_2). We have $T_c \leq J \cdot T$ possible values for τ_1. For each τ_1 value we have $2p_{(2,L)}$ possible values for τ_2. In summary, we have at most $J \cdot T \cdot 2p_{max}$ possible anchor time pairs. In the second level, we consider the flagged jobs (f_1, f_2) in Line 6. As we discussed above, given the anchor time pair (τ_1, τ_2), there are at

Algorithm 1: PCPT-solver

Initialise DICT $\Leftarrow \emptyset$, FS $\Leftarrow \emptyset$, Optimal $\Leftarrow -\infty$, OS $\Leftarrow ()$, OptimalPath $\Leftarrow []$
DICT.add$\big((0, 0, -1, -1)\big)$
FS.update$\big((0, 0, -1, -1)\big) = \emptyset$
for $\tau_1 \in [0, T_c]$ **do**
  for $\tau_2 \in [\max(0, \tau_1 - p_{(2,L)}), \tau_1 + p_{(2,L)})$ **do**
    for $(f_1, f_2) \in$ *DICT.query*(τ_1, τ_2) **do**
      $\Pi = (\tau_1, \tau_2, f_1, f_2)$
      for $o_g \in \mathcal{G}(\Pi)$ **do**
        ▷ (12)–(13).
        $\Pi' = (\tau_1', \tau_2', f_1', f_2') \Leftarrow v(\Pi, o_g)$ ▷ (14)–(18).
        $u = \omega(\Pi, o_g)$ ▷ (19).
        if $(f_1', f_2') \notin$ *DICT.query*(τ_1', τ_2') **then**
          DICT.add(Π')
          $\Omega(\Pi') = -\infty$
        if $\Omega(\Pi') < \Omega(\Pi) + u$ **then**
          $\Omega(\Pi') = \Omega(\Pi) + u$
          FS.update$(\Pi') = \Pi$ ▷ (20)–(21).
        if $(\tau_1', \tau_2') == (T_c, T_c)$ **then**
          if *Optimal* $< \Omega(\Pi')$ **then**
            Optimal $\Leftarrow \Omega(\Pi')$
            OS $\Leftarrow \Pi'$
while *FS*(*OS*) $\neq \emptyset$ **do**
  OptimalPath.append(OS)
  OS $\Leftarrow$ FS(OS)
OptimalPath.append(OS)
return *Optimal*,*OptimalPath*

most $\lceil \frac{T_{max}}{T} \rceil \cdot \lceil \frac{T_{max}+p_{max}}{T} \rceil$ combinations of possible flagged jobs. Thirdly, we consider the action set for each state in Line 8. Following (12)–(13), at each state, for any processing level, we only select one earliest job to be started. In addition, we also allow the waiting operation. Then, we have at most $L + 1$ possible operations for each system state.

As a consequence, the overall complexity of this algorithm is $\mathcal{O}(J \cdot T \cdot p_{max} \cdot \lceil \frac{T_{max}}{T} \rceil \cdot \lceil \frac{T_{max}+p_{max}}{T} \rceil \cdot L)$, which is within pseudo-polynomial computational complexity.

7. Performance Evaluation

In this section, we present simulation results to evaluate Algorithm 1 and demonstrate its advantages compared with two benchmark schemes.

7.1. Simulation Setup

We adopt one slow machine (device) and one fast machine (server). We consider 3 processing levels for the two machines. For all options, the processing times and the utilities are given in Table 1. The data we listed in Table 1 are generated from a real-world video recognition application as we discussed in the first example in Section 1.

The default values for other parameters are given below. We set the inter-arrival time $T = 33$ ms. We set the relative deadline $T_{max} = 150$ ms. In our simulation, we assume that jobs continuously arrive until the time the system shuts down, so we can use T_c to represent the number of jobs. The number of jobs is $J = \lfloor \frac{T_c}{T} \rfloor$ jobs accordingly. We set the default value of T_c as 3000 ms, and the default number of jobs is $J = 90$.

7.2. Benchmark Schemes

We adopt two heuristic benchmarks, the *Utility First Greedy (UFG)* algorithm and the *Efficiency First Greedy (EFG)* algorithm.

The UFG algorithm will make the decision as below: When one of the machines is ready to make a decision, it takes the option which can meet the

Table 1: Processing Times and Utilities.

Processing Levels & Machines		Processing Time (ms)	Utility	Utility / ms
1	Device	72	53	0.74
	Server	55		0.96
2	Device	90	68	0.76
	Server	69		0.99
3	Device	115	78	0.68
	Server	87		0.90

absolute deadline and generates the most utilities; When both machines become idle at the same time, the server will make the decision first and select the processing option with the most utility without breaching the absolute deadline; when one machine becomes idle but it does not have any available jobs, it just waits for the next job.

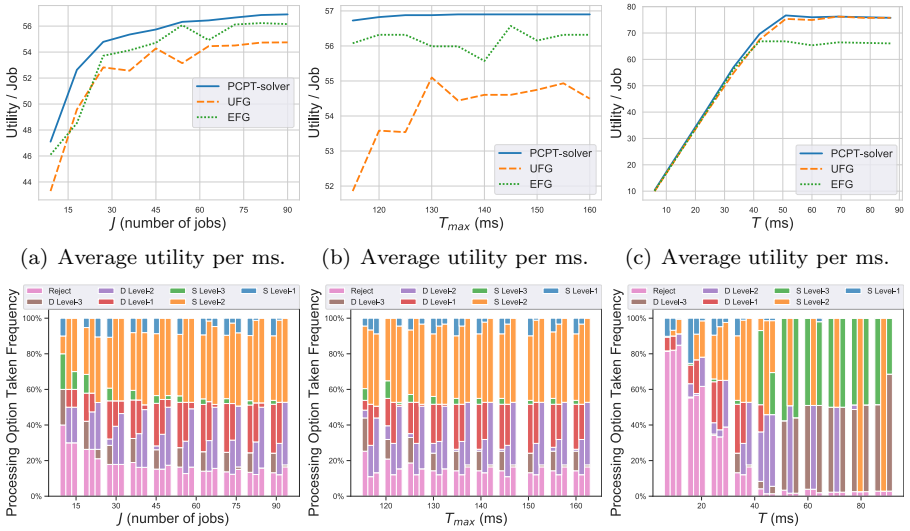
The EFG algorithm follows the strategy as below: When one of the machines becomes idle, it takes the option which can meet the absolute deadline and has the highest utility-to-time ratio; when both machines become idle at the same time, the server will make the decision first and select the processing option with the highest utility-to-time ratio and without breaching the absolute deadline; when one machine becomes idle but does not have any available job, it just waits for the next job.

7.3. Results and Discussion

7.3.1. Results under different J (number of jobs)

First, we evaluate the performance of different algorithms under the different J values. We set $J = [9, 18, 27, 36, 45, 54, 63, 72, 81, 90]$ ($T_c = [300, 600, 900, 1200, 1500, 1800, 2100, 2400, 2700, 3000]$ ms accordingly). We keep all other parameters at their default values. For a fair comparison under different J, Fig. 5(a) shows the average utility per job. Fig. 5(d) shows the percentage of jobs choosing different options under three strategies. In Fig. 5(d), each group has 3 bars, from left to right we have: UFG, EFG, and PCPT-solver.

In Fig. 5(a), PCPT-solver achieves the best overall utility compared with



(a) Average utility per ms. (b) Average utility per ms. (c) Average utility per ms.

(d) The percentage of options. (e) The percentage of options. (f) The percentage of options.

Figure 5: Results under different J (Fig. 5(a) and Fig. 5(d)), T_{max} (Fig. 5(b) and Fig. 5(e)), and T (Fig. 5(c) and Fig. 5(f)). "D" and "S" indicate jobs processed at the device and server respectively. "Level" indicates the processing level. "Reject" indicates that the jobs are not processed.

the other algorithms in all situations. In Fig.5(d), the UFG tends to select the 3rd level processing operations, because it can greedily generate the highest utility when a decision is made. However, this decision is ineffective, with the lowest utility-to-time ratio. Therefore, the UFG has the worst performance in most situations. The EFG algorithm has better performance, because it greedily selects more effective processing options. However, it is still worse than PCPT-solver. We can see that PCPT-solver chooses more level-2 options. It also actively rejects some jobs to earn more time to process the rest of the jobs with a better processing option, and thus the overall utility is maximised.

7.3.2. Results under different T_{max}

Next, we evaluate the performance of different algorithms under different T_{max} values, where $T_{max} = [115, 120, 125, 130, 135, 140, 145, 150, 155, 160]$ ms. Other values follow their default settings. PCPT-solver outperforms the other two benchmark schemes under all T_{max} values. From Fig. 5(b), we can see that

the performance of PCPT-solver is stable in all cases. As shown in Fig. 5(e), PCPT-solver dynamically adjusts its options to keep more jobs processed by level-2 under different T_{max} values.

7.3.3. Results under different T

Finally, we evaluate the performance of different algorithms under different T values. T_c and T_{max} are kept as their default settings for a fair comparison. For all cases, as shown in Fig. 5(c), PCPT-solver achieves the best performance. When T is small, all algorithms have similar performance. This is because a small T value leads to more jobs in the system. Under this situation, as shown in Fig. 5(f), all algorithms will miss the majority of the jobs, so that their performances are close. By increasing T, the system will have fewer jobs (smaller J). The EFG is able to process more jobs with the most effective processing option level-2 at the server. However, if T is greater than the processing time of level-2 at the server, the server completes a job before the arrival of the next job. This will cause a waste of time, leading to worse performance. When T is large enough, both the UFG and the PCPT-solver can process all jobs with level-3. Then, they will have a similar performance.

8. Conclusion

In this paper, we study the scheduling problem of periodic jobs with discretely controllable processing times on two machines. This problem is modelled from many real-world examples. It cannot be solved directly by dynamic programming due to its large state space. In order to address this issue, we proved the OSS and the CDM Properties to highly reduce the space to pseudo-polynomial complexity. Consequently, our proposed PCPT-solver can optimally solve this problem with a pseudo-polynomial computational complexity.

References

[1] Z. Wang, W. Bao, D. Yuan, L. Ge, N. H. Tran, A. Y. Zomaya, See: Scheduling early exit for mobile dnn inference during service outage, in: Proceed-

ings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWIM '19, Miami Beach, FL, USA, 2019.

[2] E. Li, L. Zeng, Z. Zhou, X. Chen, Edge ai: On-demand accelerating deep neural network inference via edge computing, IEEE Transactions on Wireless Communications 19 (1) (2020) 447–457.

[3] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, N. D. Lane, Spinn: synergistic progressive inference of neural networks over device and cloud, in: Proceedings of the 26th annual international conference on mobile computing and networking, 2020, pp. 1–15.

[4] Z. Huang, F. Dong, D. Shen, H. Wang, X. Guo, S. Fu, Enabling latency-sensitive dnn inference via joint optimization of model surgery and resource allocation in heterogeneous edge, in: Proceedings of the 51st International Conference on Parallel Processing, 2022, pp. 1–11.

[5] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wu, X. Shen, Distributed artificial intelligence empowered by end-edge-cloud computing: A survey, IEEE Communications Surveys & Tutorials (2022).

[6] M. R. Zakerinasab, M. Wang, Does chunk size matter in distributed video transcoding?, in: 2015 IEEE 23rd International Symposium on Quality of Service (IWQoS), 2015, pp. 69–70.

[7] K.-B. Chen, H.-Y. Chang, Complexity of cloud-based transcoding platform for scalable and effective video streaming services, Multimedia Tools and Applications 76 (19) (2017) 19557–19574.

[8] X. Li, M. A. Salehi, Y. Joshi, M. K. Darwich, B. Landreneau, M. Bayoumi, Performance analysis and modeling of video transcoding using heterogeneous cloud services, IEEE Transactions on Parallel and Distributed Systems 30 (4) (2018) 910–922.

[9] X. Jiang, F. R. Yu, T. Song, V. C. Leung, A survey on multi-access edge computing applied to video streaming: Some research issues and challenges, IEEE Communications Surveys & Tutorials 23 (2) (2021) 871–903.

[10] R. Viola, Á. Martín, M. Zorrilla, J. Montalbán, P. Angueira, G.-M. Muntean, A survey on virtual network functions for media streaming: Solutions and future challenges, ACM Computing Surveys 55 (11) (2023) 1–37.

[11] M. A. Khan, E. Baccour, Z. Chkirbene, A. Erbad, R. Hamila, M. Hamdi, M. Gabbouj, A survey on mobile edge computing for video streaming: Opportunities and challenges, IEEE Access (2022).

[12] M. U. Younus, S. ul Islam, I. Ali, S. Khan, M. K. Khan, A survey on software defined networking enabled smart buildings: Architecture, challenges and use cases, Journal of Network and Computer Applications 137 (2019) 62–77.

[13] B. Dong, V. Prakash, F. Feng, Z. O'Neill, A review of smart building sensing system for better indoor environment control, Energy and Buildings 199 (2019) 29–46.

[14] K. Alanne, S. Sierla, An overview of machine learning applications for smart buildings, Sustainable Cities and Society 76 (2022) 103445.

[15] R. Vickson, Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine, Operations Research 28 (5) (1980) 1155–1167.

[16] R. Vickson, Two single machine sequencing problems involving controllable job processing times, AIIE transactions 12 (3) (1980) 258–262.

[17] E. Nowicki, S. Zdrzałka, A survey of results for sequencing problems with controllable processing times, Discrete Applied Mathematics 26 (2-3) (1990) 271–287.

[18] T. Cheng, Q. Ding, B. M. Lin, A concise survey of scheduling with time-dependent processing times, European Journal of Operational Research 152 (1) (2004) 1–13.

[19] D. Shabtay, G. Steiner, A survey of scheduling with controllable processing times, Discrete Applied Mathematics 155 (13) (2007) 1643–1666.

[20] T. Cheng, Z.-L. Chen, C.-L. Li, Single-machine scheduling with trade-off between number of tardy jobs and resource allocation, Operations Research Letters 19 (5) (1996) 237–242.

[21] T. Cheng, Z.-L. Chen, C.-L. Li, B.-T. Lin, Scheduling to minimize the total compression and late costs, Naval Research Logistics (NRL) 45 (1) (1998) 67–82.

[22] L. Yedidsion, D. Shabtay, E. Korach, M. Kaspi, A bicriteria approach to minimize number of tardy jobs and resource consumption in scheduling a single machine, International journal of production economics 119 (2) (2009) 298–307.

[23] A. Janiak, M. Y. Kovalyov, Single machine scheduling subject to deadlines and resource dependent processing times, European journal of operational research 94 (2) (1996) 284–291.

[24] Z.-L. Chen, Q. Lu, G. Tang, Single machine scheduling with discretely controllable processing times, Operations Research Letters 21 (2) (1997) 69–76.

[25] Y. He, Q. Wei, T. Cheng, Single-machine scheduling with trade-off between number of tardy jobs and compression cost, Journal of Scheduling 10 (4) (2007) 303–310.

[26] B. Alidaee, A. Ahmadian, Two parallel machine sequencing problems involving controllable job processing times, European Journal of Operational Research 70 (3) (1993) 335–341.

[27] T. Cheng, Z. Chen, C.-L. Li, Parallel-machine scheduling with controllable processing times, IIE transactions 28 (2) (1996) 177–180.

[28] K. Jansen, M. Mastrolilli, Approximation schemes for parallel machine scheduling problems with controllable processing times, Computers & Operations Research 31 (10) (2004) 1565–1581.

[29] D. Shabtay, M. Kaspi, Parallel machine scheduling with a convex resource consumption function, European Journal of Operational Research 173 (1) (2006) 92–107.

[30] F. Zhang, G. Tang, Z.-L. Chen, A 32-approximation algorithm for parallel machine scheduling with controllable processing times, Operations Research Letters 29 (1) (2001) 41–47.

[31] M. Pinedo, Scheduling, Vol. 29, Springer, 2012.

[32] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. R. Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, in: Annals of discrete mathematics, Vol. 5, Elsevier, 1979, pp. 287–326.

[33] E. L. Lawler, J. K. Lenstra, A. H. R. Kan, D. B. Shmoys, Sequencing and scheduling: Algorithms and complexity, Handbooks in operations research and management science 4 (1993) 445–522.

[34] B. Chen, C. N. Potts, G. J. Woeginger, A review of machine scheduling: Complexity, algorithms and approximability, Handbook of combinatorial optimization (1998) 1493–1641.

[35] M. O. Adamu, A. O. Adewumi, A survey of single machine scheduling to minimize weighted number of tardy jobs, Journal of Industrial & Management Optimization 10 (1) (2014) 219.

[36] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz, Handbook on scheduling, Springer, 2019.

[37] J. M. Moore, An n job, one machine sequencing algorithm for minimizing the number of late jobs, Management science 15 (1) (1968) 102–109.

[38] M. O. Adamu, A. O. Adewumi, Minimizing the weighted number of tardy jobs on multiple machines: A review, Journal of Industrial & Management Optimization 12 (4) (2016) 1465.

[39] T. Pikies, K. Turowski, M. Kubale, Scheduling with complete multipartite incompatibility graph on parallel machines, in: Proceedings of the ICAPS, Vol. 31, 2021, pp. 262–270.

[40] R. G. Michael, S. David, Johnson: Computers and intractability: A guide to the theory of np-completeness (1979).

Appendix A. Proof of Theorem 1

Proof. The Subset Sum problem is reducible to our PCPT Problem. Let $\mathcal{A} = \{a_1, \ldots, a_n\}$ (integer set), and W (integer value) be a given instance of a subset sum. Let the integer B, where $B > a_i, \forall a_i \in \mathcal{A}$, be the value cap.

We construct a PCPT Problem as follows. We consider a special case of the standard two machines PCPT Problem. Let there be 2 machines M_1 and M_2 in our system. M_1 is a normal machine with different processing times for different processing levels (its processing times will be introduced later). M_2 is a dummy machine with infinite processing times for all processing levels, indicating that M_2 will never be used in our system. Let there be one job set $\mathcal{J} = \{0, 1, \ldots, J\}$. All jobs in $\mathcal{J}$ are independent, identical, and non-preemptive. The jobs are released with a fixed time interval $T = 1$. We set the relative deadline $T_{max} = +\infty$. We have $L = 2n$ processing levels to indicate whether one of an element $a_i \in \mathcal{A}$ has been selected or not. Please note, both the M_1 and M_2 have L processing levels, but all L processing times $\mathcal{P}_2 = \{p_{(2,1)}, \ldots, p_{(2,L)}\}$ on M_2 are set to be $+\infty$. For M_1, the processing time and the utility of the $(2i-1)$th (resp. $2i$th) processing level are both set to be $p_{(1,2i-1)} = u_{(2i-1)} = (2^{(n+1)} + 2^i) \cdot nB$ (resp. $p_{(1,2i)} = u_{(2i)} = (2^{(n+1)} + 2^i) \cdot nB + a_i$). Finally, the common deadline T_c is set to be $T_c = (n2^{(n+1)} + 2^n + \cdots + 2^1)nB + W$.

Suppose $\mathcal{A}' = \{a_{q_1}, \ldots, a_{q_k}\}, \mathcal{A}' \subseteq \mathcal{A}$ is a solution subset of the given subset sum instance, where $\sum_{a \in \mathcal{A}'} a = W$. Then, we schedule the first k jobs with the processing times $\{p_{(1,2q_1)}, \ldots, p_{(1,2q_k)}\}$ respectively; We schedule the next $n - k$ jobs with the processing times $\{p_{(1,2b_1-1)}, \ldots, p_{(1,2b_{(n-k)}-1)}\}$, where $\{b_1, \ldots, b_{(n-k)}\} = \{1, \ldots, n\} \setminus \{q_1, \ldots, q_k\}$. The first n jobs are released within times $[0, n]$. Whichever processing level the job 0 is processed at, by the time it ends, all these n jobs have arrived. Then, each of these n jobs selects one processing level $1 < l < L$ for itself without repetition, and the kth job is completed at time $T_c = (n2^{(n+1)} + 2^n + \cdots + 2^1)nB + W$. Thus, we achieve the highest possible overall utility, because there exists no idle time slot in $[0, T_c]$ on M_1 and M_2 is a dummy machine.

Conversely, if there is a schedule for the given PCPT instance that achieves the overall utility of $(n2^{(n+1)} + 2^n + \cdots + 2^1)nB + W$, then it is an optimal solution for the given subset sum problem. In this case, if there exists a job that is processed at a level $p_{(1,2i-1)}$, where $1 \leq i \leq n$, then we exclude the element $a_i \in \mathcal{A}$ in our subset sum solution; if there exists a job that is processed at a level $p_{(1,2i)}$, where $1 \leq i \leq n$, then we include the element $a_i \in \mathcal{A}$ in our subset sum solution. Therefore, no element $a_i \in \mathcal{A}$ is repeatedly selected in our solution, and the sum of the elements in our subset equals to the target value W.

The reduction from the subset sum problem to the PCPT Problem is completed. □

Appendix B. Proof of Theorem 2

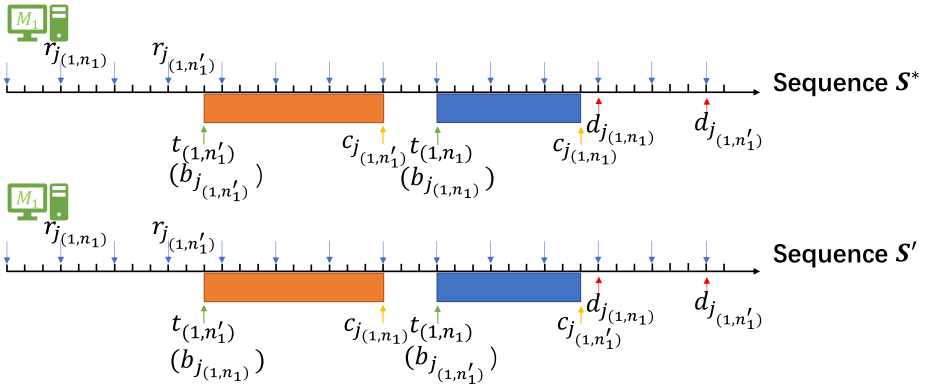


Figure B.6: Illustration of Theorem 2

Proof. Suppose there exists an optimal sequence S^* and it does not satisfy condition (1) or (2) in Theorem 2. Then, we have 3 possible cases:

① S^* satisfies condition (2), but not condition (1);

② S^* satisfies condition (1), but not condition (2);

③ S^* satisfies neither condition (1) nor condition (2).

For all cases ①–③, we aim to find an alternative sequence S' which has the same utility as S^* and satisfies both conditions (1)–(2) in Theorem 2.

Case ①: In this case, we aim to find an alternative sequence S' which has the same utility as S^* and satisfies condition (1) in Theorem 2.

As shown in Fig. B.6, we have an optimal sequence $S^* = (S_1^*, S_2^*)$ which satisfies: $\exists s_{(1,n_1)}, s_{(1,n_1')} \in S_1^*$, $n_1 \neq n_1'$, and $r_{j_{(1,n_1)}} < r_{j_{(1,n_1')}}$, but $n_1 > n_1'$. We aim to find an alternative sequence with $n_1 < n_1'$, which is equally optimal.

From the definition of PCPT Problem, we know the following facts: (a) an earlier arriving job has a non-later absolute deadline; (b) all jobs are non-preemptive; (c) the processing duration of any two jobs cannot be overlapped. Then, S^* satisfies $r_{j_{(1,n_1)}} < r_{j_{(1,n_1')}} \leq t_{(1,n_1')} < c_{j_{(1,n_1')}} \leq t_{(1,n_1)} < c_{j_{(1,n_1)}} \leq d_{j_{(1,n_1)}} < d_{j_{(1,n_1')}}$.

We can construct an alternatively optimal sequence S' by swapping the processing orders of $j_{(1,n_1)}$ and $j_{(1,n_1')}$ as well as their processing levels. We keep all other jobs unchanged. In this alternative sequence, $j_{(1,n_1)}$ will be processed in $[t_{(1,n_1')}, c_{j_{(1,n_1')}}]$, and $j_{(1,n_1')}$ will be processed in $[t_{(1,n_1)}, c_{j_{(1,n_1)}}]$. From S^*, we have $r_{j_{(1,n_1)}} < t_{(1,n_1')} < c_{j_{(1,n_1')}} < d_{j_{(1,n_1)}}$ and $r_{j_{(1,n_1')}} < t_{(1,n_1)} < c_{j_{(1,n_1)}} < d_{j_{(1,n_1')}}$.

All the completed jobs in S' are started after their release times and are completed before their absolute deadlines. The overall utility of S' is not changed, because the processing levels for $j_{(1,n_1)}$ and $j_{(1,n_1')}$ are not changed. Therefore, S' is an equally optimal sequence that satisfies both conditions (1)–(2) in Theorem 2.

Case ②: In this case, we prove that we can find an alternative sequence S' which has the same utility as S^*, but S' does not satisfy condition (2) in Theorem 2. The proof of this case is in a symmetrical manner as case ①. Such a sequence S' is an equally optimal sequence that satisfies both conditions (1)–(2) in Theorem 2.

Case ③: Following cases ①–②, we know that constructing an alternatively optimal sequence on one machine will not change the sequence on the other machine. Therefore, in case ③, we first follow case ① to find an equally optimal

sequence. Next, we follow case ② to obtain another equally optimal sequence S'. Then, S' is an equally optimal sequence that satisfies both conditions (1)–(2).

Consequently, considering all cases ①–③ above, we prove the fact that for any optimal sequence S^*, if it does not satisfy condition (1) or (2) in Theorem 2, then we can find an equally optimal sequence S' that satisfies both conditions (1)–(2) in Theorem 2.

□

Appendix C. Proof of Theorem 3

Proof. Our proof is completed through two stages. In the first stage, we prove the fact: If there exists an optimal sequence S^* that satisfies the OSS Property, but S^* does not follow one of the conditions (1.1) or (1.2) in Theorem 3, then we show that we can find an alternative sequence S'; this S' has the same utility as S^*; and it satisfies the OSS Property and both conditions (1.1)–(1.2) in Theorem 3. In the next stage, we prove the fact: If there exists a sequence S' that satisfies the OSS Property and both conditions (1.1)–(1.2) in Theorem 3, then S' will also satisfy both conditions (2.1)–(2.2) in Theorem 3.

Appendix C.1. Definition of Interchangeable Job

Before we start the first stage of the proof, we need to define an important concept. Given any sequence $S = (S_1, S_2)$, for any active operation entry $s_{(1,n_1)} \in S_1$, we may have a job k_0, satisfying the following conditions: k_0 is released before the job processed in $s_{(1,n_1)}$; k_0 is started on or after the start time of $s_{(1,n_1)}$; at $t_{(1,n_1)}$, k_0 can be possibly started by M_1 with the same processing level $l_{(1,n_1)}$ to meet its absolute deadline without breaching the OSS Property. Then, we call k_0 an interchangeable job of $s_{(1,n_1)}$ given S. Please note, k_0 may be a job that is started by M_2 on or after $t_{(1,n_1)}$ or a job that is rejected by both machines. However, it cannot be a job that is started by M_1 after $t_{(1,n_1)}$ due to the OSS Property. Given any sequence S, all the waiting operation entries have no interchangeable job, because they do not start any job. Let $E_1(s_{(1,n_1)}, S)$

denote the set that records all the interchangeable jobs of an operation entry $s_{(1,n_1)}$ given any sequence S, which is formally defined as below.

If $s_{(1,n_1)} = (t_{(1,n_1)}, O_{(j_{(1,n_1)},1,l_{(1,n_1)})})$,

$$E_1(s_{(1,n_1)}, S) = \big\{k | k \in \mathcal{J} \wedge r_k < r_{j_{(1,n_1)}} \leq t_{(1,n_1)} < t_{(1,n_1)} + p_{(1,l_{(1,n_1)})} \leq d_k \wedge b_k \geq t_{(1,n_1)} \wedge j < k, \forall j \in \{i | b_{i1} < t_{(1,n_1)}\}\big\}. \tag{C.1}$$

If $s_{(1,n_1)} = (t_{(1,n_1)}, W_{(1,t_{(1,n_1)})})$,

$$E_1(s_{(1,n_1)}, S) = \varnothing. \tag{C.2}$$

Please note, $E_1(s_{(1,n_1)}, S) = \varnothing$ represents that $s_{(1,n_1)}$ has no interchangeable job given S.

Given any sequence S, for any operation entry $s_{(2,n_2)}$, its interchangeable job set $E_2(s_{(2,n_2)}, S)$ can be defined symmetrically. Then, for any sequence S, the satisfaction of conditions (1.1)–(2.2) can be equivalent to the satisfaction of the following condition: $E_1(s_{(1,n_1)}, S) = \varnothing$ and $E_2(s_{(2,n_2)}, S) = \varnothing, \forall s_{(1,n_1)}, s_{(2,n_2)} \in S$. In the first stage of the proof, we will use this equivalent condition to replace conditions (1.1)–(1.2) in Theorem 3.

Appendix C.2. Proof of the Satisfaction of Conditions (1.1)–(1.2)

Suppose there exists an optimal sequence $S^* = (S_1^*, S_2^*)$ to PCPT Problem, satisfying the OSS Property and any of the following conditions: (a) $\exists s_{(1,n_1)} \in S^*$, where $E_1(s_{(1,n_1)}, S^*) \neq \varnothing$; (b) $\exists s_{(2,n_2)} \in S^*$, where $E_2(s_{(2,n_2)}, S^*) \neq \varnothing$. We aim to find an alternative sequence S'; this S' has the equal utility to the optimal sequence S^*; and S' satisfies the OSS Property and both conditions (a)–(b) (i.e., $E_1(s'_{(1,n_1)}, S') = \varnothing \wedge E_2(s'_{(2,n_2)}, S') = \varnothing, \forall s'_{(1,n_1)}, s'_{(2,n_2)} \in S'$).

In other words, if we have an optimal sequence S^*, but this S^* does not satisfy condition (1.1) or (1.2) in Theorem 3, then we prove that we can find an alternative sequence S'; this S' has the same utility as S^*; and S' satisfies the OSS Property and both conditions (1.1)–(1.2) in Theorem 3. We complete this proof by induction, as shown in Lemmas 4–5.

Lemma 4 (Base Case)**.** *For any sequence S, all operation entries started at time 0 have no interchangeable job, where $E_1(s_{(1,1)}, S) = \varnothing, E_2(s_{(2,1)}, S) = \varnothing$.*

Proof. First, both machines are decision makers at time 0. Then, we have $s_{(1,1)}$ and $s_{(2,1)}$, where $t_{(1,1)} = t_{(2,1)} = 0$. Only job 0 is released at time 0 in any sequence S. Obviously, given any sequence S, at time 0, there exists no active operation entry that has any interchangeable job.

□

Lemma 5 (Induction Step)**.** *For any sequence S, given time t ($0 \leq t < T_c - 1$) and given that all operations started on or before t in S have no interchangeable job, there exists an alternative sequence S' that satisfies the following conditions: (i) S' has the same utility as S; (ii) any jobs started on or before $t+1$ in S' are processed orderly on the same machines; (iii) any operation entry started on or before $t+1$ in S' has no interchangeable job.*

Proof. For an arbitrary sequence S, by induction, given that at t ($0 \leq t < T_c - 1$), all operations started on or before t in S have no interchangeable job (t-condition).

Following the definition of the system evolution, for any sequence, given a decision making time and a decision maker, we have one and only one operation entry. In S, at $t+1$, if M_1 (resp. M_2) is a decision maker, then we have $s_{(1,n_1)}$ (resp. $s_{(2,n_2)}$), where $s_{(1,n_1)} \in S_1, t_{(1,n_1)} = t+1$ (resp. $s_{(2,n_2)} \in S_2, t_{(2,n_2)} = t+1$). Then, at $t+1$ in S, we have 4 main cases with sub-cases as below.

① M_1 is the committed machine; M_2 is the committed machine.

② M_1 is the decision maker; M_2 is the committed machine:

- (a) $s_{(1,n_1)}$ has at least one interchangeable job, i.e., $E_1(s_{(1,n_1)}, S) \neq \varnothing$;
- (b) $s_{(1,n_1)}$ has no interchangeable job, i.e., $E_1(s_{(1,n_1)}, S) = \varnothing$.

③ M_1 is the committed machine; M_2 is the decision maker:

- (a) $s_{(2,n_2)}$ has at least one interchangeable job, i.e., $E_2(s_{(2,n_2)}, S) \neq \varnothing$;

(b) $s_{(2,n_2)}$ has no interchangeable job, i.e., $E_2(s_{(2,n_2)}, S) = \varnothing$.

④ Both of M_1 and M_2 are the decision makers:

(a) $s_{(1,n_1)}$ has at least one interchangeable job, i.e., $E_1(s_{(1,n_1)}, S) \neq \varnothing$;

(b) $s_{(1,n_1)}$ has no interchangeable job, and $s_{(2,n_2)}$ has at least one interchangeable job, i.e., $E_1(s_{(1,n_1)}, S) = \varnothing \wedge E_2(s_{(2,n_2)}, S) \neq \varnothing$;

(c) $s_{(1,n_1)}$ has no interchangeable job, and $s_{(2,n_2)}$ has no interchangeable job, i.e., $E_1(s_{(1,n_1)}, S) = \varnothing \wedge E_2(s_{(2,n_2)}, S) = \varnothing$.

Appendix C.2.1. Cases ①, ②(b), ③(b), and ④(c)

For all such cases, no operation entry started at $t + 1$ in S has any interchangeable job. For case ①, no operation entry is started at $t+1$ in S'. Following t-condition, for all cases ②(b), ③(b), and ④(c), any job that is started at $t+1$ on either of the machines is released after all the jobs that are started before it on the same machine. Otherwise, it will be an interchangeable job for an earlier started operation entry, contradicting with t-condition. We set $S' \Leftarrow S$. As a result, S' has the same utility as S and satisfies all conditions (i)–(iii).

Appendix C.2.2. Cases ②(a) *and* ④(a)

For these two cases, at $t + 1$, only $s_{(1,n_1)}$ has at least one interchangeable job. We swap its processed job $j_{(1,n_1)}$ with its earliest interchangeable job $k = \min E_1(s_{(1,n_1)}, S)$ to construct an alternative sequence S'. There are 3 possibilities for k.

A. $b_k = +\infty$: In this case, k is rejected by both machines. An alternative sequence S', which has the equal utility, can be achieved by processing k in time $[t_{(1,n_1)}, t_{(1,n_1)} + p_{(1,l_{(1,n_1)})}]$. We keep all other operations unchanged. From (C.1), we have $r_k < r_{j_{(1,n_1)}} \leq t_{(1,n_1)}$ indicating k has already been released before $t_{(1,n_1)}$. $b_k \geq t_{(1,n_1)}$ indicates k is not started by any machines before $t_{(1,n_1)}$. Then k can be started by M_1 at $t_{(1,n_1)}$. $t_{(1,n_1)} + p_{(1,l_{(1,n_1)})} \leq d_k$ indicates that starting k by M_1 at $t_{(1,n_1)}$ with the processing level $l_{(1,n_1)}$ will not miss its absolute deadline. We do

not change any processing level selection, and the overall utility does not change. As a result, S' has an equal utility to S.

B. $b_{k2} \in [t+1, T_c)$: In this case, w.l.o.g., let $s_{(2,n_2)}$ denote the operation entry that is started by M_2 at b_{k2}. We have $b_{k2} \geq t+1$. An alternative sequence S', which has an equal utility, can be achieved by swapping the processing orders of the two jobs $j_{(1,n_1)}$ and k. We keep all other jobs unchanged. Then, we process k in $[t_{(1,n_1)}, t_{(1,n_1)} + p_{(1,l_{(1,n_1)})}]$ on machine M_1 and process $j_{(1,n_1)}$ in $[b_{k2}, b_{k2}+p_{(2,l_{(2,n_2)})}]$ on machine M_2. In (C.1), we have $r_k < r_{j_{(1,n_1)}} \leq t_{(1,n_1)}$ indicating $j_{(1,n_1)}$ and k are released on or before $t_{(1,n_1)}$. We have $t_{(1,n_1)}+p_{(1,l_{(1,n_1)})} \leq d_k$ and $b_{k2}+p_{(2,l_{(2,n_2)})} \leq d_k < d_{j_{(1,n_1)}}$, indicating that swapping these two jobs will not breach their absolute deadlines. Since we do not change any processing levels, the overall utility does not change. Then, S' has an equal utility to S.

C. $b_{k1} \in (t+1, T_c)$: In this case, S does not satisfy the OSS Property, we can follow Theorem 2 to construct an equally optimal sequence S'.

For these 3 possibilities, we show that the alternative sequence S' has an equal utility to S, satisfying condition (i). Following t-condition, in S, k is not an interchangeable job for any operation entry that is started on or before t. It indicates that k is released after all the jobs that are started by M_1 on or before t in S. Then, at $t+1$, after we swap $j_{(1,n_1)}$ with its earliest interchangeable job k to construct S', all the jobs started by M_1 before $t+1$ are released earlier than k. Thus, in S', at $t+1$, the operation started by machine M_1 will have no interchangeable job.

If we are dealing with case ②(a), M_2 is the committed machine at $t+1$ in sequences S. Then, when we construct S', we do not change any operation entry started by M_2 on or before $t+1$. Then, in S', all operation entries started by M_2 on or before $t+1$ have no interchangeable job. Therefore, in S', all jobs started by M_2 on or before $t+1$ are processed orderly. As a result, S' is an alternative sequence that has the same utility as S and satisfies all conditions (i)–(iii).

Appendix C.2.3. Cases ③(a) *and* ④(b)

For these two cases, only $s_{(2,n_2)}$ in S has at least one interchangeable job. We follow the same swapping approach as we used in case ④(a) to construct an alternative sequence S' for both cases ③(a) and ④(b). As we discussed in case ④(a), the alternative sequence S' has the equal utility to S and satisfies all conditions (i)–(iii).

Then, we complete the proof of Lemma 5. □

Table C.2: The valid job conditions at two machines. Each tuple (σ_1, σ_2) represents the job conditions in M_1 and M_2. The characters `O,C,R,U,` and `I` represent the `OVERDUE`, `COMPLETED`, `RESIGNED`, `UNCOMMITTED`, and `IGNORED` respectively. "×" represents this condition is not possible. For example, (`O1`, `O2`) in this table is the abbreviation of the job condition (`OVERDUE 1, OVERDUE 2`).

$\sigma_1 \backslash \sigma_2$	O2	C2	R2	U2	I2
O1	(O1, O2)	(O1, C2)	(O1, R2)	(O1, U2)	(O1, I2)
C1	(C1, O2)	×	(C1, R2)	×	×
R1	(R1, O2)	(R1, C2)	×	×	×
U1	(U1, O2)	×	×	(U1, U2)	(U1, I2)
I1	(I1, O2)	×	×	(I1, U2)	(I1, I2)

Following Lemmas 4–5, we can start the induction from $t = 0$. Then, we can draw a conclusion that for any sequence S, there exists an alternative sequence S' that satisfies the following conditions: (i) S' has the same utility as S; (ii) any jobs started on or before time T_c in S' are processed orderly on the same machines; (iii) any operation entry started before T_c in S' has no interchangeable job.

The conclusion above is equivalent to the follows. If we have any sequence S, and S does not satisfy condition (1.1) or (1.2) in Theorem 3, then we can find an alternative sequence S' which has the equal utility to S and satisfies the OSS Property and both conditions (1.1)–(1.2) in Theorem 3. This final conclusion also holds for any optimal sequence S^*.

Consequently, we complete the proof of the satisfaction of conditions (1.1)–(1.2) in Theorem 3.

Appendix C.3. Proof of the Satisfaction of Conditions (2.1)–(2.2)

At this stage, we aim to prove that for any sequence S, if it satisfies the OSS Property and both conditions (1.1)–(1.2) in Theorem 3, then both conditions (2.1)–(2.2) in Theorem 3 will also hold for S. This proof is by contradiction.

Suppose there exists a sequence S to PCPT Problem, satisfying the OSS Property and both conditions (1.1)–(1.2) in Theorem 3, but S does not satisfy one of the conditions (2.1) or (2.2). Then, we have 2 possibilities as below.

① S does not satisfy (2.1).

② S does not satisfy (2.2).

We aim to prove that both cases ①–② lead to a contradiction.

Appendix C.3.1. Case ①

In this case, S satisfies the OSS Property and both conditions (1.1)–(1.2) in Theorem 3; and S has an anchor time pair (τ_1, τ_2) $(\tau_1 \leq \tau_2)$ such that there exists a job j_0 that is under `UNCOMMITTED 1` in M_1 given this (τ_1, τ_2); but at (τ_1, τ_2), there exists N jobs $j_1 < \cdots < j_N$, satisfying $j_0 < j_1, N \geq 2$, and $j_1, \cdots j_N$ are under `RESIGNED 1` in M_1. Without loss of generality, we consider two arbitrary jobs $k < i$, where $j_1 \leq k < i \leq j_N$, and both k and i are under `RESIGNED 1` in M_1.

As shown in Table C.2, for any job j that is under `UNCOMMITTED 1` in M_1, j can only be in a job condition `(U1,O2)`, `(U1,U2)`, or `(U1,I2)`. For any job j that is under `RESIGNED 1` in M_1, j can only be in a job condition `(R1,O2)` or `(R1,C2)`.

In case ①, given j is under `UNCOMMITTED 1` in M_1, j can only be under $(\texttt{U1}, \texttt{I2})$ or $(\texttt{U1}, \texttt{O2})$. We do not consider the case that j_0 is under $(\texttt{U1}, \texttt{U2})$. Because both k and i are released after j_0, and they are committed to M_2 at (τ_1, τ_2). Then, j can only be under `IGNORED 2` or `OVERDUE 2`. We also know that both k and i are under `RESIGNED 1` in M_1. As shown in Table C.2, both k and i can only be under $(\texttt{R1}, \texttt{C2})$ or $(\texttt{R1}, \texttt{O2})$.

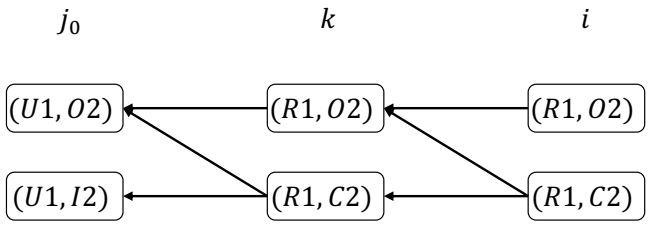


Figure C.7: The possible job conditions for j_0, k, and i. The arrows from one job to another indicate the possible job conditions for an earlier released job given the job condition of a later released job. For example, j_0 : `(U1,O2)` $\leftarrow k$: `(R1,C2)` and j_0 : `(U1,I2)` $\leftarrow k$: `(R1,C2)` indicate that j_0 can be only under `(U1,O2)` or `(U1,I2)` given k is under `(R1,C2)`.

Next, as shown in Fig. C.7, we consider the combinations of the job states of j_0, k, and i. If j_0 is under $(\texttt{U1}, \texttt{I2})$, both k and i cannot be under $(\texttt{R1}, \texttt{O2})$. This is because of $j_0 < k < i$ indicating that the absolute deadline for j_0 is earlier than both k and i. Thus, if j_0 is under $(\texttt{U1}, \texttt{I2})$, then we only need to consider the case that both k and i are under $(\texttt{R1}, \texttt{C2})$. If j_0 is under $(\texttt{U1}, \texttt{O2})$, then k can be under $(\texttt{R1}, \texttt{C2})$ or $(\texttt{R1}, \texttt{O2})$. Furthermore, if k is under $(\texttt{R1}, \texttt{C2})$, then i cannot be under $(\texttt{R1}, \texttt{O2})$. This is because the absolute deadline for k is earlier than i. Otherwise, if k is under $(\texttt{R1}, \texttt{O2})$, then i can be under $(\texttt{R1}, \texttt{C2})$ or $(\texttt{R1}, \texttt{O2})$.

In total, we have 4 situations for the state combinations of j_0, k, and i:

A. j_0: $(\texttt{U1}, \texttt{I2})$; k: $(\texttt{R1}, \texttt{C2})$; i: $(\texttt{R1}, \texttt{C2})$;

B. j_0: $(\texttt{U1}, \texttt{O2})$; k: $(\texttt{R1}, \texttt{C2})$; i: $(\texttt{R1}, \texttt{C2})$;

C. j_0: $(\texttt{U1}, \texttt{O2})$; k: $(\texttt{R1}, \texttt{O2})$; i: $(\texttt{R1}, \texttt{C2})$;

D. j_0: $(\texttt{U1}, \texttt{O2})$; k: $(\texttt{R1}, \texttt{O2})$; i: $(\texttt{R1}, \texttt{O2})$.

W.l.o.g, let $(\tau_{(1,n_1)}, \tau_{(2,n_2)}) = (\tau_1, \tau_2)$ $(\tau_{(1,n_1)} \leq \tau_{(2,n_2)})$ denote the given anchor time pair. Under all 4 situations above, we have 4 implications as below. (i) k is under `RESIGNED 1` in M_1 at $(\tau_{(1,n_1)}, \tau_{(2,n_2)})$. From conditions (1.1)–(1.2) in Theorem 3, k is the earliest released job that can be possibly started by M_2 at b_{k2}, and can be completed at c_{k2} without breaching its absolute deadline. In other words, the absolute deadline of j_0 is earlier than c_{k2}, implying $d_{j_0} < c_{k2}$. (ii) At $(\tau_{(1,n_1)}, \tau_{(2,n_2)})$ of S, we have $\tau_{(2,n_2-1)} < \tau_{(1,n_1)} \leq \tau_{(2,n_2)}$ due to the definition of the system evolution. Then, the last operation started by M_2

is started earlier than $\tau_{(1,n_1)}$. We have i in `RESIGNED 1` under all situations, indicating that i is completed by M_2 at $(\tau_{(1,n_1)}, \tau_{(2,n_2)})$. Then, even if i is the last job that is started by M_2 at $(\tau_{(1,n_1)}, \tau_{(2,n_2)})$, i will be started by M_2 before the decision making time $\tau_{(1,n_1)}$, implying $b_{i2} < \tau_{(1,n_1)}$.
(iii) k is under `RESIGNED 1` in M_1, indicating that k is completed by M_2 at $(\tau_{(1,n_1)}, \tau_{(2,n_2)})$. Due to the OSS Property, k is completed before i, implying $c_{k2} \leq b_{i2}$.
(iv) j_0 is under `UNCOMMITTED 1` in M_1 at $(\tau_{(1,n_1)}, \tau_{(2,n_2)})$, implying $\tau_{(1,n_1)} \leq d_{j_0}$.

Then, taking (ii) and (iii) into (iv), we have $c_{k2} \leq b_{i2} < \tau_{(1,n_1)} \leq d_{j_0}$. However, in (i), we have $d_{j_0} < c_{k2}$. These two inequalities contradict each other. As a result, we prove that case ① does not hold.

Appendix C.3.2. Case ②

This case follows a symmetrical property as case ①, leading to a contradiction.

In summary, following cases ①–②, we prove that there exists no sequence S in PCPT Problem, satisfying the OSS Property and both conditions (1.1)–(1.2) in Theorem 3, and it does not satisfy one of the conditions (2.1) or (2.2).

It is equivalent to the conclusion that for all sequence S to PCPT Problem, if it satisfies the OSS Property and both conditions (1.1)–(1.2) in Theorem 3, then both conditions (2.1)–(2.2) in Theorem 3 will also hold for such a S.

Consequently, we complete the proof of Theorem 3.

□