Optimal Dynamic Voltage Scaling in Power-Limited Systems with Real-Time Constraints *

Jianfeng Mao† Qianchuan Zhao‡ and Christos G. Cassandras §

Abstract

Dynamic voltage scaling is used in power-limited systems such as sensor networks as a means of conserving energy and prolonging their life. We consider a setting in which the tasks performed by such a system are nonpreemptive, aperiodic and have uncertain arrival times. Our objective is to control the processing rate over different tasks so as to minimize energy subject to hard real-time processing constraints. We prove that the solution to this problem reduces to two simpler problems which can be efficiently solved, leading to a new on-line dynamic voltage scaling algorithm. This algorithm is shown to have low complexity and, unlike similar state-of-the-art approaches, it involves no solution of nonlinear programming problems and is independent of the specific physical characteristics of the system. Both off-line and on-line versions of the algorithm are analyzed and numerical examples are provided to illustrate the relative advantages of the latter over the former.

Keywords: hard real time system, voltage scaling, optimal control, sensor networks

*This work was supported by NFSC Grant no.60274011. The work of the third author is supported by the National Science Foundation under Grants EEC-00-88073 and DMI-0330171, by AFOSR under contract F49620-01-0056, and by ARO under under grant DAAD19-01-0610.

†Contact information: Mailing address: Jianfeng Mao, Department of Automation, Tsinghua University, Beijing 100084, P.R.China. Email: maojf@mails.tsinghua.edu.cn. Tel. +86-10-62792491.

‡Mao and Zhao are with Center for Intelligent and Networked Systems in Department of Automation, Tsinghua University, Beijing 100084, China

§Cassandras is with the Manufacturing Engineering Department of Boston University, USA.
1 Introduction

Minimizing energy consumption in low-power systems has become a critical design consideration, especially in view of proliferating portable and mobile real-time embedded systems whose lifetime is strongly dependent on battery management. In emerging sensor networks, for example, nodes incorporate small, inexpensive devices with limited battery capabilities. Prolonging battery life is closely tied to the network’s overall performance; for instance, in sensor networks the failure of a few nodes can cause significant topological changes which require substantial additional power to reorganize the network [1]. In low-power systems, the processor reportedly accounts for 18-30% of the overall power consumption and often exceeds 50% [13]. Controlling the voltage and clock frequency provides the means to regulate processor power consumption leading to Dynamic Voltage Scaling (DVS) techniques [12],[13],[14]. For CMOS processors, the energy consumption $E$ is related to the operating voltage $V$ through

$$E = C_1 V^2$$

and processing frequency (clock speed) is given by

$$f = \frac{V - V_t}{C_2 V}$$

where $C_1, C_2$ are constants dependent on the physical characteristics of a device and $V_t$ is the threshold voltage, so that $V \geq V_t$. These relationships may be approximate, but the functional interdependence of $V$, $E$, and $f$ clearly indicates that reducing the voltage provides an opportunity to reduce energy at the expense of longer delays, which adversely affects performance with possibly catastrophic consequences in hard real-time systems [3]. Thus, managing this tradeoff becomes an essential design and dynamic control problem.

A number of DVS algorithms have been proposed over the last decade. Most of them are designed for preemptive scheduling of real-time systems, as in [2] and [9]. Nonpreemptive scheduling is often a better choice in practice, especially for systems with very limited resources, because uncontrolled preemption can give rise to a large number of context switches requiring larger stack sizes and increased energy consumption [7],[8]. DVS algorithms developed for the nonpreemptive case have been reviewed
Many of them were developed for systems with periodic tasks, as in [15]. In this paper, we consider a system with *aperiodic* tasks which arise in a setting consisting of asynchronously operating components (e.g., a sensor network where sensing units asynchronously supply data to a processing node). In this case, task arrival times at the processor are generally random; we will model them as being constrained to occur in a given time interval.

Our approach is motivated by the work in [5], where the nonpreemptive and aperiodic case is considered with a known arrival time schedule and an optimal control problem is solved with an objective function incorporating the tradeoff between processing performance and task timeliness. The problem is solved in [5] through the so-called Forward Decomposition Algorithm (FA) with applications motivated from manufacturing systems; it was later applied to the DVS problem in [10]. The setting in [5] and [10] corresponds to a system with soft real-time constraints, as in [10]. The FA can also be applied to a hard real time system, although details of such an extension have not been published to date. Although the FA has been shown to avoid the combinatorial complexity that often comes with such problems, it still requires the solution of $N$ (the number of tasks) nonlinear programming (NLP) problems, which is generally demanding for on-line applications with limited on-board computational capacity.

The contribution of this paper is the solution of the optimization problem whose goal is to assign processing times (equivalently, processor speeds through voltage control) to tasks so as to minimize a total energy consumption function while guaranteeing that no task completion exceeds a given deadline. Our approach is based on a similar optimization framework as in [4] and [5], but the structure of the problem in our setting leads to some attractive properties of the optimal sample path that we exploit to develop a new DVS algorithm for the types of systems described above. This algorithm is (i) computationally efficient in that no NLP problems need to be solved, and (ii) independent of the physical characteristics of the devices involved or the specific structure of the energy consumption function, as long this function is strictly convex and monotonically decreasing in the processing times.
The paper is organized as follows. In section 2, the optimization problem for a hard real-time system is formulated and the three main challenges it poses are identified. Section 3 presents the new DVS algorithm in which these challenges are met. Section 3.1 presents an off-line and an on-line DVS framework to deal with the issue of arrival time uncertainty. We then address the remaining two challenges that involve computational complexity. In section 3.2, we develop a decomposition approach leading to the Busy Period Decomposition Algorithm (BPDA) to efficiently obtain the structure of the optimal sample path of the system. In section 3.3, we enhance the decomposition approach by identifying “critical tasks” in the optimal sample path and subsequently develop the Critical Task Decomposition Algorithm (CTDA) to explicitly obtain the optimal processing times without solving any NLP problems. In section 4, some simulation-based experimental results are presented and discussed. Section 5 summarizes the paper and outlines directions for future work.

2 Problem formulation

The hard real-time system we consider is modeled as a single-stage queueing system with the objective of minimizing energy consumption while guaranteeing to meet deadlines in all cases. Let \( a_i \) and \( d_i \) denote the actual arrival time of task \( i \) and its deadline respectively. The arrival time \( a_i \) is unknown a priori, but we assume that it is constrained to occur in a known interval \([a_i^-, a_i^+]\); this is also referred to as “release time jitter” [11] (where “release time” is equivalent to “arrival time”) and includes situations where if expected tasks are not received within a particular time interval, then they are considered useless and are never processed (e.g., expected data that arrive too late to a processing node in a sensor network).

The system operates with nonpreemptive and aperiodic tasks. If all actual arrival times were known a priori, the associated scheduling problem would still be NP-complete (nonpreemptive scheduling of a set of independent tasks with arbitrary arrival times and deadlines on a variable-voltage processor is an NP-complete problem [6]). Given that in our setting arrival times are random and we are concerned
with maintaining simplicity in a system with often limited computational resources, we assume that all tasks are served on a first-come-first-served (FCFS) basis.

DVS techniques operate through the software/hardware interface [10]. The software is not aware of the operating voltage used by the hardware, i.e., the voltage is adjusted depending on the desired task processing time. Let \( \mu_i \) denote the number of operations needed for task \( i \), which may depend on the specifics of this task. Then, letting \( u_i \) denote the processing time for task \( i \) (equivalently, \( \mu_i/u_i \) is the processor speed when processing task \( i \)), we view \( u_i \) as a control variable in our problem setting and we define a power consumption function \( \theta_i(u_i) \) in terms of \( u_i \). Specifically, using the relationships (1)-(2), we can write

\[
\theta_i(u_i) = \mu_i E = \mu_i C_1 \left( \frac{V_t u_i}{u_i - \mu_i C_2} \right)^2
\]  

(3)

We emphasize that the precise form of \( \theta_i(u_i) \) or the values of the constants are not essential; as we shall see, what matters is only that \( \theta_i(u_i) \) is a strictly convex and monotone decreasing function of \( u_i \) for \( u_i > \mu_i C_2 \). Note that an additional constraint on \( V \) is imposed by the requirement that \( V \leq V_{\text{max}} \), where \( V_{\text{max}} \) is the maximal operating voltage. This, in turn, leads to a constraint on the control variables:

\[
u_i \geq u_{i\text{min}} = \frac{\mu_i C_2 V_{\text{max}}}{V_{\text{max}} - V_t}
\]  

(4)

Let \( x_i \) denote the departure (processing completion) time of task \( i \). We can now formulate the following optimization problem over \( N \) tasks required to be processed:

**Problem 1**

\[
\min_{u_1, \ldots, u_N} \left\{ J = \sum_{i=1}^{N} \theta_i(u_i) \right\}
\]

s.t. \( u_i \geq u_{i\text{min}}, \ i = 1, \ldots, N; \ x_0 = 0; \)
\[
x_i = \max (x_{i-1}, a_i) + u_i \leq d_i, \ i = 1, \ldots, N;
\]
\[
a_i \in [a_i^-, a_i^+], \ a_i^+ < a_{i+1}^+, \ i = 1, \ldots, N.
\]

In Problem 1, our goal is to assign processing times \( u_1, \ldots, u_N \) so as to minimize the total energy consumption over \( N \) tasks subject to \( u_i \geq u_{i\text{min}} \). The Lindley equation
\[ x_i = \max(x_{i-1}, a_i) + u_i \text{ and } a_1 < \cdots < a_N \text{ capture the standard queueing dynamics in the system under FCFS operation. The hard real-time constraints are captured through } x_i \leq d_i \text{ and the arrival time uncertainty is captured through } a_i \in [a_i^-, a_i^+]. \]

Since all \( a_i \) are random variables, Problem 1, as stated, cannot be solved as \( J \) is obviously dependent on these random variables. There are in fact three challenges to this problem: (i) The uncertainty introduced by “release time jitter” which makes it infeasible to obtain the optimal controls, (ii) The high dimension of the control vector, given by the number of tasks \( N \) which may be very large and can lead to combinatorial complexity similar to that encountered in [5], and (iii) The nondifferentiability of the constraints caused by the presence of the “max” operator, which would appear to call for the use of nonsmooth optimization methods. In the next section, we develop a new DVS algorithm designed to solve an appropriately modified version of Problem 1 by overcoming these three challenges.

3 Optimal DVS algorithm

3.1 Handling Uncertain Task Arrivals Off Line and On Line

As already pointed out, Problem 1 is a stochastic optimization problem, since actual arrival times are unknown, and the “ideal” optimal controls cannot be obtained. In practice, Problem 1 needs either to be replaced by one with an objective function involving some expectation or by estimates of the uncertain variables. We shall proceed along the second direction, in which case there are two ways to approach the problem, depending on whether one develops an off-line or an on-line DVS algorithm. In the off-line framework, all uncertain variables can only be estimated once, normally at the starting point. Then, all controls are computed based on this initial estimation process. In the on-line framework, the past history of the evolving process may be used and optimal controls can be updated at a number of appropriately defined decision points.

Given the presence of hard deadlines in Problem 1, our options for estimating the uncertain arrival times \( a_i \) are limited if we want to guarantee that \( x_i \leq d_i \). In fact,
worst case analysis is the only logical choice. Here, the worst case occurs when \( x_i \) reaches its maximum value for any \( a_i \). It is obvious in Problem 1 that if \( a_i = a_i^+ \) for every \( i = 1, \ldots, N \), then \( x_i \) reaches the maximum for each \( i = 1, \ldots, N \) regardless of the controls \( u_i, i = 1, \ldots, N \).

With the above observation in mind, let us first consider the off-line framework. In order to solve the optimization problem under worst-case conditions, we set our “estimate” of \( a_i \) to be \( a_i^+ \) for all \( i = 1, \ldots, N \) and all controls are determined at once. On the other hand, in an on-line framework, controls may be updated at selected “decision points”. In what follows, we choose task departure times to be these decision points (other options are the subject of ongoing work). Let \( x_{K-1} \) be the departure time of task \( K-1 \) and, consequently, such a decision point. Note that it often happens that a new task arrives before the current task finishes processing. Therefore, at time \( x_{K-1} \) it is possible that \( a_K \) has already occurred and is therefore known to the DVS controller; in fact, it is possible that several arrival times \( a_i, i > K-1 \), are such that \( a_i < x_{K-1} \), thus making this information available and providing an opportunity for improving upon the solution of the off-line problem. An example is shown in Fig. 1, where \( a_K < x_{K-1} < a_{K+1} \). Thus, at the decision point \( x_{K-1} \), the arrival time \( a_K \) is known exactly while \( a_i, i \geq K+1 \), still need to be estimated. Using the same worst-case approach as in the off-line case, we set \( a_i = a_i^+ \) for all \( i \geq K+1 \). Note, however, that since the process repeats at \( x_K \), we only need to determine an optimal \( u_K \) at time \( x_{K-1} \); there may be additional arrival time information over \( (x_{K-1}, x_K) \) which can further improve the choice of the next control, \( u_{K+1} \).

Let us use \( \bar{a}_i \) to denote the “estimate” of \( a_i \). In the off-line framework, we simply

\[ a_{K-1} \]

\[ a_K \]

\[ a_{K+1} \]

\[ a_{K+2} \]

\[ d_{K-1} \]

\[ x_{K-1} \]

Figure 1: On-line framework example: \( a_K \) does not need to be estimated at \( x_{K-1} \)
have $\bar{a}_i = a_i^+$. In the on-line setting, we have

$$\bar{a}_i = \begin{cases} 
    a_i & x_{K-1} \geq a_i \\
    a_i^+ & x_{K-1} < a_i
  \end{cases}$$

(5)

We can now modify Problem 1 as follows:

**Problem 2**

$$\min_{u_K, \ldots, u_N} \left\{ J = \sum_{i=K}^{N} \theta_i(u_i) \right\}$$

s.t. $u_i \geq u_{i\text{min}}, \quad i = K, \ldots, N; \quad x_0 = 0; \quad x_i = \max (x_{i-1}, \bar{a}_i) + u_i \leq d_i, \quad i = K, \ldots, N.$

This problem is solved at every decision point defined, depending on the selected framework. In the off-line case, it is solved once at time 0 with $K = 1$ and $\bar{a}_i = a_i^+$ for all $i = 1, \ldots, N$. In the on-line case, it is solved at every $x_{K-1}$, $K = 1, \ldots, N$ with $\bar{a}_i$ given by (5). These two frameworks are summarized in Table 1 and Table 2 respectively.

We should point out that although the on-line framework may yield lower cost (energy consumption) than the off-line one, applying it in practice may not outperform the off-line framework. This depends on the specific algorithms we derive to solve these problems and, so far, what we have obtained is just an optimal control framework – not a working algorithm of any kind. It remains to obtain core algorithms which can solve Problem 2 and the different candidates for such algorithms will largely influence the performance of the on-line framework. In particular, note that in the on-line framework, Problem 2 must be solved $N$ times during the whole process, in contrast to only once in the off-line one. If Problem 2 cannot be efficiently solved, the energy saved by using the additional arrival time information may not outweigh the energy consumption caused by the extra computation required by the optimization task itself!

In summary, to make full use of the advantage of the on-line framework, the solution method for Problem 2 must be highly efficient.
Table 1: The off-line framework

Step 1: $K = 1$;

Step 2: Compute $u_i^*, \ i = K, ..., N$ of Problem 2;

Step 3: Apply $u_i^*, \ i = K, ..., N$ to the system;

Table 2: The on-line framework

Step 1: $K = 1$; while $K \leq N$ do

Step 2: Compute $u_i^*, \ i = K, ..., N$ of Problem 2;

Step 3: Apply only $u_K^*$ to the system;

Step 4: $K \leftarrow K + 1$

End while.

3.2 Busy Period Decomposition

In this subsection, we address the issue of seeking an efficient solution method for Problem 2 given the remaining two challenges presented in Section 2, i.e., the high dimensionality of the problem and the nondifferentiable constrains included in Problem 2, both of which can cause excessive computational efforts. We take a first step towards this goal by developing an algorithm termed Busy Period Decomposition Algorithm (BPDA).

Our starting point is the same as what led to the development of the FA algorithm in [5] for a class of related optimal control problems, i.e., the observation that every sample path of our queueing model can be decomposed into busy periods. Obtaining the busy period structure of the optimal sample path is, therefore, a key step towards solving Problem 2. The FA algorithm accomplishes this by solving $N$ NLP problems, which we would like to avoid if possible.

Let $x_i^*$ denote the optimal departure time of task $i$ in Problem 2. We then introduce the following definitions:
Definition 1 A busy period (BP) on an optimal sample path is a contiguous set of tasks \( \{k, \ldots, n\} \) such that the following three conditions are satisfied: \( x_{k-1}^* < \bar{a}_k \), \( x_n^* < \bar{a}_{n+1} \), and \( x_i^* \geq \bar{a}_{i+1} \), for every \( i = k, \ldots, n-1 \).

Definition 2 A busy period structure is a partition of the tasks \( \{1, \ldots, N\} \) into busy periods.

Before identifying a BP structure, we first concentrate on the property of BPs to decompose Problem 2 into a number of separate NLP problems.

Definition 3 \( Q(k, n) \) is a NLP problem with linear constraints which corresponds to a single BP \( \{k, \ldots, n\} \):

\[
Q(k, n) : \min_{u_k, \ldots, u_n} \left\{ J(k, n) = \sum_{i=k}^{n} \theta_i(u_i) \right\} \\
\text{s.t. } x_i = \bar{a}_k + \sum_{j=k}^{i} u_j, \; i = k, \ldots, n; \\
x_i \geq \bar{a}_{i+1}, \; i = k, \ldots, n-1; \\
x_i \leq d_i, \; i = k, \ldots, n; \\
u_i \geq u_{i\min}, \; i = k, \ldots, n.
\]

Decomposing a sample path into BPs results in decoupling the optimal controls for tasks in one BP from those of any other. This allows us to reduce the solution of Problem 2 into the solution of a set of simpler problems of the form \( Q(k, n) \), provided we can identify each BP in terms of its starting and ending tasks, \( k \) and \( n \) respectively. This decomposition property is established in Proposition 1 with the aid of the following three lemmas (the proofs of these lemmas as well as all other proofs in this paper can be found in Appendix).

Lemma 1 If \( d_i < \bar{a}_{i+1} \), then \( x_i^* = d_i \).

Lemma 2 If and only if \( \bar{a}_{i+1} \leq d_i \), then \( \bar{a}_{i+1} \leq x_i^* \).

Lemma 3 If and only if \( d_i < \bar{a}_{i+1} \), then \( x_i^* < \bar{a}_{i+1} \).
Let \( u^*_i(k,n) \) denote the optimal control of task \( i \) in \( Q(k,n) \), whereas \( u^*_i \) is the optimal control of task \( i \) in Problem 2. Proposition 1 establishes the aforementioned decomposition property brought about by the BP structure.

**Proposition 1** If the contiguous tasks \( \{k, ..., n\} \) constitute a single BP in the optimal sample path, i.e., \( x^*_{k-1} < \bar{a}_k, x^*_k < \bar{a}_{n+1}, \) and \( x^*_i \geq \bar{a}_{i+1} \), for every \( i = k, ..., n - 1 \), then \( u^*_i = u^*_i(k,n), i = k, ..., n \).

It immediately follows that the optimal control of Problem 2 can be obtained by solving a set of problems of the form \( Q(k,n) \), depending on the BP structure. Thus, all that remains is to identify the optimal BP structure. The following proposition asserts that we can identify this BP structure without solving any NLP problems; rather, it can be directly identified by simple comparisons of the known \( \bar{a}_{i+1} \) and \( d_i \) for every \( i = 1, ..., N \).

**Proposition 2** Tasks \( \{k, ..., n\} \) constitute a single BP if and only if the following conditions are satisfied: \( \bar{a}_k > d_{k-1}, \bar{a}_{n+1} > d_n, \) and \( \bar{a}_{i+1} \leq d_i \), for each \( i = k, ..., n - 1 \)

Based on this result, we can derive a simple algorithm for identifying the optimal BP structure without the need for solving any NLP problems. All that is required is to carry out simple comparisons involving \( \bar{a}_{i+1} \) and \( d_i \) for \( i = 1, ..., N \). It is worth noting (from Lemma 1) that a BP in the optimal sample path always ends with \( x^*_n = d_n \). Further, we can extend the algorithm to provide a complete solution to Problem 2; the result is termed the Busy Period Decomposition Algorithm (BPDA). As in the previous section, there are two different versions, depending on whether it is used in the on-line or off-line framework. Table 3 and Table 4 are the on-line and off-line version respectively. These are to be contrasted to the FA algorithm in [5], which could also be applied in this setting. Specifically, in the off-line framework, the FA must always solve \( N \) NLP problems to obtain the full solution of Problem 2, while the BPDA needs to solve at most \( N \) ones. In the on-line framework, only the initial BP which contains the current starting task is required to be solved due to the decoupling property of distinct BPs. Assume the initial BP contains \( m \) tasks. In each
iteration, the FA has to solve $m$ NLP problems to obtain the optimal controls for the initial BP, while the BPDA needs to solve only one NLP problem no matter what $m$ is. Therefore, the BPDA can solve Problem 2 much more efficiently than the FA. Nonetheless, the BPDA still needs to solve some NLP problems to obtain the solution of $Q(k, n)$. Thus, although we have exploited some properties of our problem to reduce the computational effort required to solve Problem 2, the resulting effort may still be considerable, especially when the number of tasks contained in a BP is very large.

In the next section, we identify further structural properties that will completely eliminate the need for NLP problem solutions.

<table>
<thead>
<tr>
<th>Table 3: The BPDA (on-line version)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1:</strong> Starting with task $K$, find the first task $M$ such that $d_M &lt; \bar{a}_{M+1}$;</td>
</tr>
<tr>
<td><strong>Step 2:</strong> Compute $u^*_i$, $i = K, ..., M$ by solving $Q(K, M)$;</td>
</tr>
<tr>
<td><strong>Step 3:</strong> Apply only control $u^*_K$ to the system.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4: The BPDA (off-line version)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1:</strong> Starting with task $K$, find all tasks $K_i$, $i = 1, ..., m$ such that $d_{K_i} &lt; \bar{a}<em>{K</em>{i+1}}$;</td>
</tr>
<tr>
<td><strong>Step 2:</strong> Compute $u^*_i$, $i = K, ..., N$ by solving $Q(K, K_1)$, $Q(K_1+1, K_2)$, ..., $Q(K_m+1, N)$;</td>
</tr>
<tr>
<td><strong>Step 3:</strong> Apply all optimal controls to the system.</td>
</tr>
</tbody>
</table>

Before proceeding, it is convenient to rewrite $Q(k, n)$ in a simplified manner based on the following observations. From Lemma 1 and Proposition 2, it follows that

$$x_n = \bar{a}_k + \sum_{i=k}^{n} u_j = d_n$$

Therefore, the constraint $x_n \leq d_n$ in $Q(k, n)$ can be changed to $x_n = d_n$ without affecting the solution. In the special case of a BP containing a single task $n$, the optimal control $u^*_n$ is simply $u^*_n = d_n - \bar{a}_n$. Thus, $Q(k, n)$ is rewritten as follows:
\[ Q(k, n) : \min_{u_k, \ldots, u_n} \left\{ J(k, n) = \sum_{i=k}^{n} \theta_i(u_i) \right\} \]

s.t. \[ x_i = a_k + \sum_{j=k}^{i} u_j, \quad i = k, \ldots, n; \]
\[ a_{i+1} \leq x_i \leq d_i, \quad i = k, \ldots, n - 1; \quad x_n = d_n; \]
\[ u_i \geq u_{i\min}, \quad i = k, \ldots, n. \]

### 3.3 Critical Task Decomposition

In this subsection, we identify an additional decomposition property of the optimal sample path based on identifying certain “critical tasks”. This leads to an algorithm called Critical Task Decomposition Algorithm (CTDA) which can derive the optimal controls of \( Q(k, n) \) without solving any NLP problems.

As mentioned in Section 2, the cost function \( \theta_i(u_i) \) depends on the specifics of task \( i \). However, the form of \( \theta_i(u_i) \) for different tasks is similar. Recall that \( \mu_i \) denotes the number of operations required by task \( i \). Let

\[ \tau_i = \frac{u_i}{\mu_i} \quad \text{and} \quad \theta(\tau_i) = C_1 \left( \frac{V_t \tau_i}{\tau_i - C_2} \right)^2 \]

It follows from (3) that

\[ \theta_i(u_i) = \mu_i \theta(\tau_i) \]

Additionally, recalling the constraint (4) on \( u_i \), let

\[ \tau_{\min} = \frac{C_2 V_{\max}}{V_{\max} - V_t} \]

so that the constraint on \( \tau_i \) becomes

\[ \tau_i \geq \tau_{\min}, \quad i = 1, \ldots, n \]

Thus, given that \( C_1, C_2 \) are processor-dependent constants, the difference across tasks lies in the different number of operations needed, i.e., \( \mu_i \). With this observation in mind, we can modify \( Q(k, n) \) as follows. Let \( \tau_i^* \) and \( x_i^* \) denote the optimal control and optimal departure time of task \( i \) in \( Q(k, n) \) respectively. Obviously, as long as \( \tau_i^* \) can be computed without solving any NLP problems, \( u_i^* \) can be directly determined through \( u_i^* = \mu_i \tau_i^* \) for all \( i = k, \ldots, n \). The modified \( Q(k, n) \) is
\[ Q(k, n) : \min_{\tau_k, \ldots, \tau_n} \left\{ J = \sum_{i=k}^{n} \mu_i \theta(\tau_i) \right\} \]

s.t. \( x_i = \bar{a}_k + \sum_{j=k}^{i} \mu_j \tau_j, \ i = k, \ldots, n; \)
\( \bar{a}_{i+1} \leq x_i \leq d_i, \ i = k, \ldots, n - 1; \ x_n = d_n; \)
\( \tau_i \geq \tau_{\min}, \ i = k, \ldots, n. \)

Let us now define \( \hat{Q}(k, n) \) to be the same as \( Q(k, n) \) except that we remove the constraint \( \tau_i \geq \tau_{\min}, i = k, \ldots, n: \)

\[ \hat{Q}(k, n) : \min_{\tau_k, \ldots, \tau_n} \left\{ J(k, n) = \sum_{i=k}^{n} \mu_i \theta(\tau_i) \right\} \]

s.t. \( x_i = \bar{a}_k + \sum_{j=k}^{i} \mu_j \tau_j, \ i = k, \ldots, n; \)
\( \bar{a}_{i+1} \leq x_i \leq d_i, \ i = k, \ldots, n - 1; \ x_n = d_n. \)

The reason for introducing \( \hat{Q}(k, n) \) is that, despite the removal of the constraint on \( \tau_i, \) this problem turns out to have the same solution as \( Q(k, n) \) as shown next. Let \( \hat{\tau}_i^* \) and \( \hat{x}_i^* \) denote the optimal control and optimal departure time of task \( i \) in \( \hat{Q}(k, n) \) respectively. We first establish that the relative values of \( \hat{\tau}_i^* \) and \( \hat{\tau}_{i+1}^* \) allow us to easily determine the value of \( \hat{x}_i^* \). This will then help us show that \( \tau_i^* = \hat{\tau}_i^* \) in Proposition 4.

**Proposition 3** The solution of \( \hat{Q}(k, n) \) satisfies the following, for all \( i = k, \ldots, n - 1: \)

- If \( \hat{\tau}_i^* > \hat{\tau}_{i+1}^* \), then \( \hat{x}_i^* = \bar{a}_{i+1} \)
- If \( \hat{\tau}_i^* < \hat{\tau}_{i+1}^* \), then \( \hat{x}_i^* = d_i \)

Based on Proposition 3, the following result can be proved.

**Proposition 4** If \( Q(k, n) \) has feasible solutions, then \( \tau_i^* = \hat{\tau}_i^* \) must hold for each \( i = k, \ldots, n. \)

It follows from Proposition 4, that the optimal controls of \( Q(k, n) \) can be derived by solving \( \hat{Q}(k, n). \) Moreover, looking back at Proposition 3, note that the optimal departure time for task \( i \) is given by either the next arrival time \( \bar{a}_{i+1} \) or the deadline \( d_i \).
in the two cases. This implies that if we can detect which of the two cases in Proposition 3 applies to a task at each decision point, then the weighted sum of optimal controls in the cost function $J$ can be derived without solving any NLP problem. For example, if tasks $p$ and $q$ ($k \leq p < q \leq n$) satisfy the two cases in Proposition 3, then $\hat{x}^*_p$ and $\hat{x}^*_q$ can be directly derived. Consequently, $\sum_{i=k}^p \mu_i \hat{\tau}_i = \hat{x}^*_p - \bar{a}_k$ and $\sum_{i=p+1}^q \mu_i \hat{\tau}_i = \hat{x}^*_q - \hat{x}^*_p$.

It should be noted that there still remains the case $\hat{\tau}_i = \hat{\tau}_{i+1}$ to take into account, which is not included in Proposition 3. Fortunately, if all tasks in $\hat{Q}(k, n)$ which satisfy the two cases in Proposition 3 can be identified, then this case is easily handled without the need to solve any NLP problems. As we shall see, the values of $\hat{\tau}_i$ in this case are constants that can be evaluated in terms of $\hat{x}^*_p$ and $\hat{x}^*_q$ in the example above. Specifically, we will show that they are given by

\[
\hat{\tau}_i^* = \begin{cases} \frac{\hat{x}^*_p - \bar{a}_k}{\sum_{i=k}^p \mu_i}, & i = k, \ldots, p; \\ \frac{\hat{x}^*_q - \bar{a}_k}{\sum_{i=p+1}^q \mu_i}, & i = p + 1, \ldots, q; \\ \frac{d_n - \bar{x}_q^*}{\sum_{i=q+1}^n \mu_i}, & i = q + 1, \ldots, n. \end{cases}
\]

To prove this fact and develop the CTDA mentioned above, we begin by defining the concept of a “block” of tasks, as well as the concept of “left-critical” and “right-critical” tasks in what follows.

**Definition 4** A **block** in a BP $\{k, \ldots, n\}$ of an optimal sample path is a contiguous set of tasks $p, \ldots, q$ ($k \leq p \leq q \leq n$) such that $\hat{\tau}_i = \hat{\tau}_j$ for all $i, j \in [p, q]$.

**Definition 5** If $\hat{\tau}_i^* \neq \hat{\tau}_{i+1}^*$, task $i$ is **critical**. If $\hat{\tau}_i^* > \hat{\tau}_{i+1}^*$, then task $i$ is **left-critical**. If $\hat{\tau}_i^* < \hat{\tau}_{i+1}^*$, then task $i$ is **right-critical**.

It can be easily seen that blocks are separated by critical tasks. Based on the analysis above, computing all optimal controls within a BP is equivalent to detecting all critical tasks (if any) within this BP. To meet this goal, we need to define a subproblem of $\hat{Q}(k, n)$ in which we concentrate on tasks $\{p, \ldots, q\}$ ($k \leq p \leq q \leq n$) conditioned upon knowledge of $\hat{x}^*_{p-1}$ and $\hat{x}^*_q$.
Definition 6 Let \( \{k, \ldots, n\} \) be a BP of an optimal sample path and \( k \leq p \leq q \leq n \). Problem \( C(p, q) \) is defined as

\[
C(p, q) : \min_{u_p, \ldots, u_q} \left\{ J(p, q) = \sum_{i=p}^{q} \mu_i \theta(\tau_i) \right\}
\]

\[
\text{s.t. } g_i(p, q) \leq \sum_{j=p}^{q} \mu_j \tau_j \leq h_i(p, q), ~ i = p, \ldots, q - 1;
\]

\[
\sum_{j=p}^{q} \mu_j \tau_j = g_q(p, q) = h_q(p, q).
\]

where

\[
g_i(p, q) = \begin{cases} 
\tilde{a}_{i+1} - B(p) & i < q \\
E(q) - B(p) & i = q 
\end{cases} 
\]

\[
h_i(p, q) = \begin{cases} 
d_i - B(p) & i < q \\
E(q) - B(p) & i = q 
\end{cases} 
\]

\[
B(p) = \begin{cases} 
\hat{x}_{p-1}^s & p > k \\
\tilde{a}_k & p = k 
\end{cases}
\]

Note that \( g_i(p, q) \) and \( h_i(p, q) \) are defined so that \( C(p, q) \) is identical to \( \hat{Q}(k, n) \) with \( g_i(k, n) = \tilde{a}_{i+1} - \tilde{a}_k \), \( h_i(k, n) = d_i - \tilde{a}_k \) for \( i = k, \ldots, n - 1 \), and \( g_n(k, n) = h_n(k, n) = d_n - \tilde{a}_k \). The following result relates the solution of \( \hat{Q}(k, n) \) to the solution of problems of the form \( C(p, q) \).

Proposition 5 Let \( \hat{\tau}^*_i(p, q) \) denote the optimal controls in \( C(p, q) \). Then, \( \hat{\tau}^*_i = \hat{\tau}^*_i(p, q) \) must hold for each \( i = p, \ldots, q \).

Based on Proposition 5, the critical tasks in \( C(p, n) \) must also be critical in \( \hat{Q}(k, n) \) and vice versa. This property greatly facilitates the detection of critical tasks in \( \hat{Q}(k, n) \) by making this process equivalent to identifying the first critical task in \( C(p, n) \). Let the whole optimization process begin with \( p = k \) so that \( \hat{Q}(k, n) \) is just \( C(k, n) \). In the first iteration, we identify the first critical task \( p_1 \) in \( C(k, n) \). From the point above, task \( p_1 \) must be a critical task in \( \hat{Q}(k, n) \) and tasks \( k, \ldots, p_1 - 1 \) must not be critical. From Proposition 3, \( \hat{\tau}^*_i \) can be directly determined as either \( \tilde{a}_{p_1+1} \) or \( d_{p_1} \) and \( \hat{\tau}^*_i = (\hat{x}^*_i - \tilde{a}_k) / \sum_{i=k}^{p_1} \mu_i, \ i = k, \ldots, p_1 \). In the second iteration, the problem reduces to \( C(p_1, n) \). We identify the first critical task \( p_2 \) in \( C(p_1 + 1, n) \). Then, \( \hat{\tau}^*_i \) can be similarly obtained and \( \hat{\tau}^*_i = (\hat{x}^*_i - \hat{x}^*_{p_1}) / \sum_{i=p_1+1}^{p_2} \mu_i, \ i = p_1 + 1, \ldots, p_2 \). In the next iteration, the problem reduces to \( C(p_2 + 1, n) \) and the process repeats until all optimal
controls in $\hat{Q}(k,n)$ are derived. This process rests on the main result of this section, Proposition 6, where it is shown how to determine the first critical task in $C(p,n)$ and how to obtain a complete solution of $\hat{Q}(k,n)$ without solving any NLP problems. To present this result, we first establish some convenient notation. Define:

$$G_i(p,q) = \frac{g_i(p,q)}{\sum_{j=p}^{i} \mu_j}, \quad H_i(p,q) = \frac{h_i(p,q)}{\sum_{j=p}^{i} \mu_j}$$  \hspace{1cm} (7)$$

$$R_i = \max_{s} \{ s | H_s(p,n) \leq H_j(p,n), \forall j \in [p, i-1] \}$$

$$L_i = \max_{s} \{ s | G_s(p,n) \geq G_j(p,n), \forall j \in [p, i-1] \}$$

**Proposition 6** Consider the problem $C(p,n)$ and suppose that $G_i(p,n) \leq H_R_i(p,n)$ and $H_i(p,n) \geq G_L_i(p,n)$ for all $i = p, \ldots, m-1$, where $p \leq m \leq n$:

- If $G_m(p,n) > H_R_m(p,n)$, then task $R_m$ is a right-critical task and also the first critical task in $C(p,n)$. Moreover, $\hat{\tau}_i^* = H_R_m(p,n)$, for all $i = p, \ldots, R_m$.

- If $H_m(p,n) < G_L_m(p,n)$, then task $L_m$ is a left-critical task and also the first critical task in $C(p,n)$. Moreover, $\hat{\tau}_i^* = G_L_m(p,n)$, for all $i = p, \ldots, L_m$.

Based on Propositions 3-6, the CTDA is an algorithm developed to solve $\hat{Q}(k,n)$ without solving any NLP problems. The procedure in Table 5 is the off-line version of the CTDA. In the on-line version, only the first critical task in $\hat{Q}(k,n)$ needs to be identified since only the current starting task’s optimal control is required.

The CTDA has three notable advantages compared with algorithms involving NLP problems. Firstly, it has much smaller computational complexity which results in a faster evaluation of the optimal controls. Secondly, it also has smaller space complexity which makes it appealing for applications involving devices with limited memory. Thirdly, it is independent of the details of the energy function, which implies that there is no need to measure parameters such as $C_1$ or $C_2$ in (3). Therefore, it is suitable for use in real-time embedded systems.

The combination of the BPDA and the CTDA provides a complete solution of problem 2 without solving any NLP problems. The former is used to determine the BP structure and the latter gives the optimal controls within each BP identified.
Table 5: The CTDA (off-line version)

**Step 1:** \( p = k \);

**Step 2:** Compute \( g_i(p,n), h_i(p,n) \) for \( i = p, ..., n \) in \( C(p,n) \)

**Step 3:** Identify the first critical task in \( C(p,n) \)

\[
i = p + 1;
\]

while \( i \leq n \) {

Compute \( R_i \);

if \( (G_i(p,n) > H_{R_i}(p,n)) \)

\( R_i \) is the first critical task in \( C(p,n) \), and right-critical;

\[
\hat{\tau}^*_j = H_{R_i}(p,n), \quad u^*_j = \mu_jH_{R_i}(p,n), \quad j = p, ..., R_i;
\]

\[
\hat{x}^*_{R_i} = d_{R_i}; \quad p = R_i + 1; \quad \text{goto Step 2};
\]

Compute \( L_i \);

if \( (H_i(p,n) < G_{L_i}(p,n)) \)

\( L_i \) is the first critical task in \( C(p,n) \), and left-critical;

\[
\hat{\tau}^*_j = G_{L_i}(p,n), \quad u^*_j = \mu_jG_{L_i}(p,n), \quad j = p, ..., L_i;
\]

\[
\hat{x}^*_{L_i} = \bar{a}_{L_i+1}; \quad p = L_i + 1; \quad \text{goto Step 2};
\]

\[
i = i + 1; \}
\]

\[
\hat{\tau}^*_j = G_n(p,n), \quad u^*_j = \mu_jG_n(p,n), \quad j = p, ..., n; \quad \text{END}
\]
4 Numerical Examples

Consider the problem

\[
\min_{\mathbf{u}_1, \ldots, \mathbf{u}_N} \left\{ J = \sum_{i=1}^{N} \theta_i (u_i) = \sum_{i=1}^{N} \mu_i C_1 \left( \frac{V_i u_i}{u_i - \mu_i C_2} \right)^2 \right\}
\]

s.t. \( u_i \geq u_{i\text{min}} = \frac{\mu_i C_2 V_{\text{max}}}{V_{\text{max}} - V_i} \), \( i = 1, \ldots, N \); \( x_0 = 0 \);

\( x_i = \max(x_{i-1}, a_i) + u_k \leq d_i \), \( i = 1, \ldots, N \);

\( a_i \in [a_i^-, a_i^+] \), \( a_i^+ < a_{i+1}^- \), \( i = 1, \ldots, N \);

\( V_{\text{max}} = 5, \; V_i = 1, \; C_1 = 1, \; C_2 = 0.1 \).

The parameter values selected are motivated by CMOS microprocessor power consumption data. We will assume that there is a total of 50 tasks to be processed, i.e., \( N = 50 \). In the simulated system operation, task arrivals are randomly generated within intervals \([a_i^-, a_i^+]\), \( i = 1, \ldots, 50 \) whose size can be varied (see Table 7). Each task is also randomly assigned a number of operations \( \mu_i \) uniformly distributed over \([10, 60]\) and a deadline \( d_i \) using a randomization scheme such that \( d_i \geq a_{i+1}^- \) with probability \( p \) and \( d_i < a_{i+1}^- \) otherwise. \( p \) can be used to control the shape of the BP structure, i.e., the average number of tasks contained in a BP which equals to \( \frac{1}{1-p} \) approximately.

4.1 FA vs BPDA

As already mentioned, the performance of an on-line DVS algorithm is largely influenced by the computational complexity of the core algorithm used to solve Problem 2. In this example, our first goal is to illustrate and compare this influence over three choices for the core DVS algorithm: (i) The Forward Algorithm (FA) from [5] which requires solving \( N \) NLP problems, (ii) The BPDA without the help of the CTDA (labeled as BPDA), i.e., by solving NLP problems to determine the optimal controls within every BP, and (iii) The combination of BPDA and CTDA (labeled as BCDA).

These three algorithms are applied over 50 simulation runs with the same random seeds in an on-line and an off-line setting respectively for different \( p \). Complexity is
measured as the average number of NLPs required to solve. The results are shown in Table 6.

Table 6: Complexity comparison

<table>
<thead>
<tr>
<th>$p$</th>
<th>Off-line framework</th>
<th>On-line framework</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FA</td>
<td>BPDA</td>
</tr>
<tr>
<td>0.0</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>0.3</td>
<td>50</td>
<td>35.34</td>
</tr>
<tr>
<td>0.6</td>
<td>50</td>
<td>20.52</td>
</tr>
<tr>
<td>0.8</td>
<td>50</td>
<td>10.84</td>
</tr>
<tr>
<td>0.9</td>
<td>50</td>
<td>6.08</td>
</tr>
<tr>
<td>1.0</td>
<td>50</td>
<td>1</td>
</tr>
</tbody>
</table>

From Table 6, we can see that the computational burden of the FA increases dramatically in going from the off-line to the on-line setting. It grows even more when $p$ increases. In contrast, the BPDA always solves only 50 NLP problems no matter what $p$ is, since it needs to solve only one NLP at each decision point (for a total of 50 decision points). In the BCDA case, no NLP needs to be solved. This makes both BPDA and BCDA very attractive candidates for DVS in an on-line framework.

4.2 On line vs Off line

In this subsection, we explore the improvement provided by applying DVS on line as opposed to off line. The “ideal” method acts as the common lower bound, in which the optimal controls are computed based on knowing all actual arrival times. We denote by $\lambda$ the improvement of the on-line algorithm, defined as $\lambda = \frac{C_{\text{off}} - C_{\text{on}}}{C_{\text{off}} - C_{\text{idl}}}$, where $C_{\text{off}}$ is the cost of the off-line case, $C_{\text{on}}$ is the cost of the on-line case, and $C_{\text{idl}}$ is the cost of the ideal method.

The effect of the size of $[a_i^-, a_i^+]$ to the improvement is shown in Table 7. For
each size of \([a_i^-, a_i^+]\), three methods are applied over 50 simulation runs with the same random seeds by using \(p = 0.6\). As seen in the table, the on-line algorithm provides substantial cost reduction over its off-line counterpart. Moreover, the improvement increases along with an increase in the size of \([a_i^-, a_i^+]\), i.e., an increase in the arrival time uncertainty.

<table>
<thead>
<tr>
<th>([a_i^-, a_i^+])</th>
<th>(C_{off}(\times 10^4))</th>
<th>(C_{on}(\times 10^4))</th>
<th>(C_{id}(\times 10^4))</th>
<th>(\lambda)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.35</td>
<td>1.5325</td>
<td>1.5074</td>
<td>1.4303</td>
<td>24.59%</td>
</tr>
<tr>
<td>0.70</td>
<td>1.5325</td>
<td>1.4827</td>
<td>1.3737</td>
<td>31.33%</td>
</tr>
<tr>
<td>1.05</td>
<td>1.5325</td>
<td>1.4574</td>
<td>1.3389</td>
<td>38.81%</td>
</tr>
<tr>
<td>1.40</td>
<td>1.5325</td>
<td>1.4389</td>
<td>1.3153</td>
<td>43.09%</td>
</tr>
</tbody>
</table>

Figure 2: A typical sample path when DVS is applied off line

Figures 2 and 3 show typical sample paths of the off-line and the on-line framework respectively. One can see that the on-line algorithm can make better use of available time intervals in order to save more energy.
Figure 3: A typical sample path when DVS is applied on line

5 Conclusions

The problem formulated and solved in this paper is motivated from low power systems with hard real-time nonpreemptive and aperiodic tasks, in which the actual arrival time of each task is not known a priori. We have developed a DVS algorithm that minimizes energy consumption while guaranteeing that all tasks meet their deadlines. The main properties of this algorithm are: (i) it involves no NLP problems, and (ii) it depends solely on identifying busy periods (through its BPDA component) and "critical tasks" (through its CTDA component) on an optimal sample path using simple computations and minimal memory. The algorithm may be applied both off line or on line. In the latter case, all past observed arrival time information is used, which results in additional cost savings over its off-line counterpart.

The development of such an efficient algorithm paves the way for a variety of natural extensions. For example, we are currently investigating the effect of making decisions at task arrival times, as opposed to task departure times, which should intuitively provide additional opportunities for cost reduction in the on-line setting. Future work is aiming at incorporating additional uncertainty factors such as uncertain deadlines and task processing time, as well as systems that process tasks over multiple stages.
References


Appendix

Proof: [Lemma 1] Assume on the contrary that there exists an optimal control $u_i^*$ such that $x_i^* \neq d_i$. This implies that $x_i^* < d_i$ since $x_i^* > d_i$ is not feasible. It follows that there must exist some $u'_i > u_i^*$ such that $x'_i = d_i$. Since $d_i < \bar{a}_{i+1}$, it follows from $x_{i+1} = \max(x_i, \bar{a}_{i+1}) + u_{i+1}$ that this increase from $u_i^*$ to $u'_i$ will not affect any other control. Obviously, $\theta(u'_i) < \theta(u_i^*)$ since $\theta(u_i)$ is monotonically decreasing. This contradicts the optimality of $u_i^*$.

Proof: [Lemma 2] Sufficient condition: Assume on the contrary that $\bar{a}_{i+1} > x_i^*$ under the optimal control $u_i^*$. Then, there must exist some $u'_i > u_i^*$ such that $x'_i = \bar{a}_{i+1}$. It follows from $x_{i+1} = \max(x_i, \bar{a}_{i+1}) + u_{i+1}$ that this increase from $u_i^*$ to $u'_i$ does
not affect any other control. Obviously, \( \theta(u'_i) < \theta(u^*_i) \) since \( \theta(u_i) \) is monotonically decreasing. This contradicts the optimality of \( u^*_i \). Thus \( \bar{a}_{i+1} \leq x^*_i \) must hold.

**Necessary condition:** If \( \bar{a}_{i+1} \leq x^*_i \) and \( x^*_i \leq d_i \) (from the feasibility constraint), it follows that \( \bar{a}_{i+1} \leq d_i \).

**Proof:** [Lemma 3] By using contrapositivity, Lemma 3 is equivalent to Lemma 2, and the proof is complete.

**Proof:** [Proposition 1] Without loss of generality, assume \( k > K, n < N \). There must exist a BP preceding the current one and also a BP following the current one, i.e., \( x^*_{k-1} < \bar{a}_k \) and \( x^*_n < \bar{a}_{n+1} \).

From Lemma 3:

\[
d_k - 1 < \bar{a}_k, \quad d_n < \bar{a}_{n+1}
\]

From the Lindley equation \( x_k = \max(x_{k-1}, \bar{a}_k) + u_k \) and the constraint \( x_{k-1} \leq d_{k-1} \), we know

\[
x_k = \bar{a}_k + u_k
\]

Hence, the controls of tasks \( k, \ldots, N \) are independent of the control of tasks \( 1, \ldots, k-1 \). Similarly, the control of tasks \( K, \ldots, n \) are independent of the control of tasks \( n+1, \ldots, N \). Therefore, the controls of tasks \( k, \ldots, n \) are independent of those of other tasks.

In addition, by noting that the cost function in Problem 2 can be written as

\[
J = \sum_{i=K}^{N} \theta_i(u_i) = \sum_{i=k}^{n} \theta_i(u_i) + \left[ \sum_{i=K}^{k-1} \theta_i(u_i) + \sum_{i=n+1}^{N} \theta_i(u_i) \right]
\]

we can conclude that \( \sum_{i=k}^{n} \theta_i(u_i) \) must be optimized when \( J \) is optimized. Therefore, the optimal control \( u^*_i \) and the optimal departure time \( x^*_i \) for every \( i = k, \ldots, n \) of Problem 2 must equal to the optimal control and departure time of the problem below.

\[
\min_{u_k, \ldots, u_n} \left\{ J = \sum_{i=k}^{n} \theta_i(u_i) : u_i \geq u_{\min}, i = k, \ldots, n \right\}
\]

s.t. \( x_i = \max(x_{i-1}, \bar{a}_i) + u_i \leq d_i, \quad x_0 = 0, \quad i = k, \ldots, n. \)
Furthermore, since by assumption $x^*_i \geq a_{i+1}, i = k, \ldots, n-1$, the optimal $i$th departure time of the problem above is also no smaller than $a_{i+1}$, for every $i = k, \ldots, n-1$. Thus, the constraint above can be rewritten as

$$x_0 = 0; \quad x_i = a_i + \sum_{j=k}^{i} u_j \leq d_i, \quad i = k, \ldots, n; \quad x_i \geq a_{i+1}, \quad i = k, \ldots, n-1.$$ 

Therefore, the problem above is equivalent to the problem $Q(k, n)$ and it follows that $u^*_i = u^*_i (k, n), i = k, \ldots, n$. \hfill \blacksquare

**Proof:** [Proposition 2] From Lemmas 2 and 3,

$$\bar{a}_k > d_{k-1}, \quad \bar{a}_{n+1} > d_n, \quad \bar{a}_{i+1} \leq d_i, \quad i = k, \ldots, n-1 \iff x^*_k < \bar{a}_k, \quad x^*_n < \bar{a}_{n+1}, \quad x^*_i \geq \bar{a}_{i+1}, \quad i = k, \ldots, n-1$$

From the definition of a BP (Definition 1), the result immediately follows. \hfill \blacksquare

Before proving Proposition 3, we establish the following lemma:

**Lemma 4** If $\mu_1 \tau_1 + \mu_2 \tau_2 = \mu_1 \hat{\tau}_1 + \mu_2 \hat{\tau}_2 = C$, $\hat{\tau}_1 > \tau_1 \geq \tau_2 > \hat{\tau}_2$, and the function $\theta (\tau_i)$ is strictly convex and differentiable, then $\mu_1 \theta (\tau_1) + \mu_2 \theta (\tau_2) < \mu_1 \theta (\hat{\tau}_1) + \mu_2 \theta (\hat{\tau}_2)$.

**Proof:** Since $\theta (\tau_i)$ is strictly convex and differentiable,

$$\mu_1 \theta (\hat{\tau}_1) - \mu_1 \theta (\tau_1) > \mu_1 \theta' (\tau_1) (\hat{\tau}_1 - \tau_1), \quad \mu_2 \theta (\hat{\tau}_2) - \mu_2 \theta (\tau_2) > \mu_2 \theta' (\tau_2) (\hat{\tau}_2 - \tau_2) \quad (8)$$

From (8) and the assumption $\mu_1 (\hat{\tau}_1 - \tau_1) = -\mu_2 (\hat{\tau}_2 - \tau_2)$,

$$\mu_1 \theta (\hat{\tau}_1) + \mu_2 \theta (\hat{\tau}_2) - (\mu_1 \theta (\tau_1) + \mu_2 \theta (\tau_2)) > \mu_1 (\theta' (\tau_1) - \theta' (\tau_2)) (\hat{\tau}_1 - \tau_1) \quad (9)$$

Again, since $\theta (\tau_i)$ is strictly convex and differentiable,

$$\theta (\tau_1) - \theta (\tau_2) > \theta' (\tau_2) (\tau_1 - \tau_2), \quad \theta (\tau_2) - \theta (\tau_1) > \theta' (\tau_1) (\tau_2 - \tau_1)$$

which implies:

$$0 > (\theta' (\tau_2) - \theta' (\tau_1)) (\tau_1 - \tau_2) \quad (10)$$

From (10), and $\tau_1 \geq \tau_2$ (by assumption),

$$\theta' (\tau_1) - \theta' (\tau_2) \geq 0 \quad (11)$$

Combining (9) and (11) yields the result $\mu_1 \theta (\tau_1) + \mu_2 \theta (\tau_2) < \mu_1 \theta (\hat{\tau}_1) + \mu_2 \theta (\hat{\tau}_2)$. \hfill \blacksquare
Proof: [Proposition 3] To prove the first assertion, assume on the contrary that \( \hat{x}_i^* > \bar{a}_{i+1} \) when \( \hat{\tau}_i^* > \hat{\tau}_{i+1}^* \), where \( k \leq i < n \) (\( \hat{x}_i^* < \bar{a}_{i+1} \) is not possible since \( i \) is within a BP). Let \( x_i' \in [\bar{a}_{i+1}, \hat{x}_i^*] \). There must exist a feasible solution

\[
\tau_j = \begin{cases} 
\tau'_j & j = i, i+1 \\
\hat{\tau}_j^* & j \neq i, i+1 
\end{cases}
\]

such that

\[
\mu_i \tau'_i = \mu_i \hat{\tau}_i^* - (\hat{x}_i^* - x_i') , \quad \mu_{i+1} \tau'_{i+1} = \mu_{i+1} \hat{\tau}_{i+1}^* + (\hat{x}_i^* - x_i')
\]

which implies:

\[
\mu_i \tau'_i + \mu_{i+1} \tau'_{i+1} = \mu_i \hat{\tau}_i^* + \mu_{i+1} \hat{\tau}_{i+1}^*
\]

(13)

Since, by assumption, \( \hat{\tau}_i^* > \hat{\tau}_{i+1}^* \), there must exist some \( x_i' \) such that

\[
\hat{\tau}_{i+1}^* < \tau'_i \leq \tau_i^* < \hat{\tau}_i^* 
\]

(14)

From (14), (13) and Lemma 4,

\[
\mu_i \theta (\hat{\tau}_i^*) + \mu_{i+1} \theta (\hat{\tau}_{i+1}^*) > \mu_i \theta (\tau'_i) + \mu_{i+1} \theta (\tau'_{i+1})
\]

(15)

which implies that the control \( \tau_j \) in (12) results in a better performance than \( \hat{\tau}_j^* \), \( j = k, ..., n \). This contradicts the optimality of \( \hat{\tau}_j^* \), \( j = k, ..., n \). Therefore, \( \hat{x}_i^* = \bar{a}_{i+1} \).

The second assertion can be proved by a similar argument.

Proof: [Proposition 4] First, we prove that if \( Q(k, n) \) has feasible solutions, then \( \hat{\tau}_i^* \geq \tau_{\min} \), \( i = k, ..., n \) through a contradiction argument. Without loss of generality, assume that if \( Q(k, n) \) has feasible solutions then there exist tasks \( p, q \) (\( k < p \leq q < n \)) such that

\[
\hat{\tau}_{p-1}^* \geq \tau_{\min} , \quad \hat{\tau}_{q+1}^* \geq \tau_{\min} \\
\hat{\tau}_i^* < \tau_{\min} , \quad i = p, ..., q
\]

(16)

From (16) and Proposition 3,

\[
\hat{x}_{p-1} = \bar{a}_p , \quad \hat{x}_q^* = d_q
\]

(17)
Since \( \{k, \ldots, n\} \) in \( Q(k,n) \) is a BP, for any feasible solution of \( Q(k,n) \) we have \( \bar{a}_p \leq x_{p-1} \) and \( x_q = x_{p-1} + \sum_{i=p}^q \mu_i \tau_i \leq d_q \). Hence,

\[
\sum_{i=p}^q \mu_i \tau_i \leq d_q - \bar{a}_p
\]

Similarly, the solution of \( \hat{Q}(k,n) \) satisfies \( \hat{x}_q^* = x_{p-1} + \sum_{i=p}^q \mu_i \hat{\tau}_i \), so that

\[
\sum_{i=p}^q \mu_i \hat{\tau}_i \leq \hat{x}_q^* - x_{p-1}
\]

Combining the last two equations with (17), gives

\[
\sum_{i=p}^q \mu_i \tau_i \leq \sum_{i=p}^q \mu_i \hat{\tau}_i^*
\]  \( \text{(18)} \)

From (16) and (18),

\[
\sum_{i=p}^q \mu_i \tau_i < \tau_{\min} \sum_{i=p}^q \mu_i
\]

This implies that there must always exist at least one task \( i \) \( (p < i < q) \), such that \( \tau_i < \tau_{\min} \). This contradicts our assumption that \( \tau_i \) is feasible in \( Q(k,n) \). Therefore, if \( Q(k,n) \) has feasible solutions, then \( \hat{\tau}_i^* \geq \tau_{\min}, i = k, \ldots, n \).

Moreover, since \( \hat{Q}(k,n) \) is identical to \( Q(k,n) \) except for the constraint \( \tau_i \geq \tau_{\min} \), \( \sum_{i=k}^n \mu_i \theta(\hat{\tau}_i^*) \) is the lower bound of \( Q(k,n) \). If \( \hat{\tau}_i^* \geq \tau_{\min} \), \( i = k, \ldots, n \), then \( \tau_i^* = \hat{\tau}_i^* \) must hold for each \( i = k, \ldots, n \). Together with the result above, we conclude that if \( Q(k,n) \) has feasible solutions, \( \tau_i^* = \hat{\tau}_i^* \) must hold for each \( i = k, \ldots, n \).

**Proof:** [Proposition 5] The proof can be divided into three cases: 1. \( k < p \leq q < n \); 2. \( k = p \leq q < n \); 3. \( k < p \leq q = n \). Since cases 2 and 3 can be proved just like case 1, only case 1 is proved below.

Since \( \hat{x}_p^* = \bar{a}_k + \sum_{i=k}^p \mu_i \hat{\tau}_i^* \), \( \hat{x}_q^* = \bar{a}_k + \sum_{i=k}^q \mu_i \hat{\tau}_i^* \) and \( \hat{x}_p^*, \hat{x}_q^* \) are part of the solution of \( C(p,q) \), we can rewrite \( \hat{Q}(k,n) \) as follows:

\[
\hat{Q}(k,n) : \min_{\tau_k, \ldots, \tau_n} \left\{ J(k,n) = \sum_{i=k}^n \mu_i \theta(\tau_i) \right\}
\]

s.t. \( \bar{a}_{i+1} = \bar{a}_k + \sum_{j=k}^i \mu_j \tau_j \leq d_i, i = k, \ldots, p-1 \), \( \bar{a}_k + \sum_{j=k}^p \mu_j \tau_j = \hat{x}_p^* \)

\( \bar{a}_{i+1} = \hat{x}_p^* + \sum_{j=p+1}^i \mu_j \tau_j \leq d_i, i = p+1, \ldots, q-1 \), \( \hat{x}_p^* + \sum_{j=p+1}^q \mu_j \tau_j = \hat{x}_q^* \)

\( \bar{a}_{i+1} = \hat{x}_q^* + \sum_{j=q+1}^i \mu_j \tau_j \leq d_i, i = q+1, \ldots, n-1 \), \( \hat{x}_q^* + \sum_{j=q+1}^n \mu_j \tau_j = d_n \)
Let \( \hat{\tau}^*_i(p + 1, q) \) be the optimal control of \( C(p + 1, q) \):

\[
C(p + 1, q) : \min_{\tau_{p+1}, \ldots, \tau_q} \left\{ J(p, q) = \sum_{i=p+1}^q \mu_i \theta(\tau_i) \right\}
\]

s.t. \( \bar{a}_{i+1} \leq \hat{x}^*_p + \sum_{j=p+1}^i \mu_j \tau_j \leq d_i, i = p + 1, \ldots, q - 1 \)

\[
\hat{x}^*_p + \sum_{j=p+1}^q \mu_j \tau_j = \hat{x}^*_q
\]

Now, assume there exists some \( i \) such that \( \hat{\tau}^*_i \neq \hat{\tau}^*_i(p + 1, q) \). Then,

\[
\sum_{i=p+1}^q \mu_i \theta(\hat{\tau}^*_i) > \sum_{i=p+1}^q \mu_i \theta(\hat{\tau}^*_i(p + 1, q)) \quad (19)
\]

Since \( \hat{x}^*_p \) and \( \hat{x}^*_q \) are known, the controls \( \{\tau_i, i = k, \ldots, p\} \) and \( \{\tau_i, i = q + 1, \ldots, n\} \) are decoupled from the controls \( \{\tau_i, i = k, \ldots, n\} \). The following controls must be feasible:

\[
\tau_i = \begin{cases} 
\hat{\tau}^*_i & i < p + 1, i > q \\
\hat{\tau}^*_i(p + 1, q) & p + 1 \leq i \leq q
\end{cases}
\]

Using (19), we have

\[
\sum_{i=k}^n \mu_i \theta(\hat{\tau}^*_i) > \sum_{i=k}^p \mu_i \theta(\hat{\tau}^*_i) + \sum_{i=p+1}^n \mu_i \theta(\hat{\tau}^*_i(p + 1, q)) + \sum_{i=p+1}^q \mu_i \theta(\hat{\tau}^*_i(p + 1, q))
\]

which contradicts the optimality of \( \hat{\tau}^*_i, i = k, \ldots, n \). Thus, \( \hat{\tau}^*_i = \hat{\tau}^*_i(p + 1, q), i = p + 1, \ldots, q \).

In order to prove Proposition 6, we need the following lemma:

**Lemma 5** If \( \theta(\tau_i) \) is a strictly convex and differentiable function, then the optimal solution of

\[
\min_{\tau_1, \ldots, \tau_n} \left\{ J = \sum_{i=1}^n \mu_i \theta(\tau_i) \right\} \quad \text{s.t.} \sum_{i=1}^n \mu_i \tau_i = C
\]

is

\[
\tau_i^* = \frac{C}{\sum_{i=1}^n \mu_i}, \quad i = 1, \ldots, n
\]

**Proof:** Let \( P = \sum_{i=1}^n \mu_i \tau_i - C \) and adjoin \( P \) to the cost function using a Lagrange multiplier \( \lambda \). The necessary condition for optimality is \( \nabla J + \lambda \cdot \nabla P = 0 \), i.e.,

\[
\mu_i \theta'(u_i) + \mu_i \lambda = 0
\]

29
Therefore, $\theta'(\tau_i) = -\lambda$ must hold for each $i = 1, \ldots, n$. In addition, since $\theta(\tau_i)$ is strictly convex and differentiable, $\tau_1^* = \tau_2^* = \cdots = \tau_n^*$. Therefore, $\tau_i^* = \frac{C}{\sum_{i=1}^n \mu_i}$ must hold for each $i = 1, \ldots, n$.

**Proof:** [Proposition 6] We will prove the first assertion only, since the second one is similarly shown. Since the proof is somewhat lengthy, we divide it into four parts.

**Part 1:** We establish the following two inequalities:

\begin{align*}
G_i(p,n) &\leq H_{R_m}(p,n) \leq H_i(p,n), \quad i = p, \ldots, R_m - 1 \quad (20) \\
G_i(p,n) &\leq H_{R_m}(p,n) < H_i(p,n), \quad i = R_m + 1, \ldots, m - 1 \quad (21)
\end{align*}

By definition, $R_m = \max_s \{ s \mid H_s(p,n) \leq H_j(p,n), \forall j \in [p, m-1] \}$, so that

\begin{equation}
H_{R_m}(p,n) \leq H_i(p,n), \quad i = p, \ldots, R_m - 1 \quad (22)
\end{equation}

Observe that if $H_{R_m} = H_{R_m+j}$ for some $j = 1, 2, \ldots$, then $R_m$ would have to be replaced by $R_m + j$. Therefore,

\begin{equation}
H_{R_m}(p,n) < H_i(p,n), \quad i = R_m + 1, \ldots, m - 1 \quad (23)
\end{equation}

From (23), for $j \leq m - 1$,

\begin{equation}
H_{R_m}(p,n) < H_i(p,n), \quad i = R_m + 1, \ldots, j - 1 \quad (24)
\end{equation}

From (22), (24) and the fact that $R_j = \max_s \{ s \mid H_s(p,n) \leq H_j(p,n), \forall j \in [p, j-1] \}$,

\begin{equation}
R_j = R_m, \quad j = R_m + 1, \ldots, m - 1 \quad (25)
\end{equation}

By assumption, $G_i(p,n) \leq H_{R_i}(p,n)$, for all $i = R_m + 1, \ldots, m - 1$. Thus, using (25), we get

\begin{equation}
G_i(p,n) \leq H_{R_m}(p,n), \quad i = R_m + 1, \ldots, m - 1 \quad (26)
\end{equation}

Combining (23) and (26), yields inequality (21).

Next, by definition, $L_{R_m} = \max_s \{ s \mid G_s(p,n) \geq G_j(p,n), \forall j \in [p, R_m - 1] \}$, so that

\begin{equation}
G_{L_{R_m}}(p,n) \geq G_j(p,n), \quad j = p, \ldots, R_m - 1 \quad (27)
\end{equation}
By assumption, \( H_{R_m}(p, n) \geq G_{L_{R_m}}(p, n) \), hence (27) gives

\[
G_i(p, n) \leq H_{R_m}(p, n), \quad i = p, \ldots, R_m - 1 \tag{28}
\]

Combining (22) and (28) yields inequality (20).

**Part 2:** We shall now prove the inequality

\[
\hat{\tau}^*_{R_m} \leq H_{R_m}(p, n) \tag{29}
\]

Based on Proposition 5, \( \hat{\tau}^*_i(p, n) = \hat{\tau}^*_i \). For notational simplicity, we use \( \hat{\tau}^*_i \) to denote *both* the optimal control of \( C(p, n) \) and of \( \hat{Q}(k, n) \). By the definition of \( C(p, n) \) in Definition 6,

\[
\sum_{j=p}^{R_m} \mu_j \hat{\tau}^*_j \leq h_{R_m}(p, n) \tag{30}
\]

From (30) and (7),

\[
\sum_{j=p}^{R_m} \mu_i \hat{\tau}^*_j \leq H_{R_m}(p, n) \sum_{j=p}^{R_m} \mu_j \tag{31}
\]

We will now proceed with a contradiction argument. Assume \( \hat{\tau}^*_{R_m} > H_{R_m}(p, n) \). Then, from (31), there must exist \( s, t \ (p \leq s \leq t < R_m) \) such that

\[
\hat{\tau}^*_i < H_{R_m}(p, n), \quad i = s, \ldots, t
\]

\[
\hat{\tau}^*_{t+1} \geq H_{R_m}(p, n) \quad \text{and} \quad \hat{\tau}^*_{s-1} \geq H_{R_m}(p, n) \tag{32}
\]

When \( s = p \), the last inequality above \( (\hat{\tau}^*_{s-1} \geq H_{R_m}(p, n)) \) can be omitted. Since the proof of the case \( s = p \) is similar to that of \( s > p \), we only prove the case \( s > p \).

From (32)

\[
\hat{\tau}^*_{s-1} > \hat{\tau}^*_s, \quad \hat{\tau}^*_t < \hat{\tau}^*_{t+1}
\]

and using Proposition 3 this implies:

\[
\hat{x}^*_{s-1} = \bar{a}_s, \quad \hat{x}^*_t = d_t \tag{33}
\]

Recall that \( \hat{u}^*_j = \hat{\tau}^*_j \mu_j \). Thus, using the first inequality in (32), we get

\[
\sum_{j=s}^{t} \hat{u}^*_j < H_{R_m}(p, n) \sum_{j=s}^{t} \mu_j \tag{34}
\]
Recall that $\bar{\tau}_s^* = \bar{x}_s^{*} + \sum_{j=s}^t \hat{u}_j^*$. Then, from (33) and (34), it follows that

$$d_t - \bar{a}_s < H_{R_m} (p, n) \sum_{j=s}^t \mu_j$$  \hspace{1cm} (35)

Next, let us make use of the inequality (20) along with the definitions of $G_{s-1} (p, n)$ and $H_t (p, n)$ from (7) to obtain:

$$g_{s-1} (p, n) \leq H_{R_m} (p, n) \sum_{j=p}^{s-1} \mu_j$$

$$h_t (p, n) \geq H_{R_m} (p, n) \sum_{j=p}^t \mu_j$$

It follows that

$$h_t (p, n) - g_{s-1} (p, n) \geq H_{R_m} (p, n) \sum_{j=s}^t \mu_j$$  \hspace{1cm} (36)

From (6), since $t < n$ and $s - 1 < n$, we get

$$h_t (p, n) = d_t - B (p), \quad g_{s-1} (p, n) = \bar{a}_s - B (p)$$

which implies

$$h_t (p, n) - g_{s-1} (p, n) = d_t - \bar{a}_s$$  \hspace{1cm} (37)

Combining (37) and (36), we obtain an inequality that contradicts (35). This establishes inequality (29).

**Part 3:** Next, we prove the inequality

$$\hat{\tau}_{R_m+1}^* > H_{R_m} (p, n)$$  \hspace{1cm} (38)

From the definition of $C(p, n)$ in Definition 6,

$$\sum_{j=p}^m \mu_j \hat{\tau}_j^* \geq g_m (p, n), \quad \sum_{j=p}^{R_m} \mu_j \hat{\tau}_j^* \leq h_{R_m} (p, n)$$

which implies

$$\sum_{j=R_m+1}^m \mu_j \hat{\tau}_j^* \geq g_m (p, n) - h_{R_m} (p, n)$$  \hspace{1cm} (39)

By assumption, $G_m (p, n) > H_{R_m} (p, n)$, and by using (7) we get

$$g_m (p, n) > H_{R_m} (p, n) \sum_{j=p}^m \mu_j = H_{R_m} (p, n) \sum_{j=p}^{R_m} \mu_j + H_{R_m} (p, n) \sum_{j=R_m+1}^m \mu_j$$

$$= h_{R_m} (p, n) + H_{R_m} (p, n) \sum_{j=R_m+1}^m \mu_j$$
which implies
\[ g_m(p, n) - h_{R_m}(p, n) > H_{R_m}(p, n) \sum_{j=R_m+1}^{m} \mu_j \] (40)

From (39) and (40),
\[ \sum_{j=R_m+1}^{m} \mu_j \hat{\tau}_j^* > H_{R_m}(p, n) \sum_{j=R_m+1}^{m} \mu_j \] (41)

When \( m - R_m = 1 \), (38) immediately follows from (41). When \( m - R_m > 1 \), we prove (38) through a contradiction argument. Assume \( \hat{\tau}_{R_m+1}^* \leq H_{R_m}(p, n) \). Then, from (41), there must exist some \( s \) \((R_m + 1 < s \leq m)\) such that
\[ \hat{\tau}_i^* \leq H_{R_m}(p, n), \quad i = R_m + 1, \ldots, s - 1 \] (42)

From (42),
\[ \hat{\tau}_s^* > \hat{\tau}_{s-1}^* \] (43)

From (43) and Proposition 3, we have
\[ \hat{x}_{s-1}^* = d_{s-1} \] (44)

From (42) we also get
\[ \sum_{j=R_m+1}^{s-1} \mu_j \hat{\tau}_j^* \leq H_{R_m}(p, n) \sum_{j=R_m+1}^{s-1} \mu_j \] (45)

Then, recalling that \( \hat{x}_{s-1}^* = \hat{x}_{R_m}^* + \sum_{j=R_m+1}^{s-1} \hat{\tau}_j^* \mu_j \) and using (44), (45) we get
\[ d_{s-1} - \hat{x}_{R_m}^* \leq H_{R_m}(p, n) \sum_{j=R_m+1}^{s-1} \mu_j \] (46)

We now make use of the inequality (21) along with the definition of \( H_{s-1}(p, n) \) from (7) to obtain:
\[ h_{s-1}(p, n) > H_{R_m}(p, n) \sum_{j=p}^{s-1} \mu_j \] (47)

From (47) and the definition of \( H_{R_m}(p, n) \) in (7), i.e., \( h_{R_m}(p, n) = H_{R_m}(p, n) \sum_{j=p}^{R_m} \mu_j \), we get
\[ h_{s-1}(p, n) - h_{R_m}(p, n) > (s - R_m - 1) \cdot H_{R_m}(p, n) \sum_{j=R_m+1}^{s-1} \mu_j \] (48)
From (6), and the facts that \( R_m < m \leq n \) and \( s - 1 < m \leq n \), we have

\[
h_{R_m}(p, n) = d_{R_m} - B(p), \quad h_{s-1}(p, n) = d_{s-1} - B(p)
\]

which implies

\[
h_{s-1}(p, n) - h_{R_m}(p, n) = d_{s-1} - d_{R_m} \quad (49)
\]

From (49) and (48),

\[
d_{s-1} - d_{R_m} > H_{R_m}(p, n) \sum_{j=R_m+1}^{s-1} \mu_j \quad (50)
\]

Since \( \hat{x}_{R_m} \leq d_{R_m} \), it follows from (50) that

\[
d_{s-1} - \hat{x}_{R_m} > H_{R_m}(p, n) \sum_{j=R_m+1}^{s-1} \mu_j
\]

which contradicts inequality (46). This, inequality (38) must hold.

**Part 4**: We finally prove that task \( R_m \) is a right-critical task and also the first critical task in \( C(p, n) \), and show that \( \hat{\tau}^*_i = H_{R_m}(p, n), \quad i = p, \ldots, R_m \).

From (29) and (38), task \( R_m \) is indeed right-critical, that is,

\[
\hat{\tau}^*_{R_m+1} > \hat{\tau}^*_R \quad (51)
\]

and from (51) and Proposition 3, it follows that

\[
\hat{x}_{R_m} = d_{R_m} \quad (52)
\]

By Proposition 5, the optimal controls \( \hat{u}^*_i, i = p, \ldots, R_m \) can be derived by solving the problem \( C(p, R_m) \):

\[
\min_{u_p, \ldots, u_{R_m}} \left\{ J(p, R_m) = \sum_{i=p}^{R_m} \mu_i \theta(\tau_i) \right\}
\]

s.t. \( g_i(p, R_m) \leq \sum_{j=p}^{i} \mu_j \tau_j \leq h_i(p, R_m), \quad i = p, \ldots, R_m - 1. \)

\[
\sum_{j=p}^{R_m} \mu_j \tau_j = h_{R_m}(p, R_m)
\]

From (6) we can see that

\[
g_i(p, R_m) = g_i(p, n), \quad h_i(p, R_m) = h_i(p, n), \quad i = p, \ldots, R_m - 1 \quad (53)
\]
From (52) and (6),
\[ \sum_{j=p}^{R_m} \mu_j \tau_j = h_{R_m}(p, R_m) = h_{R_m}(p, n) = d_{R_m} - B(p) \]  \hspace{1cm} (54)

From (20) and (53), recalling the definitions in (7), we have
\[ g_i(p, R_m) \leq H_{R_m}(p, n) \sum_{j=p}^{i} \mu_j \leq h_i(p, R_m), \quad i = p, \ldots, R_m - 1 \]  \hspace{1cm} (55)

From (55), \( \{\tau_i = H_{R_m}(p, n), \ i = p, \ldots, R_m\} \) must be feasible for \( C(p, R_m) \). Thus, from Lemma 5, the optimal controls of \( C(p, R_m) \), i.e., \( \{\hat{\tau}_i^*, \ i = p, \ldots, R_m\} \) in \( \hat{Q}(k, n) \) are
\[ \hat{\tau}_i^* = H_{R_m}(p, n), \quad i = p, \ldots, R_m \]  \hspace{1cm} (56)

From (56), tasks \( i = p, \ldots, R_m - 1 \) are obviously not critical tasks. Therefore, task \( R_m \) is the first critical task in \( C(p, n) \).