Discrete Event Dyn Syst (2010) 20:37–62
DOI 10.1007/s10626-008-0052-5

Optimal Admission Control of Discrete Event Systems with Real-Time Constraints

Jianfeng Mao · Christos G. Cassandras

Received: 16 January 2008 / Accepted: 30 September 2008 / Published online: 23 October 2008

Abstract The problem of optimally controlling the processing rate of tasks in Discrete Event Systems with hard real-time constraints has been addressed in prior work under the assumption that a feasible solution exists. Since this cannot generally be the case, we introduce in this paper an admission control scheme in which some tasks are removed with the objective of maximizing the number of remaining tasks which are all guaranteed feasibility. We derive several optimality properties based on which we develop a computationally efficient algorithm for solving this admission control problem under certain conditions. Moreover, when no future task information is available, we derive necessary and sufficient conditions under which idling is optimal and define a metric for evaluating when and how long it is optimal to idle. Numerical examples are included to illustrate our results.

Keywords Discrete event system · Admission control · Real-time constraints

1 Introduction

A large class of Discrete Event Systems (DES) involves the control of resources allocated to tasks according to certain operating specifications. The basic modeling

The authors' work is supported in part by the National Science Foundation under Grants DMI-0330171 and EFRI-0735974, by AFOSR under grants FA9550-04-1-0133 and FA9550-04-1-0208, and by DOE under grant DE-FG52-06NA27490.

J. Mao (✉) · C. G. Cassandras
Division of Systems Engineering and Center for Information
and Systems Engineering, Boston University,
Brookline, MA 02446, USA
e-mail: jfmao@bu.edu

C. G. Cassandras
e-mail: cgc@bu.edu

block for such DES is a single-server queueing system operating on a first-come-first-served basis, whose dynamics are given by the well-known max-plus equation

$$x_i = \max(x_{i-1}, a_i) + s_i(u_i)$$

where a_i is the arrival time of task $i = 1, 2, \ldots$; x_i is the time when task i completes service; and $s_i(u_i)$ is its processing time which may be controllable through u_i. Examples arise in manufacturing systems, where the operating speed of a machine can be controlled to trade off between energy costs and requirements on timely job completion (Pepyne and Cassandras 2000); in computer systems, where the CPU speed can be controlled to ensure that certain tasks meet specified execution deadlines (Buttazzo 1997; Liu 2000); and in wireless networks where severe battery limitations call for new techniques aimed at maximizing the lifetime of such a network (Gamal et al. 2002; Miao and Cassandras 2006). In what follows, we concentrate on the control u_i being the processing rate of tasks and set $u_i = 1/\tau_i$ where τ_i is the processing time per operation in a task. If a task consists of μ_i operations, then we have $s_i(u_i) = \mu_i \tau_i$ and we can rewrite the equation above as

$$x_i = \max(x_{i-1}, a_i) + \mu_i \tau_i$$

A particularly interesting class of problems arises when such systems are subject to *real-time constraints*, i.e., $x_i \leq d_i$ for each task i with a given "deadline" d_i. In order to meet such constraints, one typically has to incur a higher cost associated with control τ_i. Thus, in a broader context, we are interested in studying optimization problems of the form:

$$\begin{aligned} &\min_{\tau_1,\ldots,\tau_N} \sum_{i=1}^{N} \mu_i \theta(\tau_i) \\ \text{s.t.} \quad &x_i = \max(x_{i-1}, a_i) + \mu_i \tau_i \leq d_i, \quad i = 1, \ldots, N; \\ &\tau_i \geq \tau_{\min}, \quad i = 1, \ldots, N. \end{aligned} \tag{1}$$

where $\theta(\tau_i)$ is a given cost function, $\tau_{\min}$ is the minimal processing time per operation, and all a_i, d_i are known. Such problems have been studied for preemptive tasks (Yao et al. 1995; Aydin et al. 2004), nonpreemptive periodic tasks (Jeffay et al. 1991; Jonsson et al. 1999), and nonpreemptive aperiodic tasks (Gamal et al. 2002; Miao and Cassandras 2006; Mao et al. 2007). The latter case is of particular interest in wireless communications where nonpreemptive scheduling is necessary to execute aperiodic packet transmission tasks which also happen to be highly energy-intensive; in such cases, the cost function in Eq. 1 represents the energy required for a packet transmission. One of the key challenges in dealing with Eq. 1 is to develop computationally efficient solution approaches that can be used in real-time settings and can be implemented in wireless devices with very limited computational power.

In prior work (Mao et al. 2007), we have shown that exploiting structural properties of the optimal state trajectory in Eq. 1 leads to a highly efficient (scalable) *Critical Task Decomposition Algorithm* (CTDA) for obtaining a solution *as long as the problem is feasible*. This leaves open the question of dealing with the case where feasibility does not hold, which may often arise in practice. For example, tasks may arrive in a bursty fashion causing a temporary system overload. When this happens, some tasks will violate their real-time constraints even if all tasks are processed with

the minimal cycle time, $\tau_{\min}$ (i.e., the maximal rate $1/\tau_{\min}$). Therefore, some tasks have to be sacrificed in order to allow the remaining ones to meet their deadlines. Often, a subset of tasks may be dropped without adversely affecting the overall system operation (we shall refer to these tasks as "removable" in the rest of the paper), so that we have an opportunity to render the problem feasible by selectively rejecting some such tasks. This naturally leads to an *admission control* problem where the objective becomes to ensure that as many tasks as possible meet their deadlines by rejecting some removable tasks.

To the best of our knowledge, work related to this admission control problem falls into two categories. First, from the viewpoint of real-time systems, a similar problem is formulated and studied in (Chetto and Chetto 1989; Tia et al. 1994; Silly et al. 1990; Schwan and Zhou 1992), often referred to as Optimal Acceptance Test (OAT). The motivation in this problem is similar to ours, i.e., to maximize the number of accepted tasks. There are, however, two key differences: (*i*) The application environment in OAT problems involves systems that are preemptive and adopting an Earliest Deadline First (EDF) scheduling policy; in contrast, our focus is on resource-limited systems where non-preemptive and First In First Out (FIFO) scheduling policies are preferred and often required (e.g., in order to maintain the order of packet transmissions). Note that these two different systems only coincide when all arrival times and deadlines are "compatible", i.e., $a_i < a_j \Rightarrow d_i \leq d_j$ and $d_i < d_j \Rightarrow a_i \leq a_j$, in which case FIFO is the same as EDF and a non-preemptive policy is optimal. (*ii*) The control options are different: in OAT problems, control is restricted to accepting a newly arriving task or not and all previously accepted tasks are committed to be processed in the future even if they are still unprocessed; in contrast, our approach is to allow the rejection of previously accepted but still unprocessed tasks as long as they are removable. Obviously, this control option is more flexible and generally results in a better solution as shown in our detailed analysis (see the example in Section 3 illustrating this fact).

Second, from the viewpoint of scheduling theory in the Operations Research literature the objective is to minimize the number of tardy (late) tasks. It can be easily seen that minimizing the number of tardy tasks implies maximizing the number of tasks that can meet their deadlines, which is the same as maximizing the number of accepted tasks in our problem because only those tasks meeting their deadlines will be accepted in our system. This setting also involves a non-preemptive single-server system. The first such scheduling problem studied is the well-known Moore's problem (Moore 1968) with a single machine and all task arrival times set to 0 (expressed through the shorthand notation $1|\cdot|\sum U_j$, where U_j is the indicator function selecting tardy jobs and $\sum U_j$ is the objective function). This problem can be solved in $O(n \log n)$. Later, a more general scheduling problem was studied with a single machine but different task arrival times (expressed as $1|r_j|\sum U_j$, where r_j means that task arrival times are arbitrary), which is proved to be $\mathcal{NP}$-hard (Lenstra et al. 1977). Then, a polynomially solvable case was found and solved in $O(n^2)$ when all task arrival times and deadlines are compatible (as defined earlier) (Kise et al. 1978). In recent years, several attempts were made to solve the more general ($1|r_j|\sum U_j$) problem. An efficient mixed integer programming problem was formulated in Dauzère-Pérès and Sevaux (2002) by introducing a "master sequence" and a branch and bound method is provided in Dauzère-Pérès and Sevaux (2004) to derive the exact optimal solution. There are two key differences between our

problem and this scheduling problem: (*i*) We adopt a FIFO processing order because, as already pointed out, we are interested in resource-limited systems, whereas the processing order is not fixed in the general scheduling problem $(1|r_j|\sum U_j)$. The two problems do share the same optimal solution in the compatible case mentioned above because the optimal processing order in this case is FIFO. Although the problem is still $\mathcal{NP}$-hard even if we fix the processing order to be FIFO, the system is simplified and there is an opportunity to derive efficient solutions under certain conditions. (*ii*) The scheduling problem $(1|r_j|\sum U_j)$ assumes that any task can miss its deadlines, i.e., that *all* tasks are removable; this is only a special case (which we previously considered in Mao and Cassandras 2007) of the more general problem we are interested in.

The contribution of this paper is to formulate the admission control problem described above, establish sufficient conditions which we prove can be exploited to derive an optimal solution, and develop an efficient procedure termed *Maximal Shift Task Algorithm* (MSTA) which can be used to obtain such a solution. The MSTA may be used off line when all task information is known or on line at each task departure event based on the task information currently available. In addition, in the on-line version of the problem where task arrival information is not known in advance we show that under certain necessary and sufficient conditions *idling is optimal* (in the sense of maximizing the number of tasks that can be processed without violating their deadlines) even though this may be counterintuitive. We propose a metric based on distributional information characterizing task arrivals and develop an algorithm for solving the on-line version of Eq. 1.

In Section 2 of the paper we formulate the admission control problem associated with Eq. 1. In Section 3, we exploit some optimality properties leading to the aforementioned MSTA. We consider the on-line admission control problem in Section 4, provide numerical examples in Section 5, and conclude with Section 6.

2 Problem formulation

In this section, we will formulate the admission control problem whose solution will maximize the number of accepted tasks leading to a feasible optimization problem (1). First, however, we address the question of whether we can easily determine if problem (1) is indeed feasible or not.

Lemma 2.1 *Problem* (1) *is feasible if and only if* $\tau_i = \tau_{\min}$ *for* $i = 1, ..., N$ *is a feasible solution.*

Proof We first assume $\tau_i = \tau_{\min}$ for $i = 1, ..., n$ is a feasible solution of problem (1), which immediately implies that problem (1) is feasible.

We then assume Problem (1) is feasible, which implies that there exists a feasible solution τ_i' for $i = 1, ..., n$ such that $\tau_i' \geq \tau_{\min}$ and its corresponding $x_i' \leq d_i$ for all $i = 1, ..., N$. Let x_i denote the departure time of task i when the solution is $\tau_i = \tau_{\min}$ for $i = 1, ..., n$. Since $\tau_i' \geq \tau_{\min} = \tau_i$ for $i = 1, ..., N$, we have $x_i' \geq x_i$ for $i = 1, ..., N$. Combining the two inequalities $x_i' \leq d_i$ and $x_i' \geq x_i$, we have $x_i \leq d_i$ for all $i = 1, ..., N$, which implies that $\tau_i = \tau_{\min}$ for $i = 1, ..., N$ is also feasible. □

Using Lemma 2.1, it becomes easy to check whether $x_i \le d_i$ for all $i = 1, ..., N$ when $\tau_i = \tau_{\min}$ and, therefore, whether an admission control process is in fact needed. In what follows, without loss of generality, we assume $a_i \ge 0$ for $i = 1, ..., N$ so as to avoid unnecessary technical complications that may be introduced by allowing negative event times. We then formulate the *Admission Control Problem* (ACP) as follows by making use of Lemma 2.1:

$$\max_{z_1,...,z_N} \sum_{i\in\mathcal{R}} z_i \tag{2}$$

$$\text{s.t.} \quad x_i = \max(x_{i-1}, a_i z_i) + \mu_i \tau_{\min} z_i, \ i = 1, ..., N; \tag{3}$$

$$z_i(x_i - d_i) \le 0, \ i = 1, ..., N; x_0 = 0; \tag{4}$$

$$z_i \in \{0, 1\}, i \in \mathcal{R}; \quad z_i = 1, i \notin \mathcal{R}. \tag{5}$$

where in Eqs. 2 and 5, $\mathcal{R}$ denotes the set of removable tasks and $z_i = 1$ means that the ith task is accepted, otherwise it is rejected. If $z_i = 1$, then Eqs. 3 and 4 reduce to the same dynamics and real-time constraints as in Eq. 1 respectively. On the other hand, if $z_i = 0$ then task i will never occupy the server and its associated real-time constraint can be overlooked, in which case we have $x_i = x_{i-1}$. Note that the ACP also faces a feasibility question similar to Eq. 1, which is addressed in the following lemma.

Lemma 2.2 *The ACP is feasible if and only if its solution* $z_i = 0$ *for all* $i \in \mathcal{R}$ *and* $z_i = 1$ *for all* $i \notin \mathcal{R}$ *is feasible.*

Proof Similar to the proof of Lemma 2.1. □

Using Lemma 2.2, we see that the feasibility of the ACP depends on $\mathcal{R}$, which is uncontrollable since $\mathcal{R}$ is defined based on a given system design. If the ACP is infeasible when $\mathcal{R}$ is too small (in the extreme, $\mathcal{R} = \emptyset$), then admission control can serve no purpose and the system needs to be redesigned. In this paper, we assume that all non-removable tasks can meet their deadlines by dropping all removable tasks, i.e., the feasibility condition of the ACP shown in Lemma 2.2 holds.

Clearly, the ACP is a 0–1 integer programming problem and can be solved (in principle) through standard numerical methods. However, we emphasize again that we are interested in real-time applications involving resource-limited systems where such methods are both overly time consuming and not computationally affordable. In what follows, we develop an efficient solution algorithm for the ACP by utilizing the optimality properties presented in the next section.

3 Optimality properties

Before we formally study the optimality properties of the ACP, let us recall the *Optimal Acceptance Test* (OAT) approach (Chetto and Chetto 1989; Tia et al. 1994; Silly et al. 1990; Schwan and Zhou 1992) and consider an example as shown in Table 1, where $\tau_{\min} = 1$ and all six tasks are removable: The OAT is applied under EDF in a preemptive setting. However, since in this example all arrival times

Table 1 An example

i	1	2	3	4	5	6
a_i	0	0.1	0.2	0.3	0.4	0.5
d_i	2	10	10.1	10.2	10.3	10.4
μ_i	1	8	2	2	2	2

and deadlines are compatible, the OAT is actually implementable in a FIFO and nonpreemptive way. The OAT is applied at each task arrival time and its policy for this compatible example reduces to the following: if the new arrival task can meet its deadline, then it will be accepted; otherwise it is rejected.

Implementing the OAT for this example, we can easily see that the corresponding solution will keep the first two tasks and drop all the rest, i.e., $\sum_{i=1}^{6} z_i = 2$. However, we can clearly do much better if we do not restrict control to a newly arriving task. Instead, suppose we still make admission decisions at each task arrival time and can drop not only the newly arriving task but also all unprocessed tasks already accepted. Then, when the third task arrives (at which time the first task is still in process and the second task is in queue), we drop the second task. With this approach, we ultimately obtain a solution in which we keep all tasks except the second one. This results in $\sum_{i=1}^{6} z_i = 5$, obviously a much better solution than the one obtained through the OAT. At the same time, we realize that the price to pay is the increased computational complexity imposed by the fact that we need to solve the ACP at each decision point. Therefore, to make this improvement beneficial, we need to develop a very efficient algorithm for solving the ACP. This is indeed possible and relies on the optimality properties explored in the following sections.

3.1 Busy period and first infeasible task

Let $\mathcal{X}_i$ denote the departure time of task i in the ACP when $z_i = 1$ for all $i = 1, ..., N$, that is,

$$\mathcal{X}_i = \max(\mathcal{X}_{i-1}, a_i) + \mu_i \tau_{\min}, \text{ for } \ i = 1, ..., N. \tag{6}$$

Obviously, from Eq. 3, for any solution of the ACP, the corresponding departure times x_i must satisfy

$$x_i \leq \mathcal{X}_i, \quad \text{for } i = 1, ..., N \tag{7}$$

Definition 3.1 A *Busy Period* (BP) is a set of contiguous tasks $\{k, ..., n\}$, such that $\mathcal{X}_{k-1} \leq a_k$, $\mathcal{X}_n \leq a_{n+1}$ and $\mathcal{X}_i > a_{i+1}$ for $i = k, ..., n-1$.

Based on this definition, the state trajectory defined by $\{\mathcal{X}_i\}$, $i = 1, ..., N$, is decomposed into a set of BPs that define a *BP structure*. Clearly, the BP structure can be uniquely determined by a_i and μ_i for $i = 1, ..., N$ since $\mathcal{X}_i$ is immediately obtained through Eq. 6.

Definition 3.2 An *Infeasible Task* (IT) is a task i such that $\mathcal{X}_i > d_i$. A *First Infeasible Task* (FIT) within a BP $\{k, ..., n\}$ is a task m such that $\mathcal{X}_m > d_m$, $\mathcal{X}_i \leq d_i$ for $i = k, ..., m-1$ and task m belongs to the BP $\{k, ..., n\}$.

An example of a BP, IT and FIT is given in Fig. 1. It is of course possible that a FIT may not exist in a BP, in which case the BP need not be considered in what follows since all tasks in it can meet their deadlines and BPs are decoupled from each other as will be established in Lemma 3.2. We begin with the following lemma to identify a useful property of any FIT.

Lemma 3.1 *If m is a FIT belonging to the BP $\{k, ..., n\}$, then, for any feasible solution of the ACP, there exists at least one task $j \in \mathcal{R}$, $k \leq j \leq m$, such that $z_j = 0$.*

Proof Assume on the contrary that there exists some feasible solution such that $z_j = 1$ for all $j = k, ..., m$. From the definition of BP, we have $\mathcal{X}_{k-1} \leq a_k$. From Eqs. 3 and 7, we have $x_{k-1} \leq \mathcal{X}_{k-1}$, which implies that $x_{k-1} \leq a_k$. Combining this with the assumption that $z_j = 1$ for all $j = k, ..., m$ and Eq. 3, we have $x_m = \mathcal{X}_m$. Since m is a FIT, we have $x_m = \mathcal{X}_m > d_m$ which violates the constraint of $z_m(x_m - d_m) \leq 0$ in Eq. 4 and contradicts the feasibility assumption. □

Furthermore, the FIFO processing order facilitates the decomposition of the ACP into several smaller problems corresponding to BPs.

Lemma 3.2 *If FIFO is adopted, i.e., $a_1 \leq \cdots \leq a_N$, then BPs are decoupled from each other.*

Proof Assume $\{k, ..., n\}$ is an arbitrary BP. First, we show that all constraints related to $z_k, ..., z_n$ can be satisfied independent of $z_1, ..., z_{k-1}$ and of $z_{n+1}, ..., z_N$. Obviously, the constraints in Eqs. 3 and 5 related to $z_k, ..., z_n$ can be satisfied independent of $z_1, ..., z_{k-1}$ and $z_{n+1}, ..., z_N$. In the following, we only focus on the constraints in Eq. 4.

Without loss of generality, assume $k \leq i_1 \leq ... \leq i_p \leq n$, $z_j = 1$ for $j = i_1, ..., i_p$ and $z_j = 0$ for $j \neq i_1, ..., i_p$. Since $z_j = 0$ for $j \neq i_1, ..., i_p$, the constraint $z_j(x_j - d_j) \leq 0$ can be satisfied independent of $z_1, ..., z_{k-1}$ and $z_{n+1}, ..., z_N$ for all $j \neq i_1, ..., i_p$. This leaves only $z_{i_1}, ..., z_{i_p}$ to consider next.

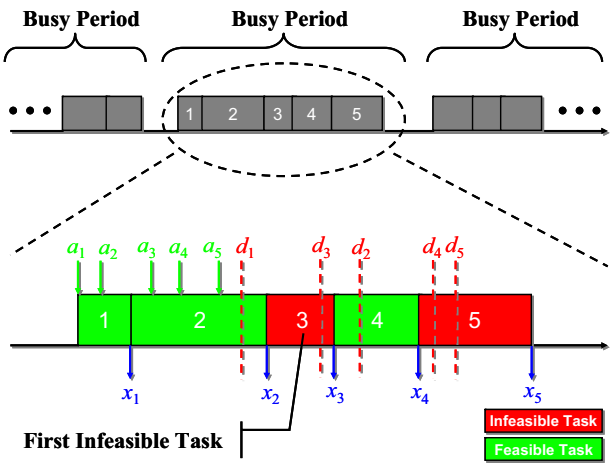


Fig. 1 Example of a busy period, infeasible task and first infeasible task

From the definition of BP, we have $\mathcal{X}_{k-1} \leq a_k$. From Eqs. 3 and 7, we have $x_{k-1} \leq \mathcal{X}_{k-1}$. Since FIFO is adopted, it follows that $x_{k-1} \leq \mathcal{X}_{k-1} \leq a_k \leq a_{i_1} \leq ... \leq a_{i_p}$. Combining this with Eq. 3, we have

$$x_{i_1} = \max(a_{i_1}, x_{k-1}) + \mu_{i_j}\tau_{\min} = a_{i_1} + \mu_{i_j}\tau_{\min};$$
$$x_{i_j} = \max(a_{i_j}, x_{i_{j-1}}) + \mu_{i_j}\tau_{\min}, \quad j = 2, ..., p.$$

which implies that x_j is independent of $z_1, ..., z_{k-1}$ and of $z_{n+1}, ..., z_N$ for $j = i_1, ..., i_p$. Thus, the constraint $z_j(x_j - d_j) \leq 0$ can be satisfied independent of $z_1, ..., z_{k-1}$ and of $z_{n+1}, ..., z_N$ for $j = i_1, ..., i_p$. We have, therefore, established that tasks $k, ..., n$ satisfy their corresponding constraints independent of $z_1, ..., z_{k-1}$ and of $z_{n+1}, ..., z_N$. Then, since the cost function (2) is the summation of $z_1, ..., z_N$, the cost related to $z_k, ..., z_n$ can also be separated from $z_1, ..., z_{k-1}$ and $z_{n+1}, ..., z_N$. It follows that BPs are decoupled from each other. □

Based on Lemma 3.2, the ACP can be decomposed into a number of smaller problems and we need only focus on finding a solution for each single BP $\{k, ..., n\}$ instead of the full ACP that involves $\{1, ..., N\}$. In the following section, we will introduce another concept based on BPs, termed a *Maximal Shift Task*, which leads to the key theorem.

3.2 Maximal shift task

In this section, we focus on a BP $\{k, ..., n\}$ which has a FIT m. Let $\mathcal{X}_i^j$ denote the departure time of task i when *only* task j is dropped, and let $\mathcal{S}_i^j$ denote the corresponding departure time shift of task i resulting from the removal of *only* task j, that is,

$$\mathcal{S}_i^j = \mathcal{X}_i - \mathcal{X}_i^j \tag{8}$$

We then define a Maximal Shift Task as follows:

Definition 3.3 A *Maximal Shift Task* (MST) relative to a FIT m within a BP $\{k, ..., n\}$ is a task r, $k \leq r \leq m$, such that

$$\mathcal{S}_m^r \geq \mathcal{S}_m^j, \quad \forall\ j \in \mathcal{R} \text{ and } k \leq j \leq r-1; \tag{9}$$

$$\mathcal{S}_m^r > \mathcal{S}_m^j, \quad \forall\ j \in \mathcal{R} \text{ and } r+1 \leq j \leq m. \tag{10}$$

An example of a MST is shown in Fig. 2, where task $m = 3$ is the FIT and task $r = 2$ is the MST relative to task 3.

Next, we derive several lemmas regarding a MST before obtaining the main theorem. We will make use of the following convenient notation:

$$\mathcal{C}_i^j = \min_{l=j+1,...,i} (\mathcal{X}_{l-1} - a_l), \quad j < i \text{ and } \mathcal{C}_i^j = +\infty, \quad i = j \tag{11}$$

where we can see that $\mathcal{C}_i^j$ is the smallest waiting time experienced by tasks $j+1, ..., i$.

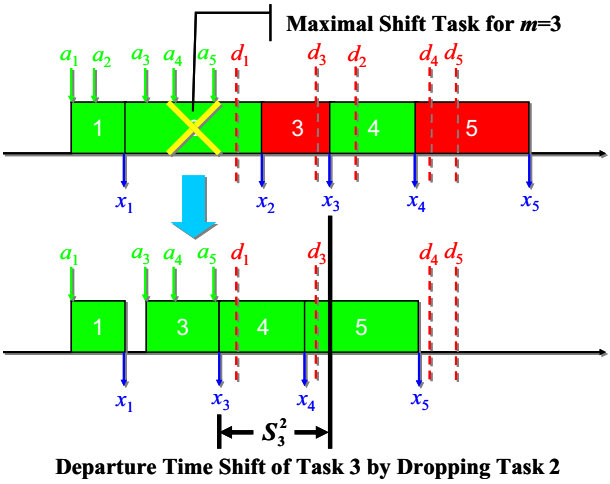


Fig. 2 Example of a departure time shift and a MST $r = 2$ relative to the FIT $m = 3$

Lemma 3.3 *For any i, j such that* $j \leq i$ *and both tasks i, j belong to the same BP,*

$$\mathcal{S}_i^j = \min\left(\mathcal{C}_i^j, \mathcal{X}_j - \mathcal{X}_{j-1}\right) \tag{12}$$

Proof Since $j \leq i$ and both tasks i, j belong to a same BP, we have

$$\mathcal{X}_l = \mathcal{X}_{l-1} + \mu_l \tau_{\min}, \quad j+1 \leq l \leq i \tag{13}$$

By the definition of $\mathcal{X}_i^j$, we have

$$\mathcal{X}_j^j = \mathcal{X}_{j-1} \tag{14}$$

$$\mathcal{X}_l^j = \max\left(a_l, \mathcal{X}_{l-1}^j\right) + \mu_l \tau_{\min}, \quad j+1 \leq l \leq i \tag{15}$$

It follows from Eqs. 13 and 14 that

$$\mathcal{X}_j - \mathcal{X}_j^j = \mathcal{X}_j - \mathcal{X}_{j-1}$$

and from Eqs. 13 and 15 we get

$$\mathcal{X}_l - \mathcal{X}_l^j = \min\left(\mathcal{X}_{l-1} - a_l, \mathcal{X}_{l-1} - \mathcal{X}_{l-1}^j\right), \quad j+1 \leq l \leq i$$

Combining two equations above, we have

$$\mathcal{X}_i - \mathcal{X}_i^j = \min(\mathcal{X}_{i-1} - a_i, ..., \mathcal{X}_j - a_{j+1}, \mathcal{X}_j - \mathcal{X}_{j-1})$$

which implies Eq. 12 by using Eqs. 8 and 11. □

Lemma 3.4 *For any i, j such that* $j < i$ *and both tasks i, j belong to the same BP, if* $a_j \leq a_{j+1} \leq \cdots \leq a_i$, *then*

$$\mathcal{S}_i^j = \min\left(\mathcal{C}_i^j, \mu_j \tau_{\min}\right) \tag{16}$$

Proof There are two possible cases: (*i*) Task j is not the first task in its corresponding BP and (*ii*) Task j is the first one in the BP. For case (*i*), we directly have $\mathcal{X}_j - \mathcal{X}_{j-1} = \mu_j \tau_{\min}$ which implies Eq. 16 based on Lemma 3.3. For case (*ii*), since task j is the first task in the BP, we have $\mathcal{X}_{j-1} \leq a_j$ and $\mathcal{X}_j - a_j = \mu_j \tau_{\min}$. Combining these facts with $a_j \leq a_{j+1}$ and Eq. 11, we have

$$\mathcal{X}_j - \mathcal{X}_{j-1} \geq \mathcal{X}_j - a_j \geq \mathcal{X}_j - a_{j+1} \geq \mathcal{C}_i^j$$

$$\mu_j \tau_{\min} = \mathcal{X}_j - a_j \geq \mathcal{X}_j - a_{j+1} \geq \mathcal{C}_i^j$$

which implies Eq. 16 based on Lemma 3.3. □

Lemma 3.5 *For any i, j such that $j \leq i$ and both tasks i, j belong to the same BP, and for any $l \in \{j, \ldots, i\}$,*

$$\mathcal{S}_i^j = \min\left(\mathcal{C}_i^l, \mathcal{S}_l^j\right)$$

Proof This is easily verified from Eqs. 11 and 12. □

The theorem below identifies an important optimality property of the MST under certain conditions. In particular, it singles out a MST as one that must be dropped in coming up with a solution of the ACP.

Theorem 3.1 *Suppose task m is a FIT in the BP $\{k, ..., n\}$ and task r is the MST relative to m. If $a_k \leq a_{k+1} \leq \cdots \leq a_r$ and either of the two conditions below is satisfied:*

$$(1)\ r = m \text{ or } (2)\ \mathcal{S}_m^r \geq \mathcal{X}_m - d_m, \tag{17}$$

then there must exist an optimal solution $[z_1^*, ..., z_N^*]$ of the ACP such that $z_r^* = 0$.

Proof Assume on the contrary that $z_r^* = 1$ in all ACP solutions. Based on Lemma 3.1, there always exists some task q ($k \leq q \leq m$) which is removed in any feasible solution. Without loss of generality, there must exist an optimal solution $[z_1^*, ..., z_N^*]$ such that $z_r^* = 1$ and $z_q^* = 0$ ($q \neq r$).

Let us construct a potential solution $[\bar{z}_1, ..., \bar{z}_N]$ such that $\bar{z}_r = 0$, $\bar{z}_q = 1$ and $\bar{z}_i = z_i^*$ for all $i \neq q, r$; that is, the role of r and q is interchanged in $[z_1^*, ..., z_N^*]$ and $[\bar{z}_1, ..., \bar{z}_N]$. If we can show that $[\bar{z}_1, ..., \bar{z}_N]$ is a feasible solution, then $[\bar{z}_1, ..., \bar{z}_N]$ must also be optimal because $\sum_{i=1}^N \bar{z}_i = \sum_{i=1}^N z_i^*$, which contradicts the assumption that r is admitted in all optimal solutions. Therefore, the theorem can be proved by showing the feasibility of the potential solution $[\bar{z}_1, ..., \bar{z}_N]$.

Let x_i^* and $\bar{x}_i$ denote the departure time of task i resulting from the optimal solution $[z_1^*, ..., z_N^*]$ and the potential solution $[\bar{z}_1, ..., \bar{z}_N]$ respectively. In the following, we will prove the feasibility of $[\bar{z}_1, ..., \bar{z}_N]$, that is, we will show that

$$\bar{z}_i(\bar{x}_i - d_i) \leq 0, \quad \forall\, i = 1, ..., N \tag{18}$$

First, we prove Eq. 18 for $i = 1, ..., k-1$, i.e.,

$$\bar{z}_i(\bar{x}_i - d_i) \leq 0, \quad \forall\, i = 1, ..., k-1 \tag{19}$$

Since $[z_1^*, ..., z_N^*]$ must be feasible, we have

$$z_i^*\left(x_i^* - d_i\right) \leq 0, \quad \forall\, i = 1, ..., N \tag{20}$$

Since $\bar{z}_i = z_i^*$ for $i = 1, ..., k-1$, we have $\bar{x}_i = x_i^*$ for $i = 1, ..., k-1$. Combining this with Eq. 20, we obtain Eq. 19.

Second, we prove Eq. 18 for $i = k, ..., m-1$, i.e.,

$$\bar{z}_i(\bar{x}_i - d_i) \leq 0, \quad \forall\, i = k, ..., m-1 \tag{21}$$

Since m is the first infeasible task, we have

$$\mathcal{X}_i \leq d_i \quad \forall\, i = k, ..., m-1 \tag{22}$$

Since $\bar{x}_i$ is the departure time after the possible removal of some tasks,

$$\bar{x}_i \leq \mathcal{X}_i, \quad \forall\, i = 1, ..., N \tag{23}$$

Combining Eqs. 23 and 22, we have $\bar{x}_i \leq d_i$ for all $i = k, ..., m-1$, which implies Eq. 21.

Third, we prove Eq. 18 for $i = m$, i.e.,

$$\bar{z}_m(\bar{x}_m - d_m) \leq 0 \tag{24}$$

If condition (1) holds, we have $r = m$ and $\bar{z}_m = \bar{z}_r = 0$ so that Eq. 24 immediately follows. If condition (2) holds, then $\mathcal{S}_m^r \geq \mathcal{X}_m - d_m$ and Eq. 8 imply that $\mathcal{X}_m^r \leq d_m$. Since at least task r is removed in $[\bar{z}_1, ..., \bar{z}_N]$, we have $\bar{x}_m \leq \mathcal{X}_m^r$. Therefore, it immediately follows that $\bar{x}_m \leq d_m$, which also implies Eq. 24.

Finally, we prove Eq. 18 for $i = m+1, ..., N$, i.e.,

$$\bar{z}_i(\bar{x}_i - d_i) \leq 0, \quad \forall\, i = m+1, ..., N \tag{25}$$

for which there are two possible cases.

Case 1: *No task between k and r is dropped in the optimal solution* $[z_1^*, ..., z_N^*]$, that is, $z_i^* = 1$ for all $i = k, ..., r$. In this case, by Lemma 3.1, there is at least one task dropped in $\{r+1, ..., m\}$ in the optimal solution $[z_1^*, ..., z_N^*]$. Thus, we can select q to be the first task removed after r, that is, $q > r$, $z_q^* = 0$, and $z_i^* = 1$ for $i = r, ..., q-1$. Since task r is the only task removed between k and q in the potential solution $[\bar{z}_1, ..., \bar{z}_N]$, we have $\bar{x}_q = \mathcal{X}_q^r$. Similarly, task q is the only task removed between k and q in the optimal solution $[z_1^*, ..., z_N^*]$, so we have $x_q^* = \mathcal{X}_q^q$. Then, from Eq. 8, we have

$$\mathcal{S}_q^r = \mathcal{X}_q - \bar{x}_q, \quad \mathcal{S}_q^q = \mathcal{X}_q - x_q^* \tag{26}$$

and from Lemma 3.5,

$$\mathcal{S}_m^r = \min\left(\mathcal{C}_m^q, \mathcal{S}_q^r\right), \quad \mathcal{S}_m^q = \min\left(\mathcal{C}_m^q, \mathcal{S}_q^q\right) \tag{27}$$

Since r is the MST and $r < q$, we obtain from Eq. 10 that

$$\mathcal{S}_m^r > \mathcal{S}_m^q \tag{28}$$

Using Eqs. 27 and 28, we can easily verify that $\mathcal{S}_q^r > \mathcal{S}_q^q$ and combining this with Eq. 26, we have $\bar{x}_q < x_q^*$. From the system dynamics, we have

$$\bar{x}_i = \max(\bar{x}_{i-1}, a_i\bar{z}_i) + \bar{z}_i\mu_i\tau_{\min}, \quad \forall\, i = q+1, ..., N$$

$$x_i^* = \max\left(x_{i-1}^*, a_i z_i^*\right) + z_i^*\mu_i\tau_{\min}, \quad \forall\, i = q+1, ..., N$$

Since $\bar{x}_q < x_q^*$ and $\bar{z}_i = z_i^*$ for $i = q+1, ..., N$, we can easily see that $\bar{x}_i \le x_i^*$ for all $i = q+1, ..., N$, which implies

$$\bar{z}_i(\bar{x}_i - d_i) \le z_i^*(x_i^* - d_i), \quad \forall\, i = q+1, ..., N$$

Combining this with Eq. 20, we have

$$\bar{z}_i(\bar{x}_i - d_i) \le 0, \quad \forall\, i = q+1, ..., N \tag{29}$$

and since $q \le m$, we obtain Eq. 25.

Case 2: *At least one task is dropped between k and r in the optimal solution* $[z_1^*, ..., z_N^*]$. Thus, we can select q to be the last task removed before r, that is, $q < r$, $z_q^* = 0$, and $z_i^* = 1$ for $i = q+1, ..., r$. From Lemma 3.5, we have

$$\mathcal{S}_m^r = \min\left(\mathcal{C}_m^r, \mathcal{S}_r^r\right), \quad \mathcal{S}_m^q = \min\left(\mathcal{C}_m^r, \mathcal{S}_r^q\right) \tag{30}$$

Since r is the MST and $q < r$, we obtain from Eq. 9:

$$\mathcal{S}_m^r \ge \mathcal{S}_m^q \tag{31}$$

We now further divide *Case 2* into two subcases, $\mathcal{S}_r^r < \mathcal{S}_r^q$ and $\mathcal{S}_r^r \ge \mathcal{S}_r^q$ in Eq. 30, as follows.

Case 2a: $\mathcal{S}_r^r < \mathcal{S}_r^q$. In this case, it follows from Eqs. 30 and 31 that

$$\mathcal{S}_r^r \ge \mathcal{C}_m^r, \quad \mathcal{S}_r^q \ge \mathcal{C}_m^r \tag{32}$$

Let $l = \arg\min_{i=r+1,...,m}\{\mathcal{X}_{i-1} - a_i\}$. Then, from Eq. 11,

$$\mathcal{C}_m^r = \mathcal{X}_l - a_{l+1}, \quad \mathcal{C}_l^r \ge \mathcal{X}_l - a_{l+1} \tag{33}$$

Invoking Lemma 3.5, we have

$$\mathcal{S}_l^r = \min\left(\mathcal{C}_l^r, \mathcal{S}_r^r\right), \quad \mathcal{S}_l^q = \min\left(\mathcal{C}_l^r, \mathcal{S}_r^q\right) \tag{34}$$

Combining Eqs. 32, 33 and 34, we have

$$\mathcal{S}_l^r \ge \mathcal{X}_l - a_{l+1}, \quad \mathcal{S}_l^q \ge \mathcal{X}_l - a_{l+1}$$

Then, from Eq. 8, we have

$$\mathcal{X}_l^r \le a_{l+1}, \quad \mathcal{X}_l^q \le a_{l+1}$$

Since $\bar{x}_l \le \mathcal{X}_l^r$ and $x_l^* \le \mathcal{X}_l^q$, we have

$$\bar{x}_l \le a_{l+1}, \quad x_l^* \le a_{l+1}$$

which means that both $\bar{x}_i$ and x_i^* for $i = l+1, ..., N$ only depend on $\bar{z}_i$ and z_i^* for $i = l+1, ..., N$ respectively. Since $\bar{z}_i = z_i^*$ for $i = l+1, ..., N$, we have $\bar{x}_i = x_i^*$ for $i = l+1, ..., N$, which implies that

$$\bar{z}_i(\bar{x}_i - d_i) \le 0, \quad \forall\, i = l+1, ..., N \tag{35}$$

Since $l \le m$, we obtain Eq. 25 for this subcase.

Case 2b: $\mathcal{S}_r^r \ge \mathcal{S}_r^q$. We now introduce another potential solution $[\tilde{z}_1, ..., \tilde{z}_N]$ of the ACP such that

$$\tilde{z}_i = \bar{z}_i = z_i^*, \quad i \ne q, r; \qquad \tilde{z}_i = 1, \quad i = q, r. \tag{36}$$

Its associated departure times satisfy

$$\begin{aligned}\tilde{x}_{q-1} &= \bar{x}_{q-1} = x^*_{q-1}\\ \tilde{x}_i &= \max(\tilde{x}_{i-1}, a_i) + \mu_i \tau_{\min}, \quad i = q, ..., r.\end{aligned} \tag{37}$$

It follows from Eq. 36 that $[\bar{z}_1, ..., \bar{z}_N]$ is obtained by removing only task r from $[\tilde{z}_1, ..., \tilde{z}_N]$ and that $[z^*_1, ..., z^*_N]$ is obtained by removing only task q from $[\tilde{z}_1, ..., \tilde{z}_N]$. Now, if there exists some i $(q+1 \le i \le r)$ such that $\tilde{x}_{i-1} \le a_i$, then removing task q from the solution $[\tilde{z}_1, ..., \tilde{z}_N]$ has no effect on $\tilde{x}_r$ because tasks q and r are decoupled by task i, that is, $x^*_r = \tilde{x}_r$. Since $[\bar{z}_1, ..., \bar{z}_N]$ has task r removed relative to $[\tilde{z}_1, ..., \tilde{z}_N]$, we have $\bar{x}_r \le \tilde{x}_r$, therefore

$$x^*_r \ge \bar{x}_r \tag{38}$$

Let us now consider the remaining case where $\tilde{x}_{i-1} > a_i$ for all $q+1 \le i \le r$. Then, Eq. 37 reduces to

$$\tilde{x}_i = \tilde{x}_{i-1} + \mu_i \tau_{\min}, \quad i = q+1, ..., r. \tag{39}$$

Since task r is removed in $[\bar{z}_1, ..., \bar{z}_N]$ and $\mathcal{S}^r_r = \min(+\infty, \mu_r \tau_{\min}) = \mu_r \tau_{\min}$, we have

$$\bar{x}_r = \tilde{x}_{r-1} = \tilde{x}_r - \mu_r \tau_{\min} = \tilde{x}_r - \mathcal{S}^r_r \tag{40}$$

Since task q is removed in $[z^*_1, ..., z^*_N]$, $x^*_q = x^*_{q-1}$. Using this fact along with Eq. 39 and $x^*_i = \max(x^*_{i-1}, a_i) + \mu_i \tau_{\min}$, we have

$$\tilde{x}_i - x^*_i = \min(\tilde{x}_{i-1} - a_i, \tilde{x}_{i-1} - x^*_{i-1}), \quad i = q+1, ..., r.$$

which implies that

$$\begin{aligned}\tilde{x}_r - x^*_r &= \min\left(\tilde{x}_{r-1} - a_r, \tilde{x}_{r-1} - x^*_{r-1}\right)\\ &= \min\left(\tilde{x}_{r-1} - a_r, \min\left(\tilde{x}_{r-2} - a_{r-1}, \tilde{x}_{r-2} - x^*_{r-2}\right)\right)\\ &= \cdots = \min\left(\tilde{x}_{r-1} - a_r, \tilde{x}_{r-2} - a_{r-1}, ..., \tilde{x}_q - a_{q+1}, \tilde{x}_q - x^*_q\right)\end{aligned}$$

Let $\tilde{\mathcal{C}}^j_i = \min_{l=j+1,...,i}(\tilde{x}_{l-1} - a_l)$. Combining this with the equality above, we have

$$\tilde{x}_r - x^*_r = \min\left(\tilde{\mathcal{C}}^q_r, \tilde{x}_q - x^*_q\right) = \min\left(\tilde{\mathcal{C}}^q_r, \tilde{x}_q - \tilde{x}_{q-1}\right) \tag{41}$$

In the following, we show that if $a_k \le a_{k+1} \le \cdots \le a_r$, then

$$x^*_r = \tilde{x}_r - \min\left(\tilde{\mathcal{C}}^q_r, \mu_q \tau_{\min}\right) \tag{42}$$

There are two possible cases: (i) $\tilde{x}_{q-1} > a_q$ and (ii) $\tilde{x}_{q-1} \le a_q$. For case (i), from Eq. 37, we have $\tilde{x}_q - \tilde{x}_{q-1} = \mu_q \tau_{\min}$ which implies Eq. 42 by using Eq. 41. For case (ii), from $a_k \le a_{k+1} \le \cdots \le a_r$, we have $a_q \le a_{q+1}$ and

$$\begin{aligned}\tilde{x}_q - \tilde{x}_{q-1} \ge \tilde{x}_q - a_q \ge \tilde{x}_q - a_{q+1} \ge \tilde{\mathcal{C}}^q_r\\ \mu_q \tau_{\min} = \tilde{x}_q - a_q \ge \tilde{x}_q - a_{q+1} \ge \tilde{\mathcal{C}}^q_r\end{aligned}$$

which implies Eq. 42 by using Eq. 41.

Since $\tilde{x}_i \leq \mathcal{X}_i$, we have

$$\tilde{\mathcal{C}}_r^q \leq \min_{i=q+1,\ldots,r} (\tilde{x}_{i-1} - a_i) \leq \min_{i=q+1,\ldots,r} (\mathcal{X}_{i-1} - a_i) = \mathcal{C}_r^q \tag{43}$$

which implies that

$$\min\left(\tilde{\mathcal{C}}_r^q, \mu_q \tau_{\min}\right) \leq \min\left(\mathcal{C}_r^q, \mu_q \tau_{\min}\right) = \mathcal{S}_r^q \tag{44}$$

It follows from Eq. 42 and Eq. 44 that $x_r^* \geq \tilde{x}_r - \mathcal{S}_r^q$. Combining this with Eq. 40 and $\mathcal{S}_r^r \geq \mathcal{S}_r^q$, we obtain

$$x_r^* \geq \tilde{x}_r - \mathcal{S}_r^q \geq \tilde{x}_r - \mathcal{S}_r^r = \bar{x}_r$$

that is, Eq. 38 again holds. From Eq. 38 and the fact that $\bar{z}_i = z_i^*$ for $i = r+1, ..., N$, we have $\bar{x}_i \leq x_i^*$ for $i = r+1, ..., N$, which implies

$$\bar{z}_i(\bar{x}_i - d_i) \leq z_i^*(x_i^* - d_i) \leq 0, \quad \forall\, i = r, ..., N \tag{45}$$

Since $r \leq m$, we obtain Eq. 25 for this subcase as well, which completes the proof. □

Corollary 3.1 *Suppose task $m \in \mathcal{R}$ is a FIT and task r is its corresponding MST. If $a_1 \leq \cdots \leq a_r$ and $d_{m-1} \leq d_m$, then there must exist an optimal solution $[z_1^*, ..., z_N^*]$ of the ACP such that $z_r^* = 0$.*

Proof There are two possible cases: (*i*) $r = m$ (*ii*) $r < m$. For the first case, we have $z_r^* = 0$ based on Theorem 3.1. For the second case, since task r is a MST and $r < m$, we have $\mathcal{S}_m^r \geq \mathcal{S}_m^m$, which implies that

$$\mathcal{X}_m - \mathcal{S}_m^r \leq \mathcal{X}_m - \mathcal{S}_m^m = \mathcal{X}_m - \mu_j \tau_{\min} = \mathcal{X}_{m-1} \tag{46}$$

Since m is the first infeasible task, we have $\mathcal{X}_{m-1} \leq d_{m-1}$. Then, from $d_{m-1} \leq d_m$, we have $\mathcal{X}_{m-1} \leq d_m$. Combining this with Eq. 46, we have $\mathcal{X}_m - \mathcal{S}_m^r \leq d_m$, that is, $\mathcal{S}_m^r \geq \mathcal{X}_m - d_m$. Then, from Theorem 3.1, we have $z_r^* = 0$. □

Theorem 3.1 provides a sufficient condition Eq. 17 towards optimality. If the condition is satisfied, we can directly remove the corresponding MSTs, construct a new smaller problem and check the condition again. If the condition is still satisfied, we can keep removing MSTs. Furthermore, if the condition is never violated until a feasible schedule is obtained (such as the compatible case in Corollary 3.1), then the remaining tasks turn out to yield the optimal solution. Even if the condition Eq. 17 is not satisfied, Theorem 3.1 can still help us develop an efficient algorithm to derive a near-optimal solution. Before doing so, we introduce another step which can be utilized to improve the efficiency of such an algorithm.

3.3 Deadline reduction

In this section, we take advantage of the fact that non-removable tasks implicitly impose more stringent deadlines on some removable tasks. In particular, all tasks before a non-removable one must complete early enough to guarantee that this non-removable task meets its deadline even if the deadlines of these previous tasks are relatively loose. The key idea then is that the existence of non-removable tasks

provides an opportunity to modify the ACP by reducing some task deadlines *without affecting the optimal solution*. To formalize this, let

$$\delta_i = \min(d_i, \delta_j - \mu_j \tau_{\min}), \text{ for } i = 1, ..., N-1 \text{ and } \delta_N = d_N,$$
$$j = \arg\min_{l \geq i} \{l : l \notin \mathcal{R}\} \tag{47}$$

where task j is simply the first non-removable task after task i. If no such j exists, we have $\delta_i = d_i$.

Lemma 3.6 *An optimal solution of the ACP is invariant to the replacement of d_i by δ_i for $i = 1, ..., N$.*

Proof Assume $z_1, ..., z_N$ is an arbitrary feasible solution in the ACP with task deadlines $d_1, ..., d_N$. First, we prove that $z_1, ..., z_N$ is also feasible in the ACP with $\delta_1, ..., \delta_N$ replacing $d_1, ..., d_N$. For each task i, there are two possible cases, (*i*) $z_i = 0$ and (*ii*) $z_i = 1$. For case (*i*), since $z_i = 0$, task i's deadline can be overlooked, so replacing d_i with δ_i will still satisfy Eq. 4.

For case (*ii*), without loss of generality, assume $j_1, ..., j_p$ are non-removable tasks after task i, where $j_1 \leq ... \leq j_p$. Since there are no non-removable task after task j_p, we have $\delta_{j_p} = d_{j_p}$ from Eq. 47. Since $j_p \notin \mathcal{R}$, we have $z_{j_p} = 1$, which requires

$$x_{j_p} \leq d_{j_p} = \delta_{j_p}$$

Since $j_{p-1} \notin \mathcal{R}$, we have $z_{j_{p-1}} = 1$, which requires $x_{j_p} \leq d_{j_p}$. Moreover, to satisfy the constraint above, i.e., $x_{j_p} \leq \delta_{j_p}$, task j_{p-1} should also satisfy $x_{j_{p-1}} \leq \delta_{j_p} - \mu_{j_p} \tau_{\min}$. Therefore, we have

$$x_{j_{p-1}} \leq \min(d_{j_{p-1}}, \delta_{j_p} - \mu_{j_p} \tau_{\min}) = \delta_{j_{p-1}}$$

Similarly, we proceed to task j_{p-2} and obtain:

$$x_{j_{p-2}} \leq \min(d_{j_{p-2}}, \delta_{j_{p-1}} - \mu_{j_{p-1}} \tau_{\min}) = \delta_{j_{p-2}}$$

Finally, we get

$$x_{j_1} \leq \min(d_{j_1}, \delta_{j_2} - \mu_{j_2} \tau_{\min}) = \delta_{j_1}$$

Since in case (*ii*) we have $z_i = 1$, this implies that $x_i \leq d_i$. Furthermore, to satisfy the constraint $x_{j_1} \leq \delta_{j_1}$, task i should also satisfy $x_i \leq \delta_{j_1} - \mu_{j_1} \tau_{\min}$. Therefore, it follows that

$$x_i \leq \min(d_i, \delta_{j_1} - \mu_{j_1} \tau_{\min}) = \delta_i$$

which implies that replacing d_i by δ_i will still satisfy Eq. 4.

In view of the above, any feasible solution in the original ACP with task deadlines $d_1, ..., d_N$ is also feasible in the modified ACP with $\delta_1, ..., \delta_N$. From Eq. 47, we have $\delta_i \leq d_i$ for $i = 1, ..., N$, which implies that any feasible solution in the modified ACP with $\delta_1, ..., \delta_N$ is also feasible in the original ACP with $d_1, ..., d_N$. Therefore, the feasible space is invariant by replacing d_i by δ_i for $i = 1, ..., N$, which implies that the optimal solution is also invariant. □

Lemma 3.7 *If task m is a FIT in the modified ACP with task deadlines $\delta_1, ..., \delta_N$, then task m must be removable.*

Proof Assume on the contrary that task m is a FIT and non-removable with $\delta_1, ..., \delta_N$. Without loss of generality, task m belongs to the BP $\{k, ..., n\}$. Since task m is a FIT, we have

$$\mathcal{X}_m > \delta_m \quad \text{and} \quad \mathcal{X}_j \leq \delta_j, \quad \text{for} \quad k \leq j \leq m-1. \tag{48}$$

Combining this with the fact that task m is non-removable in the modified ACP with $\delta_1, ..., \delta_N$, we have

$$\mathcal{X}_{m-1} \leq \delta_{m-1} = \min(d_{m-1}, \delta_m - \mu_m \tau_{\min}) \leq \delta_m - \mu_m \tau_{\min}$$

Moreover, it must hold that $m > k$, otherwise the ACP is infeasible. Thus, tasks $m-1$ and m belong to the same BP, which implies that $\mathcal{X}_m = \mathcal{X}_{m-1} + \mu_m \tau_{\min}$. Combining this fact with the inequality above, we have $\mathcal{X}_m \leq \delta_m$, which contradicts $\mathcal{X}_m > \delta_m$ in Eq. 48. □

The value of the deadline reduction process resulting in a modified ACP with invariant optimal solutions lies in the fact that the new deadlines tend to be ordered in a way which is closer to the compatible case (i.e., $a_i < a_j \Rightarrow \delta_i \leq \delta_j$ and $\delta_i < \delta_j \Rightarrow a_i \leq a_j$). Consequently, it is more likely that the conditions of Corollary 3.1 are satisfied, hence the optimality condition (17) in Theorem 3.1 also holds. Therefore, the deadline reduction in Eq. 47 can improve the applicability of Theorem 3.1. The following corollary is a useful special case that demonstrates this effect.

Corollary 3.2 *If task m is a FIT in the modified ACP with $\delta_1, ..., \delta_N$, FIFO is adopted, and $\mu_i = \mu_j$ for all i, j, then an optimal solution of the original ACP with $d_1, ..., d_N$ satisfies $z_m^* = 0$.*

Proof Without loss of generality, task m belongs to the BP $\{k, ..., n\}$. Since FIFO is adopted, from Lemma 3.4, we have

$$\mathcal{S}_m^j = \min\left(\mathcal{C}_m^j, \mu_j \tau_{\min}\right), \quad \text{for} \quad k \leq j \leq m$$

Then from $\mu_i = \mu_j$ for all i, j, we have

$$\mathcal{S}_m^m \geq \mathcal{S}_m^j, \quad \text{for} \quad k \leq j < m \tag{49}$$

Since task m is a FIT in the modified ACP with $\delta_1, ..., \delta_N$, task m must be removable from Lemma 3.7. Combining this with Eq. 49 and the definition of MST, we conclude that task m is the MST. From Theorem 3.1, an optimal solution of the ACP with $\delta_1, ..., \delta_N$ implies that $z_m^* = 0$. Then, from Lemma 3.6, an optimal solution of the ACP with $d_1, ..., d_N$ also satisfies $z_m^* = 0$. □

It is interesting to observe that a FIT may be non-removable in the original ACP with $d_1, ..., d_N$ and, further, the FIT may not be the MST; in this case, the optimality condition (17) in Theorem 3.1 may not be applied. However, all FITs must be removable in the *modified* ACP with $\delta_1, ..., \delta_N$ and the optimality condition (17) can be guaranteed for the case in Corollary 3.2.

Table 2 First order maximal shift task algorithm (MSTA-1)

Step 1:	Reduce d_i to δ_i for $i = 1, ..., N$;
Step 2:	Pick a BP $\{k, ..., n\}$, identify its FIT m and find the MST r;
Step 3:	If $r = m$ or $\mathcal{S}_m^r \geq \mathcal{X}_m - \delta_m$, then construct a new ACP by removing task r; else construct a new ACP by removing task m.
Step 4:	If there are no ITs in the new ACP, then end; otherwise goto Step 2.

3.4 Maximal shift task algorithm

In this section, we develop two efficient algorithms to derive a near-optimal (and possibly optimal) solution, termed *First Order Maximal Shift Task Algorithm* (MSTA-1) and *Second Order Maximal Shift Task Algorithm* (MSTA-2) respectively.

Table 2 describes MSTA-1, where the optimality condition (17) is checked in Step 3. If the optimality condition (17) is satisfied, MSTA-1 provides the optimal solution. Otherwise, a gap between its derived solution and the true optimal solution results from the possible failure to satisfy the optimality condition (17). In MSTA-1, we simply drop the FIT m when the optimality condition (17) fails. Obviously, this procedure cannot guarantee optimality because it is possible to remove more than one task before the FIT m to save more ITs after m. Therefore, we actually face two options when the optimality condition (17) is not met: (*i*) Drop the FIT m, or (*ii*) Keep the FIT m. Although we cannot determine which option is better in polynomial time, we can still increase the probability of choosing the better option by estimating and comparing the optimal number of accepted tasks under each of these two options.

Based on this idea, we obtain MSTA-2 in Table 3, where the optimality condition (17) is still checked in Step 3. The difference is that we do not simply drop m, but make decisions based on an estimated optimal number of accepted tasks under options (*i*) and (*ii*) mentioned above (denoted by U_1 and U_2 respectively) whenever the optimality condition (17) is not met. MSTA-1 can obtain the lower bounds of the optimal number of accepted tasks, which are good candidates for the two estimates U_1 and U_2. Under option (*i*), we just remove task m, construct a new ACP, and estimate the resulting optimal number of accepted tasks by MSTA-1. Under option (*ii*), since we decide to keep the FIT m, task m can be regarded as non-removable. Therefore, we can apply a deadline reduction process as in the previous section, that is, reduce δ_i to $\min(\delta_i, \delta_m - \mu_m \tau_{\min})$ for all tasks $i < m$. Then, we can estimate the optimal number of accepted tasks for option (*ii*) by MSTA-1 based on the reduced deadlines. By comparing the estimates, MSTA-2 can obtain a smaller optimality gap than MSTA-1.

Table 3 Second order maximal shift task algorithm (MSTA-2)

Step 1:	Reduce d_i to δ_i for $i = 1, ..., N$;
Step 2:	Pick a BP $\{k, ..., n\}$, identify its FIT m and find the MST r;
Step 3:	If $r = m$ or $\mathcal{S}_m^r \geq \mathcal{X}_m - \delta_m$, then construct a new ACP by removing task r; else compute U_1 and U_2 by applying MSTA-1; If $U_1 \geq U_2$, then construct a new ACP by removing m; else $\delta_i := \min(\delta_i, \delta_m - \mu_m \tau_{\min})$ for $i < m$ and goto Step 2;
Step 4:	If there are no ITs in the new ACP, then end; otherwise goto Step 2.

Table 4 Branch and bound algorithm

Step 1:	Reduce d_i to δ_i for $i = 1, ..., N$;
Step 2:	Pick a BP $\{k, ..., n\}$, identify its FIT m and find the MST r;
Step 3:	If $r = m$ or $\mathcal{S}_m^r \geq \mathcal{X}_m - \delta_m$, then construct a new ACP by removing task r; else construct a new ACP by setting $\delta_m = \mathcal{X}_m - \mathcal{S}_m^r$ and removing task r.
Step 4:	If there are no ITs in the new ACP, then end; otherwise goto Step 2.

In view of the analysis above, a natural question is whether we can develop an nth ($n \geq 3$) Order Maximal Shift Task Algorithm (MSTA-n) to further reduce the optimality gap. Indeed, we can always obtain MSTA-n by using MSTA-$(n-1)$ to estimate U_1 and U_2 in Step 3. However, this process will result in higher computational complexity. The worst case complexity of MSTA-1 is $O(N^2)$. For MSTA-2, the worst case complexity becomes to $O(N^3)$. Based on simulation experiments, MSTA-2 obtains sufficiently good results so it does not appear necessary to increase n to 3 or more in practice.

3.5 Branch and bound algorithm

In this section, we describe a branch and bound algorithm which we use to obtain an exact optimal solution serving as a baseline for the purpose of computing the optimality gap of MSTA-1 or MSTA-2. In the "branch" part, we break the ACP into subproblems. We only need to do so when we encounter a FIT m not satisfying the optimality condition (17). In this case, we break it into two subproblems: one is a new ACP obtained by adding the constraint $z_m = 0$ and the other a new ACP obtained by setting $\delta_i = \min(\delta_i, \delta_m - \mu_m \tau_{\min})$ for $i < m$.

In the "bound" part, we seek to obtain upper and lower bounds for the optimal number of accepted tasks. MSTA-1 and MSTA-2 can obtain lower bounds and we can derive an upper bound through the algorithm in Table 4, in which we actually construct a new ACP with larger deadlines and can guarantee to obtain its optimal solution. Since larger deadlines implies a larger optimal number of accepted tasks, its optimal solution becomes an upper bound of the optimal number of accepted tasks for the original ACP.

4 On-line admission control

So far, we have assumed that all task arrivals, number of operations, and deadlines are known at the time we solve the ACP and, subsequently, the original problem (1). This corresponds to an *off-line* approach. In this section, we tackle the *on-line* admission control problem where all tasks are assumed removable and arrival information is unavailable until tasks actually do arrive. Therefore, one can proceed in two ways: (*i*) Based only on the information available for tasks already in queue, and (*ii*) Using information in the form of the joint distribution of the arrival time, deadline, and required number of operations of the next arriving task.

We begin with case (*i*). Without loss of generality, we assume the current decision point is at time $t > 0$. Note that all tasks that arrived before t and have not been processed yet can be regarded as a set of tasks sharing a common arrival time t. Their arrival times and deadlines become compatible. Based on the results obtained

in classical scheduling theory, the optimal way to process these tasks is using the *Earliest Deadline First* (EDF) policy. Since all tasks have the common arrival time t, EDF is also FIFO and MSTA-1 can obtain the optimal admission control for all tasks in queue from Corollary 3.1.

Next, we consider case (*ii*), i.e., making decisions based on uncertain information about the next arriving task. Two natural questions arise: Is it possible to improve performance by using such information, and, if so, how to accomplish it? The first question can be answered through the simple example in Fig. 3, in which the next arriving task 5 is imminent and has an urgent deadline, while the current task 1 has a large number of operations and a relatively loose deadline. If no idling is allowed, task 5 has to be rejected. However, it can be saved without adversely affecting other tasks by slightly idling the processor as shown in Fig. 3. When distribution information for the next arrival task is available, it is possible to compute some metric quantifying the likelihood that the case above can occur. If the metric shows it can happen with sufficiently high probability, then we may rescue the upcoming task by postponing the start of the current task (i.e., by explicitly *idling*) and still successfully process the postponed task. The net effect is better performing admission control with high probability.

4.1 Idling conditions

To answer the second question, we can get some clues from the answer to the first one. In particular, the key is to specify the aforementioned metric and compute it. The theorem below is a step in this direction. Without loss of generality, assume that at the current decision time, t, there are n remaining tasks after applying admission control through MSTA-1 and they satisfy $d_1 \leq d_2 \leq \ldots \leq d_n$ and $t + \sum_{j=1}^{i} \mu_j \tau_{\min} \leq d_i$ for $i = 1, \ldots, n$ (as already mentioned, MSTA-1 yields the optimal solution in this

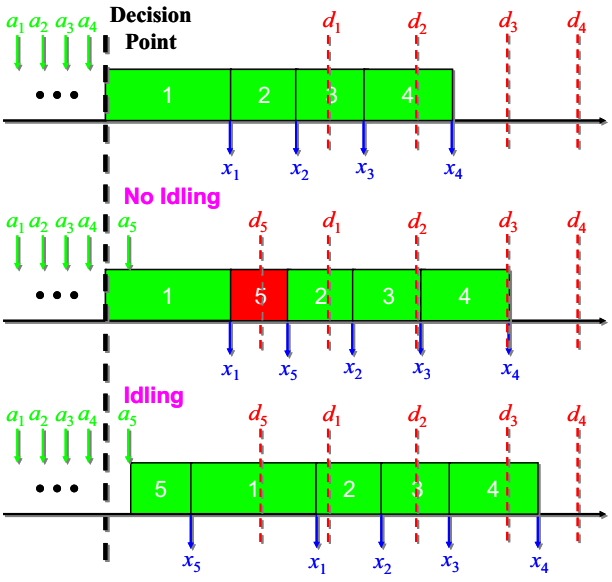


Fig. 3 No idling vs. idling

case). Moreover, the optimal processing time per operation can be assigned to task 1 using the CTDA (Mao et al. 2007) which solves problem (1). Let τ_1 be this processing time and let a_{n+1}, d_{n+1} and μ_{n+1} denote the arrival time, deadline, and number of operations of the next arriving task respectively. We are interested in establishing necessary and sufficient conditions under which the optimal number of remaining tasks that are all feasible (determined through MSTA-1 *without any future information available*) can be improved by idling *given some future information*. Then, clearly the optimal number of remaining tasks that are all feasible is at least n. If we consider the next arriving task only, then the best we can do is increase this number to $n+1$.

It follows that the statement "we can improve the optimal number of remaining tasks that are feasible under no future information" is equivalent to the two conditions:

Condition 1 The optimal number of remaining tasks is n if we do not postpone task 1.

Condition 2 The optimal number of remaining tasks is $n+1$ if we idle and postpone task 1.

Theorem 4.1 *Assume the current decision point is t and there are n tasks in queue that are feasible. Let* $\Phi(i)$ *denote the set of tasks before task i and task i itself when all* $n+1$ *tasks are sorted in ascending order of deadline and* $\Psi(i) = \Phi(i)\backslash\{1\}$. *Condition* 1 *and Condition* 2 *are satisfied if and only if*

$$\exists\, i \in \{2, ..., n+1\}, \quad t + \mu_1\tau_1 + \sum\nolimits_{j\in\Psi(i)} \mu_j\tau_{\min} > d_i; \tag{50}$$

$$a_{n+1} + \sum\nolimits_{j\in\Phi(i)} \mu_j\tau_{\min} \le d_i, \ \forall\, i = 1, ..., n+1. \tag{51}$$

Proof Let us first show that *Condition* 1 and *Condition* 2 are satisfied. We begin by proving that

$$a_{n+1} \le t + \mu_1\tau_1, \tag{52}$$

Using the definitions of $\Phi(i)$ and $\Psi(i)$, we have

$$\sum\nolimits_{j\in\Psi(i)} \mu_j\tau_{\min} \le \sum\nolimits_{j\in\Phi(i)} \mu_j\tau_{\min} \tag{53}$$

By Eq. 50, there exists some task i such that

$$t + \mu_1\tau_1 + \sum\nolimits_{j\in\Psi(i)} \mu_j\tau_{\min} > d_i$$

By Eq. 51, we have

$$a_{n+1} + \sum\nolimits_{j\in\Phi(i)} \mu_j\tau_{\min} \le d_i$$

Combining the three inequalities above, we obtain the inequality 52.

Second, we consider the case where task 1 is not postponed, implying that $t + \mu_1\tau_1$ is the next decision point. From Eq. 52, task $n+1$ has arrived at this decision point so that $t + \mu_1\tau_1 + \sum_{j\in\Psi(i)} \mu_{(j)}\tau_{\min}$ is the earliest departure time of task i for all $i = 2, ..., n+1$.

Based on Lemma 2.1, we can check feasibility by analyzing the earliest departure time which is achieved through the minimal processing time per operation $\tau_{\min}$. Then, from Eq. 50, at least one of tasks 2, ..., $n+1$ will violate its deadline. By assumption, there are n feasible tasks in queue so that we can always attain the feasibility of these n tasks by removing task $n+1$. Therefore, the optimal number of remaining tasks is n if we do not postpone task i, i.e., *Condition* 1 is met.

Next, we consider postponing task 1, implying that a_{n+1} is the next decision point and $a_{n+1} + \sum_{j\in\Phi(i)} \mu_{(j)}\tau_{\min}$ is the earliest departure time of task i for all $i = 1, ..., n+1$.

Similarly, we can still check feasibility based on the earliest departure time from Lemma 2.1. Then, from Eq. 51, all $n+1$ tasks can meet their deadlines. Thus, *Condition* 2 is satisfied as well.

Now let us establish Eqs. 51 and 50 when *Condition* 1 and *Condition* 2 are satisfied. First, we consider postponing task 1. From *Condition* 2 and Lemma 2.1, all $n+1$ tasks can meet their deadlines by using the minimal processing time per operation $\tau_{\min}$, which turns out to be the inequality (51).

Second, we prove Eq. 52 has to be satisfied. Assume on the contrary that $a_{n+1} > t + \mu_1\tau_1$. From Eqs. 53 and 51 obtained above, we have

$$a_{n+1} + \sum\nolimits_{j\in\Psi(i)} \mu_j\tau_{\min} \le d_i, \quad \forall\, i = 2, ..., n+1$$

Combining it with $a_{n+1} > t + \mu_1\tau_1$, all the other tasks can still meet their deadline by using $\tau_{\min}$ if we do not postpone task 1, which means the optimal number of remaining task also equals to $n+1$ in the case of not postponing task 1. This contradicts *Condition* 1 and leads to the conclusion that Eq. 52 must hold.

Finally, we consider the case where task 1 is not postponed. It follows from Eq. 52 that $t + \mu_1\tau_1 + \sum_{j\in\Psi(i)} \mu_{(j)}\tau_{\min}$ is the earliest departure time of task i for all $i = 2, ..., n+1$. Then, from *Condition* 1 and Lemma 2.1, if we do not postpone task 1, there is at least one among tasks 2, ..., $n+1$ such that its earliest departure time is larger than its deadline, which is precisely inequality (50). □

4.2 On-line admission control algorithm

Let f denote the joint distribution of $(a_{n+1}, d_{n+1}, \mu_{n+1})$ and $P(t, f)$ denote the probability that $(a_{n+1}, d_{n+1}, \mu_{n+1})$ satisfies Eqs. 50 and 51 at the decision point t. From Theorem 4.1, $P(t, f)$ can be the metric to indicate how likely it is that idling and postponing the current task improves performance. Thus, if $P(t, f) > p$ (typically, $p = 0.5$), then we postpone the current task and otherwise immediately process it. Although $P(t, f)$ may not be easy to compute in closed form, one can always estimate it through Monte Carlo methods, that is, randomly generating M samples of $(a^i_{n+1}, d^i_{n+1}, \mu^i_{n+1})$ and calculating $\sum_{i=1}^{M} \mathbf{1}(a^i_{n+1}, d^i_{n+1}, \mu^i_{n+1})/M$, where $\mathbf{1}(\cdot)$ is the indicator function indicating whether $(a^i_{n+1}, d^i_{n+1}, \mu^i_{n+1})$ satisfies Eqs. 50 and 51. When selecting M, we face a trade-off between computing time and the extent of performance improvement we can realize. We can save computing time by reducing M, but the estimation of $P(t, f)$ becomes less accurate. To guarantee a performance improvement, we need to increase the threshold, p, which in turn results in reduced improvement of performance.

Further, if the decision is to idle, there remains a question regarding the length of idling. Let $P(t + w, f)$ denote the probability that $(a_{n+1}, d_{n+1}, \mu_{n+1})$ satisfies

Table 5 On-line admission control algorithm

Step 1:	Sort all tasks waiting in queue in ascending order of deadline and apply MSTA and CTDA;
Step 2:	Estimate $P(t,\ f)$ by Monte Carlo method;
Step 3:	If $P(t,\ f) > 0.5$, then estimate w^* and postpone the current task for w^*; otherwise just process the current task.

Eqs. 50 and 51 if we postpone the current task by w and the next task still does not arrive. Define

$$\Omega = \{w : P(t + w,\ f) \leq 0.5,\ w \geq 0\} \text{ and } w^* = \min_{w \in \Omega} w.$$

We can see that w^* is the optimal idling time. However, w^* is much harder to compute in closed form than $P(t,\ f)$. Let $L = \min_{i=1,\ldots,n}(d_i - \sum_{j=1}^{i} \mu_i \tau_{\min})$. We can bound w^* as follows: $0 \leq w^* \leq L - t$, because Eq. 51 requires the next task to arrive before L. Moreover, it is clear that the longer idling is, the less likely we are to increase the optimal number of remaining tasks since less time becomes available for rescuing the next task. Thus, assuming that $P(t + w,\ f)$ is monotonically decreasing in w, an estimate of w^* may be efficiently obtained through simple binary search over the interval $[0,\ L - t]$. This discussion leads to the on-line admission control algorithm shown in Table 5.

5 Numerical results

5.1 Optimality gap

We first compare the optimality gap of the three algorithms: OAT (discussed in Section 1), MSTA-1 and MSTA-2. In this experiment, interarrival times have an exponential distribution with mean 8, μ_i is a random integer uniformly distributed in $\{1, ..., 10\}$, $d_i - a_i$ is uniformly distributed in $[2\mu_i, 4\mu_i]$ and $\tau_{\min} = 1$. We test cases where N varies from 20 to 50 in increments of 3 and randomly generate 100 samples for each N. The results are shown in Fig. 4, in which the optimality gap $G(U)$ is normalized using

$$G(U) = (U_B - U)/(N - U_B)$$

where U_B is the optimal number of accepted tasks obtained by the branch and bound algorithm and U is the result obtained by any one of the three algorithms of interest. The plots on the left compare the optimality gap in the average case and the ones on the right compare the optimality gap in the worst case. It can be easily seen that MSTA-2 has a much smaller optimality gap than the other two.

5.2 Idling vs non-idling

Next, we compare two on-line admission control algorithms: one without idling and the other with idling. The setting of the on-line experiments is the same as above and we select $M = 50$ and $p = 0.5$. Both algorithms were programmed using MATLAB 7.1 on an Intel Pentium4 3.06GHz, 1.0GB RAM machine. Figure 5 compares the performance of these two on-line algorithms, in which the average number of tasks

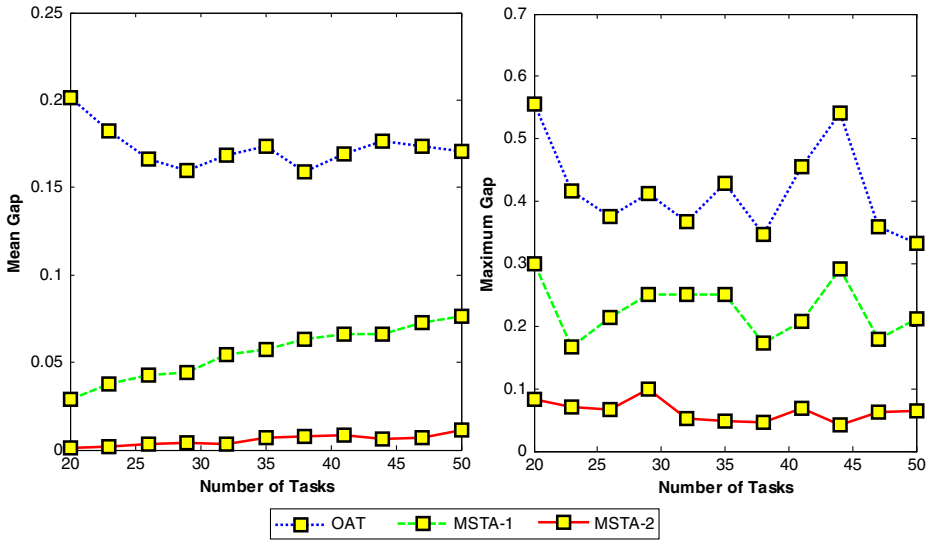


Fig. 4 Mean and maximum optimality gaps

removed is plotted as a function of the total number of tasks N. We see that idling results in removing about $28-31\%$ fewer tasks compared to no idling. Figure 6 compares the complexity of these two on-line algorithms in terms of average CPU time. Obviously, on-line admission control without idling has a lower complexity because it does not need to calculate the metric $P(t, f)$ and the length of idling. However, the algorithm with idling is still very efficient (e.g., it can obtain on-lines solutions for $N = 1000$ tasks in less than 0.9 seconds, i.e., less than 0.9 milliseconds for each task). Moreover, the efficiency can be further improved by programming in C code instead of MATLAB.

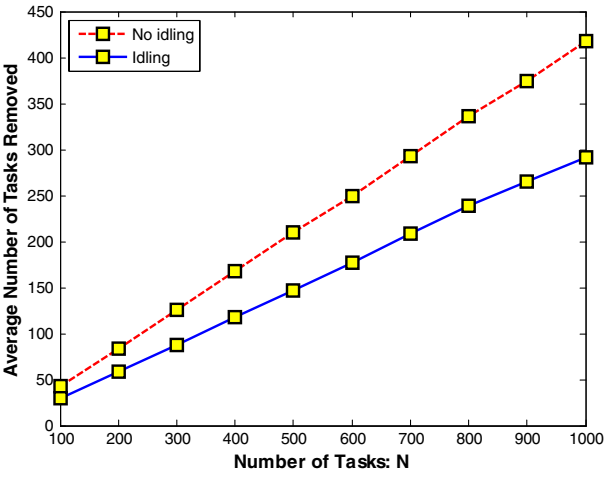


Fig. 5 Performance comparison of two on-line algorithms

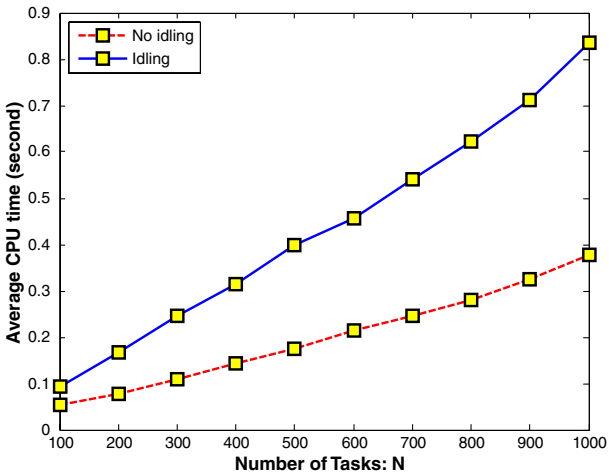


Fig. 6 Complexity comparison of two on-line algorithms

6 Conclusions

In this paper, we have removed the restriction that a feasible solution exists in Problem (1) for DES with hard real-time constraints and have formulated an associated admission control problem (ACP) for maximizing the number of tasks processed which are guaranteed feasibility. We exploit an optimality property based on identifying and removing "maximal shift tasks" and develop an efficient admission control algorithm, termed Maximal Shift Task Algorithm (MSTA). We show that the MSTA yields an optimal solution when a sufficient condition is satisfied, which actually applies to many situations in practice. Further, we exploit an invariance property of the ACP with respect to reducing deadlines of certain tasks and modify the original MSTA, referred to as First Order Maximal Shift Task Algorithm (MSTA-1), to obtain a Second Order Maximal Shift Task Algorithm (MSTA-2). The latter reduces the optimality gap present when MSTA-1 is only able to deliver a near-optimal solution.

In addition, we have shown that in on-line control cases where no future task information is available, idling, though possibly counterintutive, may in fact be optimal. We have also derived necessary and sufficient conditions under which this property holds and have defined a metric for evaluating when and how long it is optimal to idle.

Problem (1) characterizes a single-stage DES. Multi-stage systems studied in Mao and Cassandras (2006) face similar feasibility issues, which are complicated by the coupling among stages. Our ongoing work is aimed at studying the admission control problem in such systems.

References

Aydin H, Melhem R, Mossé D, Mejia-Alvarez P (2004) Power-aware scheduling for periodic real-time tasks. IEEE Trans Comput 53(5):584–600, May

Buttazzo GC (1997) Hard real-time computing systems: predictable scheduling algorithms and applications. Kluwer, Norwell

Chetto H, Chetto M (1989) Some results of the earliest deadline scheduling algorithm. IEEE Trans Software Eng 15(10):1261–1269

Dauzère-Pérès S, Sevaux M (2002) Using lagrangean relaxation to minimize the weighted number of late jobs on a single machine. Nav Res Logist 50(3):273–288

Dauzère-Pérès S, Sevaux M (2004) An exact method to minimize the number of tardy jobs in single machine scheduling. J Sched 7(6):405–420

Gamal AE, Nair C, Prabhakar B, Uysal-Biyikoglu E, Zahedi S (2002) Energy-efficient scheduling of packet transmissions over wireless networks. In: Proceedings of IEEE INFOCOM, vol 3, 23–27. IEEE, New York, pp 1773–1782

Jeffay K, Stanat DF, Martel CU (1991) On non-preemptive scheduling of periodic and sporadic tasks. In: Proc. of the IEEE real-time systems symposium. IEEE, Piscataway, pp 129–139

Jonsson J, Lonn H, Shin KG (1999) Non-preemptive scheduling of real-time threads on multi-level-context architectures. In: Proceedings of the IEEE workshop on parallel and distributed real-time dystems, vol 1586. Springer, Berlin, pp 363–374

Kise H, Ibaraki T, Mine H (1978) A solvable case of the one-machine scheduling problem with ready and due times. Oper Res 26(1):121–126

Lenstra JK, Rinnooy Kan AHG, Brucker P (1977) Complexity of machine scheduling problems. Ann Discrete Math 1:343–362

Liu JWS (2000) Real-time systems. Prentice Hall, Englewood Cliffs

Mao J, Cassandras CG (2006) Optimal control of multi-stage discrete event systems with real-time constraints. In: Proc. of 45rd IEEE conf. decision and control. IEEE, Piscataway, pp 1057–1062 (subm. to IEEE Trans. on Automatic Control, 2007)

Mao J, Cassandras CG (2007) Optimal admission control of discrete event systems with real-time constraints. In: Proc. of 46rd IEEE conf. decision and control. IEEE, Piscataway

Mao J, Cassandras CG, Zhao QC (2007) Optimal dynamic voltage scaling in power-limited systems with real-time constraints. IEEE Trans Mobile Comput 6(6):678–688, June

Miao L, Cassandras CG (2006) Optimal transmission scheduling for energy-efficient wireless networks. In: Proceedings of INFOCOM. IEEE, Piscataway

Moore JM (1968) A *n* job, one machine sequencing algorithm for minimizing the number of late jobs. Manag Sci 15(1):102–109

Pepyne DL, Cassandras CG (2000) Optimal control of hybrid systems in manufacturing. In: Proceedings of the IEEE, vol 88. IEEE, Piscataway, pp 1108–1123

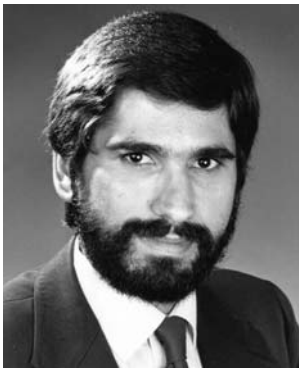
Schwan K, Zhou H (1992) Dynamic scheduling of hard real-time tasks and real-time threads. IEEE Trans Softw Eng 18(8):736–748

Silly M, Chetto H, Elyounsi N (1990) An optimal algorithm guaranteeing sporadic tasks in hard real-time systems. In: Proceedings of the 2nd IEEE symposium on parallel and distributed processing. IEEE, Piscataway, pp 578–585

Tia T, Liu W, Sun J, Ha R (1994) A linear-time optimal acceptance test for scheduling of hard real-time tasks. Preprint, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign

Yao F, Demers A, Shenker S (1995) A scheduling model for reduced cpu energy. In: Proceedings of the 36th Annual symposium on foundations of computer science (FOCS'95). IEEE Computer Society, Los Alamitos, pp 374–382

Jianfeng Mao received the B.E. degree (Major in Automatic Control) and M.E. degree (Major in Control Theory and Applications) from Tsinghua University, Beijing, China in 2001 and 2004, respectively. Currently, he is a Ph.D. candidate of Systems Engineering at Boston University, Boston, MA. He specializes in the areas of modeling and optimization of complex systems with application to sensor networks, manufacturing systems. He is a student member of IEEE.



Christos G. Cassandras received the B.S. degree from Yale University, New Haven, CT, the M.S.E.E degree from Stanford University, Stanford, CA, and the S.M. and Ph.D. degrees from Harvard University, Cambridge, MA, in 1977, 1978, 1979, and 1982, respectively. From 1982 to 1984 he was with ITP Boston, Inc. where he worked on the design of automated manufacturing systems. From 1984 to 1996 he was a Faculty Member at the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst. Currently, he is Head of the Division of Systems Engineering and Professor of Electrical and Computer Engineering at Boston University, Boston, MA and a founding member of the Center for Information and Systems Engineering (CISE). He specializes in the areas of discrete event and hybrid systems, stochastic optimization, and computer simulation, with applications to computer networks, sensor networks, manufacturing systems, transportation systems, and command/control systems. He has published over 250 papers in these areas, and four books. Dr. Cassandras is currently Editor-in-Chief of the IEEE Transactions on Automatic Control and has served on several editorial boards and as Guest Editor for various journals. He has also served on the IEEE Control Systems Society Board of Governors. He is the recipient of several awards, including the Distinguished Member Award of the IEEE Control Systems Society (2006), the 1999 Harold Chestnut Prize, and a 1991 Lilly Fellowship. He is a member of Phi Beta Kappa and Tau Beta Pi, a Fellow of the IEEE and a Fellow of the IFAC.