GENERALIZED ELASTIC SCHEDULING FOR REAL-TIME SYSTEMS

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science
in Computer Science and Engineering

by

Thidapat Chantem, B.S.

Xiaobo Sharon Hu, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
April 2008
GENERALIZED ELASTIC SCHEDULING FOR REAL-TIME SYSTEMS

Abstract

by

Thidapat Chantem

The elastic task model is a powerful model for adapting periodic real-time systems in the presence of uncertainty. This thesis generalizes the existing elastic scheduling approach in several directions. First, it presents a general framework, which formulates a trade-off between task schedulability and a specific performance metric as an optimization problem. Such a framework allows real-time systems under overloads to graciously adapt by adjusting their performance level.

Second, it is shown in this thesis that the well-known task compression algorithm in fact solves a quadratic programming problem that seeks to minimize the sum of the squared deviation of a task’s utilization from initial desired utilization. This finding indicates that the task compression algorithm may be applied to efficiently solve other similar types of problems that often arise in real-time applications. In particular, an iterative approach is proposed to solve the period selection problem for real-time tasks with deadlines less than respective periods. Further, the framework is adapted to solve the deadline selection problem, which is useful in some real-time control systems with fixed periods.
CONTENTS

FIGURES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iv

TABLES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v

ACKNOWLEDGMENTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . vi

CHAPTER 1: INTRODUCTION AND RELATED WORK . . . . . . . . . . 1

CHAPTER 2: PRELIMINARIES . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.1 Periodic Task Model . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.2 Elastic Task Model . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

CHAPTER 3: MOTIVATIONS . . . . . . . . . . . . . . . . . . . . . . . . . . 9

CHAPTER 4: PERIOD SELECTION FOR THE BASIC TASK MODEL . . . 13

CHAPTER 5: PERIOD SELECTION WITH ADDITIONAL DEADLINE
  CONSTRAINTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
  5.1 Simplified Sufficient Condition for Schedulability . . . . . . . . . 19
  5.2 Period Selection with Deadlines Less than Periods . . . . . . . . . 24
  5.3 Proposed Heuristic . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

CHAPTER 6: DEADLINE SELECTION FOR REAL-TIME TASKS . . . . . 38

CHAPTER 7: EXPERIMENTAL RESULTS . . . . . . . . . . . . . . . . . . . 44
  7.1 Period Selection with Deadlines Equal to Periods . . . . . . . . . 44
  7.2 Period Selection with Deadlines Less Than Periods . . . . . . . . 46
  7.3 Experimental Data for Real-World Workloads . . . . . . . . . . . . 52
    7.3.1 Videophone Application . . . . . . . . . . . . . . . . . . . . . 52
FIGURES

7.1 Utilization perturbation example ........................................ 46
7.2 Performance of simplified sufficient condition ........................ 49
7.3 Comparison of the heuristic and Loqo in terms of percentage of solutions found ................................................................. 50
7.4 Comparison of the heuristic and Loqo in terms of optimality of solutions 51
TABLES

3.1 TASK SET PARAMETERS FOR THE MOTIVATING EXAMPLE . 10

7.1 TASK SET PARAMETERS FROM [11] . . . . . . . . . . . . . . . . . 45
ACKNOWLEDGMENTS

This Master’s thesis would not have been possible without the patient help and guidance of my advisor, Dr. Sharon Hu. Under her care and mentoring, I have learned a great deal of knowledge, knowledge which is not only reflected in this thesis, but also knowledge about how to become a better researcher. Thank you for having taken me under your wings and given me this opportunity.

Dr. Michael Lemmon played a crucial role in what has become this thesis. I have always appreciated his sense of humor, which, at the same time, is always full of philosophical guidance and good intentions. Thank you for always reminding me that one should not forget the true motivations behind research and for the many discussions that we have had, many of which were for me and not for work.

I would also like to thank Dr. Christian Poellabauer, who formally introduced me to real-time systems and who, through his teachings, has convinced me of my interest in the area. Dr. Chris is always full of energy and it has been a pleasure to listen to him share his many perspectives on research and I hope that I will be given the opportunity to continue to do so.

The work in this thesis is supported in part by NSF under grant number CNS-0410771.

I owe my most sincere thank to Dr. Manimaran Govindarasu, who has given me my first exposure to research and who has continued to be my mentor ever since. Dr. Mani Mina never lets me forget about myself and understands how easy it is for one to lose perspectives in such a pursue for knowledge. I would also like to thank all my professors, instructors, and teachers, who have always greatly influenced me.
I am indebted to Dr. Bren Mochocki who has so generously created the task set generator for me to use in my experiments. A big, heartfelt thank also goes to my peers, both at Notre Dame and at Iowa State, who have made my days in higher education such a memorable experience. You know who you are. Believe me, I will always remember the times we had together, especially the laughter we shared during our most stressful times.

I have been so fortunate to have such a strong support system, which I find in my family and friends. Without them, I may have settled for something less, simply because I had doubts about my own ability, or perhaps because I was afraid to be so challenged. This thesis is for you, Dads and Moms. My Dad Sermsak and my Mom Patcharee have always believed in education, and never tired to iterate that it is the only lasting gift they could give me. Thank you for always standing by me, no matter which paths I have chosen and which paths lay ahead of me. My Dad Harry and my Mom Jeanette have always been there for me as well. Thank you for believing in me and for your encouragements. I hope I have made you all proud.

To my sister Kate, who takes such good care of me I feel like I am the little sister sometimes, thank you. Also, thank you to my brother Joshua and to the rest of my family and friends, which are so large and great that it would be impossible to name and recognize them all. Doing so would be a thesis by itself. Thank you all for being in my life.

Last but most definitely not least, I would like to thank my fiancé Ryan, for having given me his unconditional support throughout this journey and whose mere presence brightens my life. Ryan has been the one who encouraged me on when I wanted to leave it all behind, and who made me slow down when I began to move forward too fast. I can only hope that I am doing the same for him.
CHAPTER 1

INTRODUCTION AND RELATED WORK

As our daily life becomes more and more entwined with electronic devices, a particular type of systems has become predominant and commanded significant research attention. These systems are known as real-time systems, which require not only correctness, but also timing constraints. A familiar example of a real-time system is the breaking system in a car.

A desirable property of any real-time system is the guarantee that it will perform at least beyond some pre-specified thresholds defined by system designers to ensure performance (e.g., to ensure safety in the breaking system, the car must break within $x$ seconds). This is usually not a concern under normal situations where analysis has been done offline to ensure system performance based on the regular workload. However, in response to an event such as user’s input or changing environment, the load of the system may dynamically change in such a way that a temporal overload condition occurs. The challenge, then, is to provide some mechanism to guarantee the minimum performance level under such circumstances.

Many periodic real-time task models have been proposed to extend timing requirements beyond the hard and soft deadlines based on the observation that jobs can be dropped without severely affecting performance ([6], [21]). For example, Ramanathan et al. proposed both online [18] and offline [33] scheduling algorithms that are based on the (m,k) model, which is analyzed in [19]. In this model, up
to $k - m$ consecutive jobs are allowed to be dropped in any sliding window of $k$. Moreover, [41] presented the Dynamic Window-Constrained Scheduling (DWCS) algorithm, which is similar except that the window $k$ is fixed. Due to their success, these scheduling algorithms have further been enhanced. For example, the authors in [32] proposed a pattern rotation scheme for the (m,k) model to improve schedulability by avoiding some critical instants. Mok et. al. modified DWCS, which is primarily deadline-based, by incorporating the concept of pfairness [3] to improve the success rate for tasks with unit-size execution time [30]. Other frameworks such as the imprecise computation model [15] and reward-based model [1] can be used to capture situations where the quality of service is proportional to the amount of workload completed. In [25], the authors defined a novel QoS constraint for firm real-time systems using Markov chains to model the probabilistic behavior of real-time tasks.

Despite the success of the aforementioned models in alleviating overload situations, it is sometimes more suitable to execute jobs less often instead of dropping them or allocating fewer cycles. For example, limitations on the throughput capacity of ad hoc communication networks [2] make it highly desirable to reduce overall network traffic by having control tasks adaptively adjust their periods in response to the actual activity level of the control application.

The work in [23] was among the first to address task period adjustments. Seto, et al. considered the problem of finding a feasible set of task periods as a non-linear programming problem which seeks to optimize a specific form of control performance measure [35]. In [36], finding all feasible periods of a given set of tasks was studied for the Rate Monotone (RM) scheduling algorithm. Cervin et al. used optimization theory to solve the period selection problem online by adaptively adjusting task periods while optimizing a specific form of control performance [13]. Recently, [8]
offered a search algorithm that optimally solves the period selection problem for
fixed-priority scheduling schemes. The algorithm may be applicable only during
the design phase due to its potentially high complexity. Another interesting frame-
work was introduced in [22] where task periods are adjusted in response to varying
computation times.

Buttazzo and his colleagues proposed an elegant and flexible framework known
as the elastic task model [10] where deadline misses are avoided by increasing tasks
periods. The work in [12] extended the basic elastic task model to handle cases where
the computation time is unknown, [11] incorporated a mechanism to handle resource
constraints within the elastic framework, and [9] provided a means to smoothly
adjust task execution rate. In addition, [16] used a control performance metric as
cost function to find an optimal sampling interval for each task.

We will focus our attention on the elastic task model where the selection of
task periods is central. The existing elastic scheduling algorithm determines task
periods based on an elegant analogy between spring systems and task scheduling in
which a task’s resistance to changing its period is viewed as a spring’s resistance to
being compressed. In accordance with the principle of least action found in classical
mechanics, this suggests that the elastic task model is really attempting to minimize
some overall measure of the task set’s energy, whose precise nature was not made
clear in the original work.

Based on our previous findings in [14], this thesis generalizes the existing elas-
tic scheduling approach in several directions. First, we re-examine the problem of
period determination in the elastic task model and show that the task compres-
QP problem seeks to minimize the sum of the squared deviation of every task’s
utilization from its initial utilization. Identifying the nature of the optimization
problem underlying the task compression algorithm is important in several aspects. For instance, it may suggest other relevant optimization objectives and shed light on determining task periods in the presence of overloads for other task models.

Second, the proposed framework is extended to cases where task deadlines are less than task periods. Such task specifications often arise in control systems to reduce jitters, for example. Moreover, in some situations, it is desirable that tasks finish executing sooner, even if their periods are not up. We formulate the problem of period selection as a constrained optimization problem and propose a heuristic approach based on the task compression algorithm in [11] to solve the problem. The heuristic is guaranteed to find a feasible solution, if one exists. It is quite efficient and is hence suitable for online period adjustments. Experimental results show that the heuristic actually finds the global optimal solution in many cases.

Finally, the proposed framework is generalized to solve the deadline selection problem whose objective is to find a set of task deadlines such that the task set is feasible. Solving this problem is useful for systems where tasks have specific periods but some delays in task completions are tolerable.

We begin by reviewing key background materials in Chapter 2 and provide some motivations to this work in Chapter 3. Chapter 4 presents the solutions to the period selection problem for tasks with deadlines equal to periods. The optimization approach is then extended in Chapter 5 to treat the case where task deadlines are less than task periods. In Chapter 6, we demonstrate how the proposed framework can be used to solve the deadline selection problem. Experimental results are presented and discussed in Chapter 7 and we conclude with Chapter 8.
CHAPTER 2

PRELIMINARIES

This chapter describes the system under consideration and states important assumptions pertaining to our task model. We also briefly review the task compression algorithm used for period selection [11].

2.1 Periodic Task Model

We consider a system where each task $\tau_i$ is periodic and is characterized by the following 6-tuple: $(C_i, D_i, T_i, T_{i_{\text{min}}}, T_{i_{\text{max}}}, e_i), i = 1, \ldots, N$, where $N$ is the number of tasks in the system, $C_i$ is the worst case execution time of $\tau_i$, $D_i$ is its deadline, and $T_i$ is $\tau_i$'s actual period. Furthermore, $T_{i_{\text{min}}}$ denotes the most desirable period of $\tau_i$, as specified by the application, whereas $T_{i_{\text{max}}}$ represents the maximum period beyond which the system performance is no longer acceptable. The elastic coefficient, $e_i$, represents the resistance of task $\tau_i$ to increasing its period in face of changes. The smaller the elastic coefficient of a task, the harder it is to increase that task’s period. Given a task set $\Gamma$, tasks are arranged in a non-decreasing order of deadlines and all tasks start at time 0.

Task deadlines are first assumed to be equal to task periods. This requirement will be relaxed in Chapter 5; in that chapter, we treat the case where task deadlines are less than task periods. All task attributes are real values and are assumed to
be known \textit{a priori}. The current utilization of $\tau_i$ is $U_i = \frac{C_i}{T_i}$. Similarly, the minimum and the desired utilizations of $\tau_i$ is $U_{i\text{min}} = \frac{C_i}{T_{i\text{max}}}$ and $U_{i\text{max}} = \frac{C_i}{T_{i\text{min}}}$, respectively.

2.2 Elastic Task Model

In [11], Buttazzo, et al. modeled a task system as a spring system, where increasing or decreasing a task period is analogous to compressing or decompressing a spring. The elastic coefficient, $e_i$, introduced above hence has its intuitive meaning of the hardness of the spring. The purpose of increasing task periods is to drive the total utilization of the system down to some desired utilization level, $U_d$, analogous to a spring system trying to minimize its energy under an external force.

The attractiveness of the elastic task model is its accompanying task compression algorithm, which is quite efficient ($O(N^2)$) and can readily be used online. (In fact, the elastic task model and the task compression algorithm have already been implemented in the S.Ha.R.K. kernel [17].) The task compression algorithm works as follows. If it is possible to drive the system utilization down to $U_d$ without violating any period bounds, the algorithm will return a set of feasible periods $(T_1, T_2, \ldots, T_N)$ that can be used by the system. Tasks whose periods are fixed (i.e., $e_i = 0$ or $T_{i\text{min}} = T_{i\text{max}}$) are considered inelastic and are treated as special cases. The amount of utilization that each remaining, non-inelastic task should receive is computed based on its elastic coefficient, initial period, and the amount of utilization that must be reduced to achieve $U_d$. The resultant period of a task $\tau_i$ is guaranteed to fall somewhere between $T_{i\text{min}}$ and $T_{i\text{max}}$. For completeness, the task compression algorithm is reproduced in Algorithm 1.

Throughout this work, we will assume that the Earliest Deadline First (EDF) scheduling algorithm [24] is used. Furthermore, we will focus our attention on cases where tasks need to decrease their utilization in response to either internal
Algorithm 1 Task_Compress($\Gamma$, $U_d$)

1: $U_0 = \sum_{i=1}^{n} C_i / T_{i\text{min}}$
2: $U_{\text{min}} = \sum_{i=1}^{n} C_i / T_{i\text{max}}$
3: if ($U_d < U_{\text{min}}$) then
4: return INFEASIBLE
5: end if
6: repeat
7: $U_f = E_v = 0$
8: for (each $\tau_i$) do
9: if (($e_i == 0$) or ($T_i == T_{i\text{max}}$)) then
10: $U_f = U_f + U_i$
11: else
12: $E_v = E_v + e_i$
13: $U_{i\emptyset} = U_0 - U_f$
14: end if
15: end for
16: $ok = 1$
17: for (each $\tau_i \in \Gamma_v$) do
18: if (($e_i > 0$) and ($T_i < T_{i\text{max}}$)) then
19: $U_i = U_{i\text{max}} - (U_{i\emptyset} - U_d + U_f) e_i / E_v$
20: if ($U_i < U_{i\text{min}}$) then
21: $U_i = T_{i\text{min}}$
22: $ok = 0$
23: end if
24: end if
25: end for
26: until ($ok == 1$)
27: return FEASIBLE
(e.g., change in sampling rate of one or more tasks in the system) or external (e.g., network traffic) factors.
CHAPTER 3

MOTIVATIONS

This work focuses on the problem of how to effectively and efficiently adjust task periods or deadlines such that a temporarily overloaded system becomes feasible. There are many situations where a real-time system may experience temporal overload conditions. Consider, for example, a city sewer control system, which is responsible for drainage and waste management. Said system consists of several sensors and actuators, which are spread throughout the city and are controlled by a central computer. The sensors report, among other things, the level of water in a particular pipe, while the actuators may be used to control the flow rate of water in that pipe. Suppose now that part of the city has been experiencing a heavy rain. Since the water level in some of the pipes may be increasing at an alarming rate, we may need to sample the state of the pipes more frequently and send control commands to the relevant actuators more often to avoid flooding. Because of these new requirements, the city sewer control system is temporally under an overload situation, as more attention needs to be paid to the part of the city receiving heavy rainfall. In such a situation, it is better to increase the period of some other control tasks that control regions not under duress instead of hoping that the entire sewer control system will be able to manage (i.e., to properly control drainage) without any overload management scheme. For more information regarding work in control systems on combined sewer overflow (CSO) events, the reader is referred to [34],
and [31]. The following motivating example, which will also be referred to later, describes this scenario quantitatively.

Consider a task set consisting of 5 tasks, each of which represents a control task for a certain part of the city sewer control system. The parameters for each task are shown in Table 3.1. All tasks start at time 0 and task deadlines are less than task periods. Note that although all the five tasks have the same purpose (i.e., to control drainage), they have different timing requirements. Some of the reasons for such differences would be different control laws or hardware.

\begin{table}[h]
\centering
\caption{Task Set Parameters for the Motivating Example}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
Task & $C_i$ & $D_i$ & $T_{min}$ & $T_i$ & $T_{max}$ & $e_i$ \\
\hline
$\tau_1$ & 37 & 60 & 60 & 90 & 120 & 1 \\
$\tau_2$ & 3 & 43 & 50 & 100 & 230 & 1 \\
$\tau_3$ & 6 & 47 & 55 & 80 & 250 & 1 \\
$\tau_4$ & 10 & 93 & 100 & 110 & 280 & 1 \\
$\tau_5$ & 25 & 141 & 145 & 150 & 300 & 1 \\
\hline
\end{tabular}
\end{table}

At the startup of the system, there is no danger of flooding and each task $\tau_i$ has an initial period $T_i$ between $T_{min}$ and $T_{max}, i = 1, \ldots, 5$. It can readily be verified that the task set from Table 3.1 is schedulable under EDF based on the initial periods, $T_i, i = 1, \ldots, N$.

Assume that, when the heavy rain starts, $\tau_1$ needs to reduce its period to 60 time units in order to ensure no flooding occurs around the part of the city whose sewer system is controlled by $\tau_1$. Task $\tau_1$ is allowed to do so since its minimum period, $T_{min} = 60$ time units. However, the task set is no longer feasible. In other words, to allow for $\tau_1$ to change its period, the period of tasks $\tau_2, \tau_3, \tau_4$, and $\tau_5$ must increase.
for the system to remain schedulable. Suppose that there exists a mechanism to find a feasible task set by adjusting task periods, then said mechanism might assign $T_2 = 230$ time units, $T_3 = 179$ time units, $T_4 = 133$ time units, and $T_5 = 150$ time units. The task set is now schedulable and the tasks will continue to execute with their assigned periods until the heavy rain ends and $\tau_1$ goes back to its original period.

Note that when the system is overloaded, there may exist many different combinations of feasible task periods. For instance, in the above example, we could set $T_i = T_{i_{max}}$, $i = 2, \ldots, 5$, which is the maximum allowable periods for these tasks. This period assignment indeed makes the task set schedulable but it underutilizes the computing resource compared to the earlier assignment. As will be shown in the next chapter, we attempt to find the best period assignment for each task by trading off performance and schedulability using optimization theory. Our optimization framework has an objective that can be set to reflect the desired system performance. Such an objective does not affect the schedulability of the system; it merely specifies how the period of each task should be selected.

While task period adjustment may be appropriate for some control systems, for others, task deadline adjustment may be more suitable. For example, consider a control system whose initial sampling interval is large (e.g., 1000 time units). Under normal circumstances, the system may require the deadline of its task to be 100 time units. However, under perturbations, the deadline of 50 time units may be desirable as it allows the system to have smaller jitters. (Reducing jitters permits maximum predictable performance guarantee.) Thus, in this type of control systems, it is more appropriate to adjust task deadlines instead of task periods. Depending on the system, tasks can tolerate some specific range of deadlines and some may be able to increase its deadline temporally without significantly sacrificing performance. As
before, we formulate our problem using optimization theory to maximize some type of performance metric. For this reason, our work is much different from [7] where the focus is placed on characterizing feasible deadline regions assuming EDF scheduling policy.
CHAPTER 4

PERIOD SELECTION FOR THE BASIC TASK MODEL

In this chapter, we focus on the basic periodic task model where, for each task, \( D_i = T_i \), \( i = 1, \ldots, N \). Given an initially infeasible set of real-time tasks, there may exist numerous sets of feasible periods. It is not difficult to see that different sets of periods would lead to different performance of the resultant system. In general, the period selection problem can be expressed as an optimization problem. That is,

\[
\text{optimize: } \text{performance metric} \\
\text{s.t.: } \text{tasks are schedulable} \\
\quad \text{period bounds are satisfied}
\]

Below we introduce a specific performance metric and discuss its implications. We assume that task deadlines equal task periods.

Processor utilization by each task is an important measure for any real-time system. It not only reveals the amount of system resource dedicated to the task but also impacts schedulability. In the elastic task model, one consequence of changing task periods is changing the utilization of tasks. From the standpoint of performance preservation, it is desirable to minimize the changes in task utilization. (For example, decreasing a task utilization may mean that a control system executes less often and is more susceptible to perturbations. Therefore, we would want to decrease the task utilization as little as possible.) This objective can be captured by the following constrained optimization problem.
\[ \text{min: } E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i (U_{i_{\text{max}}} - U_i)^2 \quad (4.1) \]

\[
\begin{align*}
\text{s.t.: } & \sum_{i=1}^{N} U_i \leq U_d \\
& U_i \geq U_{i_{\text{min}}} \quad \text{for } i = 1, 2, \cdots, N \\
& U_i \leq U_{i_{\text{max}}} \quad \text{for } i = 1, 2, \cdots, N
\end{align*}
\quad (4.2) \quad (4.3) \quad (4.4)
\]

In the formulation, \( N \) is the number of tasks in the system, \( U_{i_{\text{max}}} \) is the desired utilization of task \( \tau_i \) and \( U_{i_{\text{max}}} \geq U_{i_{\text{min}}} \), \( U_i \) is the utilization of \( \tau_i \) to be determined, and \( U_d \) is the desired total utilization. (\( U_d \) is usually set to 1 for EDF scheduling.) Constant \( w_i (\geq 0) \) is a weighting factor and reflects the criticality of a task. More critical tasks would have larger \( w_i \)'s. The first constraint simply states the schedulability condition under EDF. The rest of the constraints bound the utilization, equivalently bound the task period by \( T_{i_{\text{min}}} \) and \( T_{i_{\text{max}}} \) where \( T_{i_{\text{min}}} = C_i / U_{i_{\text{max}}} \) and \( T_{i_{\text{max}}} = C_i / U_{i_{\text{min}}} \), \( i = 1, \ldots, N \).

It is worth noting that for \( w_i = 0 \), (4.1) does not change regardless of what \( U_i \) value is used. To help satisfy (4.2), it is natural to simply use \( U_i = U_{i_{\text{min}}} \). Hence, for the rest of this work, we will focus on the case where \( w_i > 0 \), \( i = 1, \ldots, N \).

The problem in (4.1)–(4.4) belongs to the category of quadratic programs and can be solved in polynomial time. However, solving such a problem using a quadratic program solver (such as Loqo [38]) during runtime can be too costly. What makes the above formulation attractive is that its solution is exactly the same as that found by the task compression algorithm in [11]. We introduce a lemma and a theorem to support this argument.

\textbf{Lemma 4.1} Given the constrained optimization problem as specified in (4.1)–(4.4) and \( \sum_{i=1}^{N} U_{i_{\text{max}}} > U_d \), any solution, \( U_i^* \), to the problem must satisfy \( \sum_{i=1}^{N} U_i^* = U_d \) and \( U_i^* \neq U_{i_{\text{max}}} \), for \( i = 1, \ldots, N \).
Proof

We prove the lemma by utilizing the Karush-Kuhn-Tucker (KKT) necessary conditions for the solution to the given problem, which can be written in terms of the Lagrangian function for the problem as

$$J_a(U, \mu) = \sum_{i=1}^{N} w_i (U_{i_{max}} - U_i)^2 + \mu_0 \left( \sum_{i=1}^{N} U_i - U_d \right) + \sum_{i=1}^{N} \mu_i (U_{i_{min}} - U_i) + \sum_{i=1}^{N} \lambda_i (U_i - U_{i_{max}})$$  \hspace{1cm} (4.5)

where $\mu_0, \mu_i,$ and $\lambda_i$ are Lagrange multipliers, $\mu_0 \geq 0, \mu_i \geq 0,$ and $\lambda_i \geq 0,$ for $i = 1, \ldots, N.$ The necessary conditions for the existence of a relative minimum at $U^*_i, i = 1, \ldots, N,$ are

$$0 = \frac{\partial J_a}{\partial U^*_i} = -2w_i (U_{i_{max}} - U^*_i) + \mu_0 - \mu_i + \lambda_i$$  \hspace{1cm} (4.6)

$$0 = \mu_0 \left( \sum_{i=1}^{N} U^*_i - U_d \right)$$  \hspace{1cm} (4.7)

$$0 = \mu_i (U_{i_{min}} - U^*_i)$$  \hspace{1cm} (4.8)

$$0 = \lambda_i (U^*_i - U_{i_{max}}).$$  \hspace{1cm} (4.9)

Assume that (4.2) is inactive, i.e., $\mu_0 = 0$ and $\sum_{i=1}^{N} U^*_i < U_d.$ Then at least one constraint in (4.3) or (4.4) must be active. Suppose the $k$-th constraint in (4.3) is active. That is, $U^*_k = U_{k_{min}}$ and $\mu_k \geq 0.$ Then, the $k$-th constraint in (4.4) must be inactive, i.e., $\lambda_k = 0.$ From (4.6), we obtain

$$\mu_k = -2w_k (U_{k_{max}} - U_{k_{min}}) < 0$$  \hspace{1cm} (4.10)

which contradicts the assumption that $\mu_k \geq 0.$ Therefore, if any $U^*_k = U_{k_{min}},$ constraint (4.2) must be active.
Now assume that some constraint in (4.4) is active while others are inactive. Suppose $U^*_h = U_{h_{max}}$ (active) and $U_{k_{min}} < U_k < U_{k_{max}}$ (inactive). Then $\mu_h = 0$, $\lambda_h \geq 0$, and $\mu_k = \lambda_k = 0$. From (4.6), we have

\begin{align*}
\mu_0 &= 2w_h(U_{h_{max}} - U^*_h) + \mu_h - \lambda_h = -\lambda_h \quad (4.11) \\
\mu_0 &= 2w_k(U_{k_{max}} - U^*_k). \quad (4.12)
\end{align*}

Note that (4.11) and (4.12) cannot be simultaneously satisfied. Therefore, we can either have all the constraints in (4.4) be active or all are inactive. If all the constraints in (4.4) are active, we have

\begin{equation}
\sum_{i=1}^{N} U^*_i = \sum_{i=1}^{N} U_{i_{max}} > U_d \quad (4.13)
\end{equation}

which contradicts the assumption that the resultant task set is schedulable. If all the constraints in (4.4) are inactive, (4.12) requires that $\mu_0 > 0$, which contradicts the assumption that constraint (4.2) is inactive. Therefore, for any solution to the optimization problem, constraint (4.2) must be active, i.e., $\sum_{i=1}^{N} U^*_i = U_d$ and $U^*_i \neq U_{i_{max}}$.

\begin{equation}
\n\end{equation}

**Theorem 4.1** Given the constrained optimization problem as specified in (4.1)–(4.4), $\sum_{i=1}^{N} U_{i_{max}} > U_d$, and $\sum_{i=1}^{N} U_{i_{min}} \leq U_d$, let $\hat{U} = \sum_{i \neq i_{min}} U^*_i + \sum_{i = i_{min}} U_{i{min}}$. A solution to the problem, $U^*_i$, is optimal if and only if

\begin{equation}
U^*_i = U_{i_{max}} - \frac{1}{w_i} \left( \hat{U} - U_d \right) \quad (4.14)
\end{equation}

for $\hat{U} > U_d$ and $U^*_i > U_{i_{min}}$, and $U^*_i = U_{i_{min}}$ otherwise.

**Proof**
Consider the KKT conditions given in (4.6)–(4.9). From Lemma 4.1, we know that any solution to the given optimization problem must satisfy (4.2), i.e., \( U_d = \sum_{i=1}^{N} U_i^* \), and \( U_i^* \neq U_{i_{\text{max}}} \). Hence, we only need to consider the case where \( \lambda_i = 0 \), for all \( i = 1, \ldots, N \). Suppose that the \( k \)-th constraint in (4.3) is active. We have \( U_k^* = U_{k_{\text{min}}} \), and

\[
\mu_k = \mu_0 - 2w_k (U_{k_{\text{max}}} - U_{k_{\text{min}}}).
\]  

(4.15)

Otherwise, we have \( \mu_k = 0 \). By summing up (4.6) for all \( i \) and using the conclusions above,

\[
\mu_0 = \frac{2 (\hat{U} - U_d)}{\sum_{U_i^* \neq U_{i_{\text{min}}}} (1/w_i)},
\]  

(4.16)

as long as \( \hat{U} > U_d \), \( \mu_0 > 0 \), \( \mu_i \geq 0 \), and the constraints in (4.4) are satisfied. Therefore, a solution, \( U_i^* \), to the optimization problem either satisfies \( U_i^* = U_{i_{\text{min}}} \) or can be obtained by combining (4.16) with (4.6) for \( U_i^* > U_{i_{\text{min}}} \). (Note that \( \mu_i = 0 \) when \( U_i^* > U_{i_{\text{min}}} \).) That is,

\[
U_i^* = U_{i_{\text{max}}} - \frac{1}{w_i} \left( \hat{U} - U_d \right) \frac{1}{\sum_{U_j^* \neq U_{j_{\text{min}}}} (1/w_j)}.
\]  

(4.17)

Additionally, since the objective function and the inequality constraints in (4.1)–(4.4) are convex, the necessary conditions for optimality provided by the KKT conditions also become the sufficient conditions for optimality \[28\]. Hence, the solution found in Theorem 4.1 is a global minimum.

\[\square\]

Based on the previous lemma and theorem, we can draw the following conclusion.
Corollary 1 Consider a set of $N$ tasks where $U_i$ is the utilization of the $i$th task, $i = 1, \ldots, N$. Let $U_{i_{\text{max}}}$ denote the initial desired utilization of task $\tau_i$ and let $e_i = \frac{1}{w_i} > 0$ be a set of elastic coefficients, $i = 1, \ldots, N$. Let $U_d$ be the desired utilization level and $\sum_{i=1}^{N} U_{i_{\text{max}}} > U_d$. The task utilizations $U_i$, $i = 1, \ldots, N$, obtained from the task compression algorithm in [11] minimize

$$E(U_1, \ldots, U_N) = \sum_{i=1}^{N} \frac{1}{e_i} (U_{i_{\text{max}}} - U_i)^2$$

subject to the inequality constraints $\sum_{i=1}^{N} U_i \leq U_d$, $U_i \geq U_{i_{\text{min}}}$, and $U_i \leq U_{i_{\text{max}}}$, for $i = 1, \ldots, N$.

The above corollary has several significant consequences. First, it reveals the optimization criterion inherent in the task compression algorithm. Second, it provides guidance on the selection of other performance measures. Third, the task compression algorithm may be modified and/or extended to solve similar convex programming problems. An example of this extension will be described in the next chapter.
CHAPTER 5

PERIOD SELECTION WITH ADDITIONAL DEADLINE CONSTRAINTS

In this chapter, we consider the case where task deadlines are less than task periods. This more general task model is useful in situations where it is desirable for a task to finish executing early (before its period ends). By using the optimization framework introduced in Chapter 4, we again formulate the period selection problem as a constrained optimization problem and propose a novel heuristic based on the task compression algorithm. The algorithm is guaranteed to find a solution to the problem, if one exists, and is efficient enough for online use.

5.1 Simplified Sufficient Condition for Schedulability

Baruah et al. considered the case where task deadlines are less than or equal to task periods and derived a sufficient and necessary condition for EDF schedulability [4], which is later improved in [5]. The condition is restated in the following theorem.

**Theorem 5.1** [4] Given a periodic task set with $D_i \leq T_i$, the task set is schedulable if and only if the following constraint is satisfied $\forall L \in \left\{ kT_i + D_i \leq \min(B_p, H) \right\}$ and $k \in \mathbb{N}$ (the set of natural numbers including 0), where $B_p$ and $H$ denote the busy period and hyperperiod, respectively,

$$L \geq \sum_{i=1}^{N} \left( \left\lceil \frac{L - D_i}{T_i} \right\rceil + 1 \right) C_i$$

(5.1)
Based on Theorem 5.1, the period determination problem can be formulated as follows.

\[
\begin{align*}
\text{min:} & \quad E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i (U_{i_{\text{max}}} - U_i)^2 \\
\text{s.t.:} & \quad L \geq \sum_{i=1}^{N} \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \\
& \quad L \in \{kT_i + D_i \leq \min(B_p, H)\}, k \in \mathbb{N} \\
& \quad U_i \geq U_{i_{\text{min}}} \quad \text{for } i = 1, 2, \cdots, N \\
& \quad U_i \leq U_{i_{\text{max}}} \quad \text{for } i = 1, 2, \cdots, N
\end{align*}
\]

(5.2) (5.3) (5.4) (5.5)

Solving the above constrained optimization problem can be extremely time consuming. Hence, we investigate solving the problem approximately with an efficient algorithm. An approximate solution is both acceptable and preferred, as a rapid response allows the system to degrade gracefully instead of entering into potentially catastrophic states due to some dynamic perturbations.

Since verifying the constraint in (5.1) for all \( L \) values is the main source of complexity, we consider simplifying the schedulability test by using the following stronger schedulability condition,

\[
L \geq \sum_{i=1}^{N} \left( \frac{L - D_i}{T_i} + 1 \right) C_i
\]

(5.6)

It is not difficult to see that if the inequality in (5.6) is satisfied then the original inequality in (5.1) must also be satisfied. What makes (5.6) an excellent candidate for online use is that the schedulability of a task set can be determined based on a single \( L \) value, \( L^* \). Below, we introduce several lemmas and a theorem to support this claim.
For simplicity, we denote the set of all possible values of $L$ by a distinct ordered set $\mathcal{L} = \{L_0, L_1, \ldots\}$ where $L_j = kT_i + D_i, k \in \mathcal{N}$ and $L_j \leq \min(B_p, H)$.

**Lemma 5.1** Given a set $\Gamma$ of $N$ tasks with $D_i \leq T_i$, let $L_j$ and $L_{j+1} \in \mathcal{L}$ and let $L_j < L_{j+1}$. If the constraint in (5.6) is satisfied for $L_j$, then it is satisfied for $L_{j+1}$.

**Proof**

By regrouping the terms in (5.6), we can rewrite the inequality as follows.

$$L \geq \frac{\sum_{i=1}^{N} C_i - \sum_{i=1}^{N} U_i D_i}{1 - \sum_{i=1}^{N} U_i} \quad (5.7)$$

Given that $L_j$ satisfies the constraint in (5.7) and $L_{j+1} > L_j$, it immediately follows that $L_{j+1}$ satisfies the constraint in (5.7).

Based on the above lemma, we can conclude that if the constraint in (5.6) is satisfied for $L_j$, then it is also satisfied for all $L_k \in \mathcal{L}$, where $L_k > L_j$. It may then seem natural to simply set $L^*$ to be the minimum of all $L$ values in $\mathcal{L}$. However, such a choice can be extremely pessimistic, often resulting in finding no feasible solutions to the problem. To avoid being too pessimistic, we introduce the next lemma, which identifies useful necessary conditions for any feasible task set. The lemma helps to eliminate pessimistic choices of $L^*$.

**Lemma 5.2** Let $D_i$ be the deadline of task $\tau_i$ in a given task set $\Gamma$, $i = 1, \ldots, N$. Assume that $\tau_i \in \Gamma$ starts at time 0. Further, let the tasks in $\Gamma$ be ordered in a non-decreasing order of deadlines and suppose that $D_{\min}$ is unique. Regardless of the choices of periods, any task set that is schedulable must satisfy the following property:

$$\sum_{i=1}^{j} C_i \leq D_j, j = 1, \ldots, N \quad (5.8)$$
Proof

We prove Lemma 5.2 by contradiction. Suppose the task set is schedulable and \( \sum_{i=1}^{j} C_i > D_j \) for some \( j \). By time \( D_j \), at least one instance of \( \tau_1, \ldots, \tau_j \) must each finish executing. Thus, The total processor demand at \( D_j \) is at least \( \sum_{i=1}^{j} C_i \) time units. However, since there are only \( D_j \) time units available and \( \sum_{i=1}^{j} C_i > D_j \), at least one instance of \( \tau_1, \ldots, \tau_j \) has to miss its deadline. This contradicts the assumption that the task set is schedulable. Hence, (5.8) must be true for all \( j = 1, \ldots, N \).

\[ \]

We are now ready to introduce two lemmas which form the basis for our selection of \( L^* \).

**Lemma 5.3** Consider a set \( \Gamma \) of \( N \) tasks that satisfy the condition in Lemma 5.2. Let the tasks in \( \Gamma \) be sorted in a non-decreasing order of deadlines. If \( D_1 + T_1 \leq D_2 \), and \( L^* = D_2 \) satisfies the inequality constraint in (5.6), then the task set is guaranteed to be schedulable.

**Proof**

Let \( L_h = L^* = D_2 \). By Lemma 5.1, any \( L_j \in \mathcal{L} \) with \( j > h \) satisfies constraint in (5.6) and hence satisfies (5.1). Now consider \( j < h \). Since \( D_i \geq D_2 \) for \( i > 2 \), \( L_j \) can only be equal to \( D_1 + kT_1 \) (since the task set must satisfy Lemma 5.2) for some \( k \in \mathcal{N} \). In order for \( L_j \) to satisfy (5.1), noting that \( |D_1 + kT_1 - D_i| < T_i \) for \( i \geq 2 \), we need \( D_1 + kT_1 \geq (k + 1) \cdot C_1 \), which holds true according to Lemma 5.2. Therefore, for all values of \( L \in \mathcal{L} \), (5.1) is satisfied.

\[ \]

**Lemma 5.4** Consider a set \( \Gamma \) of \( N \) tasks that satisfy the condition in Lemma 5.2. Let the tasks in \( \Gamma \) be sorted in a non-decreasing order of deadlines. If \( D_1 + T_1 > D_2 \), and \( L^* = \min_{i=1}^{N}(T_i + D_i) \) satisfies the inequality constraint in (5.6), then the task set is guaranteed to be schedulable.
Proof

Let $L_h = L^* = \min_{i=1}^{N} (T_i + D_i)$. By Lemma 5.1, any $L_j \in \mathcal{L}$ with $j > h$ satisfies constraint in (5.6) and hence satisfies (5.1). Now consider $j < h$. $L_j$ can only be equal to $D_k$, $k = 1, \ldots, N$, such that $D_k < L_h$. In order for $L_j$ to satisfy (5.1), we need $\sum_{i=1}^{j} C_i \leq D_j$, which holds true according to Lemma 5.2. Therefore, for all values of $L \in \mathcal{L}$, (5.1) is satisfied.

\[ \square \]

From Lemmas 5.2–5.4, we can conclude that given an arbitrary task set $\Gamma$ of $N$ tasks, a maximum number of $N + 1$ checks need to be performed to test the schedulability of that task set. Namely, at most $N$ checks must be performed to determine whether $\Gamma$ satisfies Lemma 5.2 and one check must be performed to determine whether either Lemmas 5.3 or 5.4 is satisfied. We collect these conclusions in the following theorem.

**Theorem 5.2** Consider a set $\Gamma$ of $N$ tasks that satisfy the condition in Lemma 5.2. Let the tasks in $\Gamma$ be sorted in a non-decreasing order of deadlines. A given task set is schedulable if

\[
L^* \geq \sum_{i=1}^{N} \left( \frac{L^* - D_i}{T_i} + 1 \right) C_i
\]

(5.9)

where

\[
L^* = \begin{cases} 
D_2 & : D_1 + T_1 \leq D_2 \\
\min_{i=1}^{N} (T_i + D_i) & : \text{otherwise}
\end{cases}
\]

Proof

From Lemmas 5.3 and 5.4, we know that the only $L$ that needs to be checked against the constraint in (5.6) is $D_2$ if $D_1 + kT_1 \leq D_2$, $k = 0, 1, \ldots$, and $\min_{i=1}^{N} (T_i + D_i)$ otherwise. Moreover, we have proved in Lemma 5.1 that if (5.6) is satisfied for some $L_j$ then it is also satisfied for $L_{j+1}$ where $L_j < L_{j+1}$. This, in turns, implies
that if (5.6) is satisfied for $L_j$ then it is also satisfied for all $L_k \in L$, where $L_k > L_j$.

Taken together, if the constraint in (5.6) is satisfied for $L^*$ then the task set is schedulable.

\[ \square \]

The above theorem paves the way to finding a simpler constrained optimization problem for the purpose of period determination for tasks with deadlines less than periods. We present the actual problem formulation in the following section.

### 5.2 Period Selection with Deadlines Less than Periods

By using Theorem 5.2, we can express the period selection problem where task deadlines are less than task periods as a constrained optimization problem similar to that in (4.1)–(4.4). Letting $r_i = L - D_i$, (5.6) can be rewritten as

$$
\sum_{i=1}^{N} r_i U_i \leq L - \sum_{i=1}^{N} C_i. \tag{5.10}
$$

Then the period determination problem where task deadlines are less than task periods can be formulated as

\begin{align*}
\text{min:} & \quad E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i (U_{i_{\text{max}}} - U_i)^2 \tag{5.11} \\
\text{s.t.:} & \quad \sum_{i=1}^{N} r_i U_i \leq L - \sum_{i=1}^{N} C_i \tag{5.12} \\
& \quad L = \begin{cases} 
D_2 & : D_1 + \frac{C_i}{U_i} \leq D_2 \\
\min\left(\frac{C_i}{U_i} + D_i\right) & : \text{otherwise}
\end{cases} \tag{5.13} \\
& \quad U_i \geq U_{i_{\text{min}}} \quad \text{for} \quad i = 1, 2, \cdots, N \tag{5.14} \\
& \quad U_i \leq U_{i_{\text{max}}} \quad \text{for} \quad i = 1, 2, \cdots, N \tag{5.15}
\end{align*}

24
Note that the above constrained optimization problem would have exactly the same format as the QP problem in (4.1)–(4.4) if $L$ and $r_i$ can be treated as constants. Unfortunately, this is not the case, as the actual value of $L$ is dependent on variable $U_i, i = 1, \ldots, N$. Consequently, the above optimization problem must be treated as a nonlinear program which can be too costly to solve in terms of both processor time and memory usage. (In fact, even a commercial nonlinear solver cannot guarantee that a solution will be found, even if one exists.)

The challenge, then, is to solve the problem efficiently as to allow the system to respond to dynamic changes in a timely manner. We propose using an iterative heuristic based on the task compression algorithm to tackle the challenge. Let us first treat $L$ as a constant. The solution for solving the optimization problem in (5.11)–(5.15) for a fixed value of $L$ will be applied iteratively to solve the original problem (where the value of $L$ may change).

Below, we introduce a lemma and a couple of theorems to show how the optimization problem in (5.11)–(5.15) can be optimally solved for a fixed value of $L$. We then give the details of our heuristic algorithm and discuss the solution quality.

**Lemma 5.5** Given the constrained optimization problem as specified in (5.11)–(5.15) and $\sum_{i=1}^{N} r_i u_{i_{\text{max}}} > L - \sum_{i=1}^{N} C_i$, any solution, $U_i^*$, to the problem must satisfy $\sum_{i=1}^{N} r_i U_i = L - \sum_{i=1}^{N} C_i$.

**Proof**

This lemma can be proved using the same technique as in Lemma 4.1.

We first consider solving the optimization problem in (5.11)–(5.15) when $D_1 + \frac{C_i^*}{U_i^*} \leq D_2$. According to Lemma 5.3, we only need to check that (5.12) is satisfied for $L^* = D_2$. It follows that we can solve the optimization problem efficiently as shown in the following theorem.
Theorem 5.3 Given the constrained optimization problem as specified in (5.11)–(5.15), for \( L = D_2 \), \( \sum_{i=1}^{N} r_i U_{i\text{max}} > L - \sum_{i=1}^{N} C_i \), and \( U_{1\text{min}} \leq U_i^* < U_{1\text{max}} \), a solution, \( U_i^* \), is optimal if and only if

\[
U_i^* = \begin{cases} 
D_2 - \sum_{j=1}^{N} C_j - \sum_{j=3}^{N} r_j U_{j\text{max}} & : i = 1 \\
U_{i\text{max}} & : \text{otherwise}
\end{cases}
\]

for \( D_2 > \sum_{j=1}^{N} C_j + \sum_{j=3}^{N} r_j U_{j\text{max}} \).

Proof

Let \( L_d = L - \sum_{i=1}^{N} C_i \). The KKT conditions for the solution to the optimization problem in (5.11)–(5.15) can be written as follows.

\[
\begin{align*}
0 &= -2w_i (U_{i\text{max}} - U_i^*) + r_i \mu_0 - \mu_i + \lambda_i \quad (5.16) \\
0 &= \mu_0 \left( \sum_{j=1}^{N} r_j U_j^* - L_d \right) \quad (5.17) \\
0 &= \mu_i (U_{i\text{min}} - U_i^*) \quad (5.18) \\
0 &= \lambda_i (U_i^* - U_{i\text{max}}) \quad (5.19)
\end{align*}
\]

for \( i = 1, \ldots, N \), where \( \mu_0, \mu_i \)'s and \( \lambda_i \)'s are Lagrange multipliers, \( \mu_0 \geq 0, \mu_i \geq 0 \), and \( \lambda_i \geq 0 \) for \( i = 1, \ldots, N \).

Consider first those tasks with \( D_k = D_2 \). Then \( r_k = L - D_k = 0 \) and (5.16) reduces to

\[
\mu_k - \lambda_k = -2w_k (U_{k\text{max}} - U_k^*). \quad (5.20)
\]

Assume that \( U_{k\text{min}} < U_k^* < U_{k\text{max}} \). In order to satisfy (5.18) and (5.19), we must have \( \mu_k = \lambda_k = 0 \), which contradicts (5.20). Now assume that \( U_k^* = U_{k\text{min}} \). Then to satisfy (5.19), we need \( \lambda_k = 0 \). However, this leads to \( \mu_k < 0 \) from (5.20), which violates the KKT conditions. Therefore, for those tasks with \( D_k = D_2 \), \( U_k^* = U_{k\text{max}} \).

(It can be readily proved that such a solution indeed satisfies the KKT conditions.)
Similarly, consider next those tasks with $D_h > D_2$. Then, $r_h = L - D_h < 0$ and (5.16) becomes

$$-2w_h (U_{h,\text{max}} - U_{h}^*) = \mu_h - \lambda_h - r_h \mu_0.$$  

(5.21)

In order to satisfy (5.18) and (5.19), we must have $\mu_h = \lambda_h = 0$, which will cause (5.21) to become

$$-2w_h (U_{h,\text{max}} - U_{h}^*) = D_h - D_2.$$  

(5.22)

This is clearly a contradiction, since $D_h > D_2$ and $U_{h,\text{max}} - U_{h}^* \geq 0$. Now, assume that $U_{h}^* = U_{h,\text{min}}$. Then, to satisfy (5.19), we need $\lambda_h = 0$. However, this leads to $\mu_h < 0$ in (5.21), which violates the KKT conditions. Therefore, for any task with $D_h > D_2$, $U_{h}^* = U_{h,\text{max}}$.

For $i = 1$, we first note that $\lambda_1$ must be equal to 0, since $U_{i}^* = U_{i,\text{max}}$, for $i = 2, \ldots, N$, and $\sum_{i=1}^{N} r_i U_{i,\text{max}} > L - \sum_{i=1}^{N} C_i$. By replacing $U_{i}^* = U_{i,\text{max}}$ for $i \geq 2$ in (5.15), we obtain the value of $U_{1}^*$ exactly as defined in Theorem 5.3. Moreover, if $U_{1}^* \geq U_{i,\text{min}}$ then $\mu_1 \geq 0$. Otherwise, if $U_{1}^* < U_{i,\text{min}}$, then the task set is infeasible and $\mu_1 = 0$. In any case, $\mu_1 \geq 0$.

We have shown that the values of $U_{i}^*$ as defined in Theorem 5.3 satisfy the KKT conditions and form a feasible solution to the problem under consideration. Since the constrained optimization problem is convex, it follows that this feasible solution is also an optimal one [28].

\[ \Box \]

Theorem 5.3 immediately leads to an efficient algorithm to solve the optimization problem in (5.11)–(5.15) when $D_1 + T_1 \leq D_2$. Let us next consider the case where $D_1 + T_1 > D_2$. According to Lemma 5.4, one needs to check whether
$L^* = \min_{i=1}^N (T_i + D_i)$ satisfies (5.6) to determine feasibility. The following theorem forms the basis for solving the optimization problem in (5.11)–(5.15) when $D_1 + T_1 \leq D_2$.

**Theorem 5.4** Given the constrained optimization problem as specified in (5.11)–(5.15), for a fixed value of $L$ (where $L = \min_{i=1}^N \left( \frac{C_i}{U_i} + D_i \right)$) and $\sum_{i=1}^N r_i U_{i_{\max}} > L - \sum_{i=1}^N C_i$. Let

$$
R = \sum_{U_j^* \neq U_{j_{\min}}} \frac{r_j^2}{w_j} - \sum_{U_j^* = U_{j_{\max}}} \frac{r_j^2}{w_j}
$$

$$
\bar{V} = \sum_{U_j^* \neq U_{j_{\min}}} r_j U_{j_{\max}} - (L - \sum_{i=1}^N C_i) + \sum_{U_j^* = U_{j_{\min}}} r_j U_j.
$$

If a solution, $U_i^*$, is optimal then

$$
U_i^* = U_{i_{\max}} - \frac{r_i}{w_i R} \cdot \bar{V}
$$

(5.23)

for $r_i > 0$ and $0 \leq \frac{\bar{V}}{R} \leq \frac{w_i}{r_i} (U_{i_{\max}} - U_{i_{\min}})$, and $U_i^* = U_{i_{\max}}$ for $r_i \leq 0$.

**Proof**

According to Lemma 5.5, the constraint in (5.12) must be active. In other words, any solution to the given optimization problem must satisfy $L_d = L - \sum_{i=1}^N C_i = \sum_{i=1}^N r_i U_i$. We consider the cases where $r_k \leq 0$ and $r_k > 0$ separately.

**Case 1** ($r_k \leq 0$): Consider the KKT conditions given in (5.16)–(5.19). Assume that both constraints in (5.14) and (5.15) are inactive (i.e., $U_{k_{\max}} < U_k^* < U_{k_{\min}}$, $\mu_k = \lambda_k = 0$). Then, (5.16) becomes

$$
2w_k(U_{k_{\max}} - U_k^*) = r_k \mu_0.
$$

(5.24)

However, since $U_k^* < U_{k_{\max}}$, $r_k \leq 0$, and $\mu_0 \geq 0$, the above equation cannot hold. Therefore, for any solution $U_k^*$, either the constraint in (5.14) or (5.15) must be active.
Let us first assume that $\mu_k > 0$ but $\lambda_k = 0$. Then (5.16) gives

$$2w_k(U_{k_{max}} - U_k^*) + \mu_k = r_k\mu_0,$$

which contradicts the assumption that $\mu_k > 0$. Consequently, $U_k^* = U_{k_{max}}$ for $r_k \leq 0$.

**Case 2** ($r_k > 0$): Suppose that the $k$-th constraint in (5.14) is active. That is, $U_k^* = U_{k_{min}}$, $\mu_k > 0$, and $\lambda_k = 0$. Then, from (5.16),

$$\mu_k = r_k\mu_0 - 2w_k(U_{k_{max}} - U_{k_{min}}).$$

If the constraint in (5.14) is inactive, then $\mu_k = 0$. Similarly, if the $h$-th constraint in (5.15) is active, then $U_h^* = U_{h_{max}}$, $\lambda_h > 0$, $\mu_h = 0$ and

$$\lambda_h = -r_h\mu_0$$

and $\lambda_h = 0$ if the constraint in (5.15) is inactive. Multiplying (5.16) by $r_i$, summing it up for all $i$, and using the conclusions above, we have

$$\mu_0 = \frac{2\nabla}{R}$$

By combining (5.28) with (5.16), we get

$$U_i^* = U_{i_{max}} - \frac{r_i\nabla}{w_iR}$$

To enforce the condition of $U_{i_{min}} \leq U_i^* \leq U_{i_{max}}$, $\frac{\nabla}{R}$ must satisfy $\frac{w_i}{r_i} (U_{i_{max}} - U_{i_{min}})$. Summarizing Case 1 and Case 2, we have that a solution to the optimization problem either satisfies $U_i^* = U_{i_{max}}$ for $r_i \leq 0$ or $U_i^* = U_{i_{max}} - \frac{r_i\nabla}{w_iR}$, for $r_i > 0$ and $0 \leq \frac{\nabla}{R} \leq \frac{w_i}{r_i} (U_{i_{max}} - U_{i_{min}})$.
Theorems 5.3 and 5.4 show how an optimal set of task periods can be determined given a fixed value of $L$. We now explain how to use Theorems 5.3 and 5.4 to solve the optimization problem in (5.11)–(5.15) when the value of $L$ may change as the task periods change.

5.3 Proposed Heuristic

The main idea of our heuristic is as follows. Suppose at iteration $h$, a set of periods $T_i(h)$ is found by solving the optimization problem in (5.11)–(5.15). In iteration $h + 1$, we compute the value of $L$ based on Theorem 5.2 using $T_i(h)$. A check is then performed to see whether the constraint in (5.6) is satisfied. If this is the case and if $T_i(h)$ also minimizes the objective function until now, then the algorithm keeps $T_i(h)$ as the current best solution to the problem and continues the search process. Otherwise, if the constraint in (5.6) is not satisfied, we modify the periods found in iteration $h$ in some manner and use them as the periods found in iteration $h + 1$. This process is repeated in an attempt to find the best set of periods that the heuristic can offer. A summary of the heuristic is given in Algorithms 2 and 3. Algorithm 2 shows the main procedure and Algorithm 3 is called by Algorithm 2 to perform a specific set of functions as will be explained below.

We now describe the main procedure (Algorithm 2) in more detail. In each iteration $h$, we fix the value of $L$ as either $L(h) = D_2$ if $T_i(h-1) + D_1 \leq D_2$ or $L(h) = \min_{i=1}^{N}(T_i(h-1) + D_i)$ otherwise (Lines 14–18 in Algorithm 2).

For any task $\tau_i$ whose $r_i = L(h) - D_i \leq 0$, its period is immediately set to $T_{i_{\text{min}}}$ (Lines 32–37 in Algorithm 2). For any task $\tau_i$ whose $r_i > 0$, its utilization, $U_i(h)$, can be determined using Theorem 5.3 (Line 26 in Algorithm 2) or Theorem 5.4 (Line 39 in Algorithm 2), respectively. If $L(h) = D_2$ and $h = 0$, our heuristic
will only require one iteration to find an optimal solution and exit immediately, since the solution set will remain unchanged in subsequent iterations. In the case of \( L(h) = \min_{i=1}^{N}(T_i(h-1) + D_i) \), \( U_i \) is obtained by using a slightly modified task compression algorithm \texttt{Mod_Task_Compress()} (Line 39 in Algorithm 2). The following modifications were made to the original task compression algorithm: (i) the inputs to the task compression algorithm are task set \( \Gamma \) and \( L(h) \), instead of \( \Gamma \) and \( U_d \), and (ii) the equation \( U_i = U_{i_{\text{max}}} - (U_{v_{\text{max}}} - U_d + U_f) E_i/E_v \) in the original algorithm is replaced by (5.23). For the case where \( L(h) = D_2 \), Theorem 5.3 is applied straightforwardly.

As described earlier, during iteration \( h + 1 \), our heuristic will perform a check to determine whether the set of periods found in iteration \( h \) is feasible and if the solution is the best one so far. Algorithm 3 is used to accomplish this task. If the constraint in (5.6) is not satisfied, the heuristic will perform a period “rollback” (Lines 26–33 in Algorithm 3). Essentially, the idea behind a period rollback is to reconsider the current best solution and reduce the corresponding periods by some factor in order to find an even better solution. In our heuristic, the rollback process is controlled by a user-defined parameter, \( prec \), which denotes the starting percentage value for period reductions. The iterative process will terminate when certain stopping criterion is met (to be discussed later). The solution thus found may not be optimal but it is guaranteed to be schedulable by the EDF policy.

If the set of periods found in iteration \( h \) is feasible and if it is the best one we have seen so far, the heuristic will keep track of this solution (Line 14 in Algorithm 3). To improve on the feasible solution, if one has already been discovered, the heuristic will continue until the periods found in iterations \( h \) and \( h + 1 \) are the same. To determine such a solution, a user-defined parameter, \( \Delta \), is included as a stopping criterion; if the difference between \( U_i \) found in the current iteration and \( U_i \) found in
Algorithm 2 Task_Compress_Period($\Gamma$, maxIter)

1: \(\text{sumC} = 0\)
2: \(\text{for each } (\tau_i \in \Gamma) \text{ do}\)
3: \(\text{sumC} = \text{sumC} + C_i\)
4: \(\text{if } (\text{sumC} > D_i) \text{ then}\)
5: \(\text{return} \quad \text{NULL} \quad // \text{By Lemma 5.2, no feasible solution exists}\)
6: \(\text{end if}\)
7: \(\text{end for}\)
8: \(\text{bestObjF} = \infty\)
9: \(\text{for each } (\tau_i \in \Gamma) \text{ do} \quad // \text{Initialize some variables}\)
10: \(\text{prevT}_i = T_{i_{\text{min}}}\)
11: \(\text{currT}_i = T_{i_{\text{max}}}\)
12: \(\text{end for}\)
13: \(\text{for } h = 0, h < \text{maxIter}, h = h + 1 \text{ do}\)
14: \(\text{if } (D_1 + \text{currT}_1 \leq D_2) \text{ then} \quad // \text{Compute } L \text{ using the set of periods from the previous iteration}\)
15: \(L = D_2\)
16: \(\text{else}\)
17: \(L = \min_{i=1}^{\mathcal{N}}(\text{currT}_i + D_i)\)
18: \(\text{end if}\)
19: \(\text{status} = \text{Check\_Record\_Rollback}(\Gamma, L, \text{currT}, \text{prevT}, \text{bestT}, \text{bestObjF}, h)\)
20: \(\quad // \text{The following variables are passed by reference: } \text{currT}, \text{bestT} \text{ and } \text{bestObjF}\)
21: \(\text{if } \text{status} = -1 \text{ then}\)
22: \(\text{return} \quad \text{NULL} \quad // \text{No feasible solution can be found}\)
23: \(\text{else if } \text{status} = 1 \text{ then}\)
24: \(\text{break} \quad // \text{Solution cannot be improved further}\)
25: \(\text{end if}\)
26: \(\text{if } (D_1 + \text{currT}_1 \leq D_2) \text{ then}\)
27: \(\text{Compute } \text{currT} \text{ following Theorem 5.3}\)
28: \(\text{if } (h = 0) \text{ then}\)
29: \(\text{break}\)
30: \(\text{end if}\)
31: \(\text{else}\)
32: \(\text{for each } (\tau_i \in \Gamma) \text{ do}\)
33: \(r_i = L - D_i\)
34: \(\text{if } (r_i \leq 0) \text{ then} \quad // \text{For such a task, set its period to its desired period}\)
35: \(T_i = T_{i_{\text{min}}}\)
36: \(e_i = 0\)
37: \(\text{prevT}_i = T_{i_{\text{min}}}\)
38: \(\text{end if}\)
39: \(\text{end for}\)
40: \(\text{currT} = \text{Mod\_Task\_compress}(\Gamma, L) \quad // \text{Using Theorem 5.4}\)
41: \(\text{end if}\)
42: \(\text{return} \quad \text{bestT}\)
the previous iteration is smaller than \( \Delta i = 1, \ldots, N \), the algorithm terminates and returns the best set of periods it has encountered (Lines 23–25 in Algorithm 3). The same action is taken by the heuristic when the constraint in (5.6) is active (Lines 16–18 in Algorithm 3), where \( \epsilon \) is some small constant. To handle the case where task periods do not converge to some fixed values (or when it may take too long for the solution to converge), the algorithm uses another user-defined parameter, \( maxIter \), to limit the maximum number of iterations.

An additional challenge is how to assign the initial value of \( L \). We propose to set the initial value of \( L \) to \( \min_{i=1}^{N}(T_{i} + D_{i}) \). In this way, if the task set is found to be infeasible, then the algorithm immediately exits since the task set cannot be made schedulable without violating the given period bounds. The following lemma serves to support our choice of \( L \) as well as the iterative approach.

**Lemma 5.6** Let \( T_{i}, i = 1, \ldots, N \), be a set of periods that satisfy the constraint in (5.6) and let \( L = D_{2} \) if \( T_{1} + D_{1} \leq D_{2} \) and \( L = \min_{i=1}^{N}(T_{i} + D_{i}) \) otherwise. Then any set of \( T'_{i} \geq T_{i} \) also satisfy the constraint in (5.6).

**Proof**

Given that \( T'_{i} \geq T_{i}, i = 1, \ldots, N \), it follows that \( L' \geq L \). Let \( L' = L + \Delta L \) where \( \Delta L \geq 0 \). Since \( L \) and \( T_{i}, i = 1, \ldots, N \), satisfy the constraint in (5.6), the following must hold true.

\[
L + \Delta L \geq \sum_{i=1}^{N} \left( \frac{L - D_{i}}{T_{i}} + 1 \right) C_{i} + \Delta L \tag{5.30}
\]

Using the fact that \( \sum_{i=1}^{N} \frac{C_{i}}{T_{i}} \leq 1 \), we obtain

\[
L + \Delta L \geq \sum_{i=1}^{N} \left( \frac{L - D_{i}}{T_{i}} + \frac{1}{T_{i}} \Delta L + 1 \right) C_{i} \tag{5.31}
\]

It follows that

\[
L' \geq \sum_{i=1}^{N} \left( \frac{L' - D_{i}}{T_{i}} + 1 \right) C_{i} \tag{5.32}
\]
The above lemma has two significant consequences. First, if our algorithm cannot find a feasible solution when setting $L(0) = \min_{i=1}^{N}(T_{i,\text{max}} + D_{i})$, it is not fruitful to continue with the algorithm as any smaller $T_{i}$’s would not satisfy the constraints in (5.6). Second, even if a set of feasible periods is found, the algorithm can still attempt to improve on the previously obtained periods in the subsequent iterations. For such iterations $h$, we set $L(h) = \min_{i=1}^{N}(T_{i,h-1} + D_{i})$.

To further improve on the quality of the solutions, we will run our proposed algorithm twice. In the first run, we set the initial value of $L$ to be $\min_{i=1}^{N}(T_{i,\text{max}} + D_{i})$ for the reason mentioned above. In the second run, the initial value of $L$ is set to $\min_{i=1}^{N}(T_{i,\text{min}} + D_{i})$. The same heuristic can be used to accomplish this task with some trivial changes. (Additional checkpoints merely need to be included.) In this way, if a better solution can be found at or near this second value of $L$, the heuristic will be more likely to find it. Finally, the solution from both runs are compared and the better one (i.e., the solution with a smaller objective function value) will be selected.

Let us use the example introduced in Chapter 3 to illustrate the application of Algorithms 2 and 3. Recall that when $\tau_{1}$ needed to decrease its period to 60 time units, the task set is no longer feasible. To find a set of schedulable periods, we make use of our heuristic, setting the elastic coefficient of $\tau_{1}$ to 0, denoting $\tau_{1}$ as a rigid task. The task set is first tested against the necessary condition from Lemma 5.2 to ensure that it can be made schedulable, which is indeed the case. In the first iteration of the main loop (Line 13), $L = T_{1} + D_{1} = 120$ time units and the value for the right-hand side of (5.6) is 119.971 time units, which tells us that the task set is schedulable when $T_{i} = T_{i,\text{max}}$, $i = 2, \ldots, N$. The value of the objective function is about 0.0129. In addition, the current task periods $T_{1} = 60$ time units.
and \( T_i = T_{i\text{max}}, \ i = 2, \ldots, N \), will be recorded as the current best set of periods.

The heuristic now attempts to find a better set of task periods by applying, in this case, Theorem 5.4. The resultant periods are \( T_1 = 60 \) (since it is a rigid task), \( T_2 = 230, T_3 = 178.46, T_4 = 132.27, \) and \( T_5 = 150 \) time units. In the next iteration, \( L = 120 \) time units as before, and the value of the right-hand side of (5.6) is also 120, which makes the constraint in (5.12) active and the heuristic exits. The resultant value for the objective function is now much lower, at about 0.00223.

The following theorem states the correctness and complexity of our proposed algorithm.

**Theorem 5.5** Consider the period selection problem of a task set with \( N \) periodic tasks whose deadlines are less than periods as formulated in (5.11)–(5.15). If there exists a set of task periods such that the task set is schedulable, then Algorithm 2 will always return a feasible solution. That is, the set of periods returned by Algorithm 2 is guaranteed to be schedulable by the EDF policy. In addition, the time complexity of Algorithm 2 is \( O(N^2 \cdot \text{maxIter}) \).

**Proof**

We prove Theorem 5.5 by first considering the case where a feasible solution does not exist. In such a case, when Algorithm 2 calls Algorithm 3 on Line 19, Algorithm 3 will return \(-1\), and consequently cause Algorithm 2 to return NULL in the first iteration. According to Lemma 5.6, if the task set is infeasible when \( T_i = T_{i\text{max}} \), then it cannot be made feasible by our heuristic. Since our heuristic initializes the current set of periods to be \( T_{i\text{max}} \), for all \( i = 1, \ldots, N \) (Line 11 in Algorithm 2), whenever our heuristic returns an empty solution set, it is guaranteed that no feasible set of periods to the optimization problem in (5.11)–(5.15) exists.

Now consider what happens when a solution exists. In such a case, the first set of feasible periods, must be \( T_{i\text{max}} \), since \( \text{curr} \ T_i \) is initialized to \( T_{i\text{max}} \) (Line 11). Additionally, In Algorithm 2, for the set of periods obtained in iteration \( h \), our heuristic performs a check to see whether the task set is schedulable during iteration \( h + 1 \).
Algorithm 3 Check_Record_Rollback(Γ, L, currT, prevT, bestT, bestObjF, h)

1: objF = 0
2: for each (τi ∈ Γ) do // Compute objective value
3:     objF = objF + \frac{1}{e_i}(U_{i_{max}} - C_i/currT_i)^2
4: end for
5: cns = 0
6: for each (τi ∈ Γ) do // Compute the right-hand side of the constraint in (5.6) to check for feasibility
7:     cns = cns + \left(\frac{L-D_i}{currT_i} + 1\right)C_i
8: end for
9: if (cns > L) and h = 0 then // No feasible solution found
10:     return -1
11: else if (cns ≤ L) and (objF < bestObjF) then // Best solution seen so far, so keep it
12:     bestObjF = objF
13:     for each (τi ∈ Γ) do
14:         bestT_i = currT_i
15:     end for
16:     if (cns = L + \epsilon) then // The schedulability constraint is active and the task set is feasible so quit
17:         return 1
18:     end if
19:     for each (τi ∈ Γ) do // Check whether task periods have converged to some fixed values
20:         deltaT_i = |currT_i - prevT_i|
21:     prevT_i = currT_i
22:     end for
23:     if deltaT_i ≤ \Delta then
24:         return 1 // Solution converges
25:     end if
26: else if (cns > L) then // The current set of periods is infeasible, perform a period rollback
27:     prec = prec - 1 // prec is a global variable
28:     if (prec < 0) then
29:         return 1
30:     end if
31:     for each (τi ∈ Γ) do
32:         currT_i = bestT_i - \frac{prec}{100} · bestT_i
33:     end for
34: end if
35: return 0 // Continue improving on solution
(Line 19). When the algorithm terminates, the set of task periods that minimizes the objective function, \( bestT_i, i = 1, \ldots, N \), is returned. From our heuristic, we can see that the only place where \( bestT_i \)'s is updated is on Line 14 of Algorithm 3; the update takes place when the task set with \( T_i(h) \), for all \( i = 1, \ldots, N \), is feasible and if it is the best set of periods that our heuristic has seen so far (i.e., the set of periods that minimizes the objective value until now). In addition, the value \( bestObjF \) is initialized to \( \infty \), which means that the first set of feasible periods will be recorded as \( bestT_i \)'s until a better set of feasible periods is found. Hence, our heuristic will always return a solution, if one exists. In addition, if a solution is returned, then that solution is feasible.

We now examine the time complexity of our heuristic. In [11], Buttazzo et al. proved that the task compression algorithm takes \( O(N^2) \) time. Since the changes made to said algorithm does not affect its time complexity, \( \text{Mod_Task_compress()} \) will also take \( O(N^2) \) time. In addition, since the modified task compression algorithm constitutes the most expensive step in the main for-loop controlled by the user-defined parameter \( maxIter \), the worst-case running time of the proposed heuristic is \( O(N^2 \cdot maxIter) \).

Observe that although our heuristic will always return a feasible solution, if one exists, said solution may not be a global optimal solution to the problem in (5.11)–(5.15). However, as will be shown in Chapter 7, our heuristic finds an optimal solution to over 50% of the test cases.

Finally, with sufficiently small \( maxIter \), the time complexity makes the proposed algorithm suitable for online period adjustments. In Chapter 7, we will provide some guidance on how to adjust user-defined parameters (\( maxIter \), \( \Delta \), and \( prec \)) based on experimental results.
CHAPTER 6

DEADLINE SELECTION FOR REAL-TIME TASKS

In real-time systems, task deadlines are usually considered fixed parameters. However, there exist situations where it may be more suitable to treat the deadline as an adjustable parameter while keeping the task periods constant. For instance, in some control systems, the effects of jitters could severely degrade performance and even result in instability [29]. A popular method to remedy said effects is to reduce task deadlines. More specifically, some control systems compensate for jitters by requiring fixed sampling periods (e.g., task periods) with short deadlines. When an overload occurs, the executing interval is kept unchanged, while the deadlines are increased to improve schedulability. Increasing task deadlines is acceptable as long as the resultant deadlines are within some pre-specified range.

Since we assume that EDF is used as the scheduling policy, adjusting task deadlines really means adjusting scheduling priorities. Contrary to period adjustment, changing task deadlines allows for the workload to be preserved. In other words, the amount of required work remains the same. This last observation is particularly useful when designing a system where a certain amount of work must be done in a specific time interval.

Given a range of allowable deadlines, we redefine our task model from Chapter 2.1 as follows. A task $\tau_i$, $i = 1, \ldots, N$, is characterized by the following 6-tuple: $(C_i, D_i, D_{i_{\text{min}}}, D_{i_{\text{max}}}, T_i, e_i)$, where $D_{i_{\text{min}}}$ and $D_{i_{\text{max}}}$ is the desired and maximum al-
allowable deadlines, respectively, and the rest are the same as defined in Chapter 2.1.

The deadline selection problem can be formulated as:

\[
\begin{align*}
\text{min: } & \quad J(D_1, \ldots, D_N) = \sum_{i=1}^{N} w_i (D_{i_{\text{min}}} - D_i)^2 \\
\text{s.t.: } & \quad \sum_{i=1}^{N} U_i D_i \geq L \sum_{i=1}^{N} (U_i - 1) + \sum_{i=1}^{N} C_i \tag{6.1}
\end{align*}
\]

\[
L = \begin{cases} 
D_2 & : \ D_1 + T_i \leq D_2 \\
\min(T_i + D_i) & : \text{otherwise}
\end{cases} \tag{6.2}
\]

\[
D_i \leq D_{i_{\text{max}}}, \forall i = 1, \ldots, N \tag{6.3}
\]

\[
D_i \geq D_{i_{\text{min}}}, \forall i = 1, \ldots, N \tag{6.4}
\]

where (6.2) is obtained by moving \( D_i \)'s in (5.6) to the left-hand side of the inequality and regrouping the terms.

The above optimization problem has the same form as the optimization problem in (5.11)–(5.15). Since \( L \) cannot be treated as constant, the optimization problem in (6.1)–(6.5) belongs to the class of nonlinear programs. Hence, we take the same approach as in the last chapter. Namely, \( L \) is treated as a constant in each iteration and a heuristic is used to solve the problem efficiently. The following theorem describes the optimal value of \( D_i^* \) for the optimization problem in (6.1)–(6.5) when \( L \) is a constant.

**Theorem 6.1** Given the constrained optimization problem as specified in (6.1)–(6.5), for a fixed value of \( L \) (where \( L = D_2 \) if \( D_1 + T_i \leq D_2 \) and \( L = \min_{i=1}^{N}(T_i + D_i) \) otherwise), and \( \sum_{i=1}^{N} U_i D_{i_{\text{min}}} < L \sum_{i=1}^{N} (U_i - 1) + \sum_{i=1}^{N} C_i \). Let
\[ S = \sum_{D_j^* \neq D_{j_{\text{max}}}} U_j^2 \frac{w_j}{w_j}, \text{ and} \]

\[ \overline{D} = \sum_{D_j^* \neq D_{j_{\text{max}}}} U_j D_{j_{\text{min}}} + \left( L \sum_{j=1}^{N} (U_j - 1) + \sum_{j=1}^{N} C_j \right) \]

\[ - \sum_{D_j < D_{j_{\text{max}}}} U_j D_{j_{\text{max}}}. \]

A solution, \( D^*_i \), is optimal if and only if

\[ D^*_i = D_{i_{\text{min}}} - \frac{U_i}{w_i S} \cdot \overline{D} \] \hspace{1cm} (6.6)

for \( \frac{w_i}{U_i} (D_{i_{\text{min}}} - D_{i_{\text{max}}}) \leq \frac{\overline{D}}{S} \leq 0. \)

**Proof**

The theorem can be proved by the same argument used in proving Theorem 5.4.

\( \square \)

The similarity between the solutions to the optimization problem in (6.1)–(6.5) and those to the one in (5.11)–(5.15) should be apparent. Hence, the heuristic proposed in the last chapter can easily be modified to solve this problem with the same time complexity. Specifically, (6.6) is used instead of (5.23) and the deadline bounds must be checked instead of the period bounds. For completeness, a summary of the heuristic for the deadline selection problem as defined in (6.1)–(6.5) is given in Algorithms 4 and 5. In **Check_Record_Rollback_D()** (Algorithm 5), instead of using \( \text{curr}T_i \) and \( \text{prev}T_i \) as in Algorithm 3, the period is replaced by \( T_i, i = 1, \ldots, N \). In addition, we replace \( D_i \) by \( \text{curr}D_i \) and \( \text{prev}D_i \), \( i = 1, \ldots, N \), as appropriate. We also need to ensure that the condition in Lemma 5.2 is satisfied whenever a new set of deadlines is obtained. As before, some modifications needed to be made to the original task compression algorithm. Namely, the inputs to **Mod_Task_Compress_2()**
are the task set $\Gamma$ and $L(h)$, instead of $\Gamma$ and $U_d$. In addition, the equation $U_i = U_{i_{\max}} = (U_{v_{\max}} - U_d + U_f) E_i / E_v$ in the original algorithm is replaced by (6.6). It is worth noting that, as in the period adjustment problem, our heuristic may or may not find an optimal solution to the problem in (6.1)--(6.5), although it is guaranteed to find a feasible solution, if one exists.

Interestingly, the idea of deadline selection can be extended to treat systems consisting of sporadic tasks. A sporadic task is a real-time task whose arrival time is not known a priori, but there exists some minimum inter-arrival time between any two instances of such task [26]. Although sporadic tasks usually have hard deadlines, for systems where some delays are acceptable, our proposed framework can be used. Specifically, the optimization problem in (6.1)--(6.5) can be straightforwardly applied to adjust the deadline of sporadic tasks with a minor change. That is, $T_i$ now denotes the (known) minimum inter-arrival time of a sporadic task, instead of the period of a periodic task as originally defined.

An important implication of using the optimization problem in (6.1)--(6.5) for a system with sporadic tasks is that, as long as $I_i \geq T_i$, for $i = 1, \ldots, N$, where $I_i$ is the actual inter-arrival time of a sporadic task $\tau_i$, the system will remain schedulable using the set of deadlines obtained from solving the optimization problem in (6.1)--(6.5). Lemma 5.6 can be straightforwardly applied to validate this claim.
Algorithm 4 Task Compress Deadline($\Gamma$, $maxIter$)

1: $sum = 0$
2: for each ($\tau_i \in \Gamma$) do
3:   $sum = sum + \frac{C_i}{T_i}$
4: end for
5: if $sum > 1$ then
6:   return NULL
7: end if
8: bestObjF = $\infty$
9: for each ($\tau_i \in \Gamma$) do // Initialize some variables
10:   $prevD_i = D_{\text{min}}$
11:   $currD_i = D_{\text{max}}$
12: end for
13: for $h = 0, h < maxIter, h = h + 1$ do
14:   if ($currD_1 + T_1 \leq currD_2$) then // Compute $L$ using the set of deadlines from the previous iteration
15:     $L = currD_2$
16:   else
17:     $L = \min_{i=1}^{N} (T_i + currD_i)$
18:   end if
19:   status = Check_Record_Rollback_D($\Gamma$, $L$, $currD$, $prevD$, $bestD$, $bestObjF$, $h$) // The following variables are passed by reference: $currD$, $bestD$ and $bestObjF$
20:   if $status = -1$ then
21:     return NULL // No feasible solution can be found
22:   else if $status = 1$ then
23:     break // Solution cannot be improved further
24:   end if
25: for each ($\tau_i \in \Gamma$) do
26:   $r_i = L - currD_i$
27: end for
28: $currD = \text{Mod Task compress 2}($$\Gamma$, $L$) // Using Theorem 6.1
29: end for
30: return bestD
Algorithm 5 Check_Record_Rollback_D(Γ, L, currD, prevD, bestD, bestObjF, h)

1: objF = 0
2: for each (τi ∈ Γ) do // Compute objective value
3:     objF = objF + 1/ei(Di0 - currDi)^2
4: end for
5: cns = 0
6: for each (τi ∈ Γ) do // Compute the right-hand side of the constraint in (5.6)
7:     cns = cns + (L - currDi)ti + 1)Ci
8: end for
9: if Lemma 5.2 is not satisfied then
10:     necCondFeas ← 0
11: else
12:     necCondFeas ← 1
13: end if
14: if (cns > L) and h = 0 then // No feasible solution found
15:     return -1
16: else if (cns ≤ L) and (necCondFeas = 1) and (objF < bestObjF) then // Best solution seen so far, so keep it
17:     bestObjF = objF
18:     for each (τi ∈ Γ) do
19:         bestDi = currDi
20:     end for
21:     if (cns = L + ε) and (necCondFeas = 1) then // The schedulability constraint is active and the task set is feasible so quit
22:         return 1
23:     end if
24:     for each (τi ∈ Γ) do // Check whether task deadlines have converged to some fixed values
25:         deltaDi = |currDi - prevDi|
26:         prevDi = currDi
27:     end for
28:     if deltaDi ≤ Δ then
29:         return 1 // Solution converges
30: end if
31: else if (cns > L) or (necCondFeas = 0) then // The current set of deadlines is infeasible, perform a deadline rollback
32:     prec = prec - 1 // prec is a global variable
33:     if (prec < 0) then
34:         return 1
35: end if
36:     for each (τi ∈ Γ) do
37:         currDi = bestDi - prec/100 · bestDi
38:     end for
39: end if
40: return 0 // Continue improving on solution
CHAPTER 7

EXPERIMENTAL RESULTS

In this chapter, we begin by verifying our claims made in Chapter 4 with regards to the optimization criterion of the task compression algorithm in [11]. We then compare our simplified sufficient schedulability condition for tasks with deadlines less than periods (Chapter 5) to the exact formula in (5.1). The quality of our heuristic is also discussed both in the context of randomly generated task sets and based on real-world workloads.

7.1 Period Selection with Deadlines Equal to Periods

To verify that the task compression algorithm solves the optimization problem in (4.1)–(4.4), we reuse the task set provided in the experimental results section of [11] (reproduced below in Table 7.1). The task compression algorithm was written in C++, while MatLab was used to obtain the results for the constrained optimization problem in (4.1)–(4.4). All tasks start at time 0 and task deadlines equal task periods. Each task has an initial period of 100 time units. The required minimum utilization of the overall system is $\frac{24}{500} + \frac{24}{500} + \frac{24}{500} + \frac{24}{500} = 0.192$. Since the current utilization is $\frac{24}{100} + \frac{24}{100} + \frac{24}{100} + \frac{24}{100} = 0.96$, the task set is schedulable under EDF.

Assume that, at time 10000, $\tau_1$ needs to reduce its period to 33 time units, perhaps due to some external factors not experienced by other tasks. Since the new required minimum utilization of the system is $\frac{24}{33} + \frac{24}{500} + \frac{24}{500} + \frac{24}{500} = 0.871$, which is
less than 1, $\tau_1$ will be allowed to change its periods as desired. However, $\tau_2$, $\tau_3$, and $\tau_4$ can no longer execute with their initial period, as this would drive the system utilization to $\frac{24}{33} + \frac{24}{100} + \frac{24}{100} + \frac{24}{100} = 1.45$. In other words, to allow $\tau_1$ to change its period, the period of tasks $\tau_2$, $\tau_3$, and $\tau_4$ must increase for the system to remain schedulable. Specifically, $T_2 = 174.1$, $T_3 = 276.4$, and $T_4 = 500$. At time 20000, $\tau_1$ goes back to its original period and the system is no longer overloaded.

Figures 7.1 shows the cumulative number of executed instances for each task as its period changes over time. Recall that the system is initially schedulable until $\tau_1$ needs to reduce its period to 33 time units, which cause the system to be infeasible. At this point, we apply the task compression algorithm and solve the optimization problem in (4.1)–(4.4) to obtain two sets of feasible periods. As can be seen from Figure 7.1, both techniques yield the exact same periods and thus the same number of executed instances of each task.

Observe that although all four tasks have the same timing parameters, their resultant periods are different. This is because the tasks have different elastic coefficients. Namely, both $\tau_1$ and $\tau_2$ are the most rigid tasks (i.e., it is harder to increase their periods as opposed to increasing the periods of $\tau_3$ and $\tau_4$.) As a result, $\tau_2$ is determined to have the smallest (e.g., best) sampling period because of its weight.
On the other hands, $\tau_4$ has the largest sampling period because it is considered to be of least importance. In addition, as shown in Figure 7.1, the number of executed instances of a task is inversely proportional to its elastic coefficient, which, in turns, is inversely proportional to its weight.

7.2 Period Selection with Deadlines Less Than Periods

To illustrate the practicality and performance of our heuristic approach, we present the following comparisons in this section. First, we compare the simplified sufficient condition in (5.6) with the original necessary and sufficient condition in (5.1). Second, we demonstrate the performance of the proposed heuristic by comparing the number of problems it is able to solve with what can be solved by an optimization solver for the optimization problem in (5.11)–(5.15). Third, to assess the solution quality of the heuristic, we compare the solutions obtained using the heuristic with the optimal solutions.
To perform the aforementioned comparisons, 1000 task sets consisting of 5 tasks each were randomly generated for 9 different utilization levels ($U_{\text{level}} = 0.1, \ldots, 0.9$) with a total of 9000 task sets in total. The utilization level is defined to be $U_{\text{level}_i} = \sum_{j=1}^{N} \frac{C_j}{T_{j_{\text{max}}}}, i = 0.1, \ldots, 0.9$. Each task set is initially unschedulable with $T_{i_{\text{min}}} = D_i$, $i = 1, \ldots, N$, but at least a feasible schedule can be found by setting $T_i = T_{i_{\text{max}}}$, $i = 1, \ldots, N$, using the necessary and sufficient condition in (5.1). In addition to the utilization level, the maximum hyperperiod, minimum period, maximum period, precision, and maximum number of tries must also be specified. In our experiment, we set the maximum hyperperiod, minimum period, and maximum period to 500,000, 10,000, and 40,000, respectively. The precision was specified to be 100, whereas the maximum number of tries was set to 10,000. The precision denotes the minimum increment in any task period. For example, if the precision is set to 100, a task period could be 5200, but not 5010.

In a nutshell, the following steps were taken to generate a task set. First of all, a set of periods were randomly generated based on the minimum period, maximum period, hyperperiod bound, and precision. Task periods are generated in such a way that the hyperperiod is no larger than the maximum hyperperiod. (This could take a number of tries.) Each task is randomly assigned an execution time such that the total utilization equals that specified by the user. No task will have a utilization that is greater than half of the specified total utilization. Then, each task is assigned a deadline that ensure that $\sum_{i=1}^{N} \frac{C_i}{D_i} > 1$. As a final step, the random task set generator tests the schedulability of a task set using the necessary and sufficient condition in (5.1). If the task set is unschedulable, task deadlines are randomly increased such that the new deadline is greater than the previous deadline but $\sum_{i=1}^{N} \frac{C_i}{D_i}$ is still greater than 1. This final step is repeated until either a feasible task set has been found or the maximum number of tries has been reached.
To perform the experiments for comparing the simplified sufficient condition in (5.6) with the original condition in (5.1), we make the following observation. We know that the randomly generated task sets pass the test in (5.1), provided that we use $T_{i_{\text{max}}}$ as the period for each task $\tau_i$, $i = 1, \ldots, N$. Hence, we only need to test the condition in (5.6) in the same fashion.

Figure 7.2 compares the performance of the condition in (5.6) to that in (5.1). The x-axis shows each utilization level and the y-axis indicates the percentage of task sets that were found to be feasible by the condition in (5.6). (Recall that the percentage of schedulable task sets using the test in (5.1) is 100% for each utilization level.) As can be seen from the plot, the simplified sufficient condition in (5.6) found that a feasible solution exists for all task sets with $U_{\text{level}} \leq 0.3$. As expected, said condition becomes more pessimistic as $U_{\text{level}}$ increases. However, it still finds over 50% of the task set with $U_{\text{level}}$ of 0.6 to be feasible. Note that we also test each task set against the existing sufficient condition (a task set is schedulable if $\sum_{i=1}^{N} \frac{C_i}{D_i} \leq 1$ [26]), but said test will always fail, since the task set generator returns a task set while guaranteeing that $\sum_{i=1}^{N} \frac{C_i}{D_i} > 1$. Based on these results, we can conclude that the condition in (5.6) is not too pessimistic, especially at low utilization levels.

In the second experiment, we compare the percentage of solutions found by our heuristic, as opposed to solving the optimization problem directly using an optimization solver. The purpose of this comparison is to illustrate the fact that a non-linear solver cannot guarantee that a solution to the problem defined in (5.11)–(5.15) will be found, even if one exists (although a solution returned by a non-linear solver is an optimal one). Our heuristic was implemented in C++. For the optimization solver, we used Loqo [38], a nonlinear solver with an interior-point algorithm. In our heuristic, the user-defined parameters $\text{maxIter}$ and $\Delta$ were set
to 200 and $1 \times 10^{-5}$, respectively. The precision parameter, $prec$, used in period rollback as described in Section 5.3, was set to 20 (denoting 20\% period reductions).

We limited the maximum number of iterations allowed by Loqo to be 200; we did not allow Loqo to run longer, as, according to [39], it is unlikely that an optimal solution would be found after the 100th iteration.

Figure 7.3 compares the number of solutions found by Loqo and that found by our heuristic. As before, the x-axis shows the different utilization levels, whereas the y-axis shows the percentage of solutions found. It is clear from the plot that the heuristic is able to find a much larger number of solutions than Loqo. As previously mentioned, an optimization software does not guarantee that a solution to a non-linear programming problem will be found, even if one exists. This is one important advantage of the heuristic; according to Theorem 5.5, it is guaranteed to always return a feasible task set, if one exists.

Figure 7.2. Performance of simplified sufficient condition
The last experiment examines the quality of the solutions found by our heuristic as opposed to the optimal solutions found by Loqo. For the 9000 task sets (1000 task sets for each utilization level), Loqo only solved 1416 task sets optimally. Therefore, we could only compare the results from the heuristic to these solutions. It must be emphasized, however, that the solutions from the heuristic is a superset of those from Loqo.

Figure 7.4 shows a bar chart depicting the solution quality of the proposed heuristic as compared to the optimal solutions found by Loqo. The x-axis shows the differences in the objective function values between the heuristic and Loqo. The y-axis shows the percentage of the solutions by our heuristic that result in a given difference. The first, and most obvious, observation is that the heuristic is able to find the global optimal solution to over 50% of the solutions when numerical errors...
are ignored. Also, almost an additional 30% of the solutions found by the heuristic is very close to the global optimal ones.

Figure 7.4. Comparison of the heuristic and Loqo in terms of optimality of solutions

The above experimental results demonstrate that our heuristic performs well enough to be deployed in real applications; not only does the heuristic find the global or close to global optimal solution in many cases, it also guarantees to always return a feasible schedule if one exists. In addition, the low time complexity of our heuristic makes it suitable for online use for dynamic period adjustments. The experiment also suggests that the maximum number of iterations, \( \text{maxIter} \), need not be greater than 200 for the heuristic to find a solution. The user-defined parameter \( \Delta \) can be set to be equal to the time granularity used by the operating system, since this time granularity is the smallest time unit that the operating system can handle. Finally, we suggest setting the period rollback precision, \( \text{prec} \), to some small value compared to the value of \( \text{maxIter} \). (It does not make sense to set \( \text{prec} \) to be a large value (e.g. 50) if \( \text{maxIter} \) is relatively small (e.g. 200), as the heuristic will then be performing the rollback process most of the time.)
7.3 Experimental Data for Real-World Workloads

Using a large number of randomly generated task sets, we have shown that our heuristic can find many optimal or near-optimal solutions. However, it is important to quantify the performance of the heuristic under real-world workloads. In this section, we test our heuristic using three benchmarks, whose task sets reflect the type of applications that may require online period adjustment during overload conditions. The applications we consider are: a videophone application, a computerized numerical control (CNC) application, and an avionics application. We will describe the benchmarks one by one and report the results.

7.3.1 Videophone Application

A benchmark for a typical videophone application was presented in [37]. The task set consists of four real-time tasks: video encoding, video decoding, speech encoding, and speech decoding. The worst-case execution time and period for each task is given. The period values from the benchmark are considered as maximum periods $T_{i,\text{max}}, i = 1, \ldots, 4$. For testing purpose, we randomly generate task deadlines using a uniform distribution. Task deadlines are guaranteed to be at least 95% of the period and are less than the periods. In addition, task deadlines are generated such that the task set is unschedulable when $T_i = D_i, i = 1, \ldots, 4$. Finally, since the video encoding task is said to have the lowest priority, its weight is set to 1, while the weights of the other tasks are set to 2.

We apply our heuristic to the task set to solve the period adjustment problem when task deadlines are less than task periods. The heuristic required only 8 iterations for the periods to converge to optimal period values. On the other hand, Loqo needed 20 iterations in total.
7.3.2 Computerized Numerical Control Application

The authors in [20] presented a benchmark which consists of 8 tasks with measured execution times and derived periods and deadlines of some CNC controller tasks. The task set is initially feasible. However, we consider situations where the task execution times may be longer and the system is overloaded (e.g., the primary computer is down and the backup computer is less powerful). To illustrate such a scenario, we multiply each task execution time by a factor of 1.6 to make the task set unschedulable. The weight for each task is set to 1 since the benchmark in [20] lacks priority specifications for each task.

Using Loqo, an optimal set of task periods is found, requiring 93 iterations in total. On the other hand, our heuristic was able to find an optimal solution using only 3 iterations.

7.3.3 Generic Avionics Application

In [27], a benchmark containing 17 periodic tasks and an aperiodic task was given for a Generic Avionics Platform (GAP). Said benchmark contains a priority for each task, which directly translates to its weight. The period and execution time of each task is also given. Since task deadlines were not given, we randomly generate them to be between 50-60% of the periods using a uniform distribution. In addition, since the GAP benchmark dates back to 1991, we reduce each task execution time by half. The resultant task set (with 17 periodic tasks) is schedulable.

Now, assume that at some point in time, the aperiodic task arrives with a specific deadline and a minimum inter-arrival time. The system is no longer schedulable and therefore we apply our heuristic to find a set of feasible periods, which it does within 4 iterations. Unfortunately, we are unable to assess the quality of our solution, since Loqo could not find a solution within 200 iterations.
In this work, the following main contributions were made. First, we introduced a
general framework which formulates a trade-off between task set schedulability and
a specific performance metric (such as task utilization) as an optimization prob-
lem. Such a framework allows for real-time systems under temporal overloads to
graciously adapt by adjusting their performance level. Second, we proved the opti-
mality of the existing task compression algorithm in [11]. Said algorithm allows for
the period selection problem for tasks with deadlines equal periods to be solved op-
timally in an online manner. Third, our framework is further generalized to consider
situations where task deadlines are less than task periods. A sufficient schedulability
condition is proposed to solve the problem efficiently. Our heuristic, which contains
a slightly modified task compression algorithm, can be used to solve the problem
online. Experimental results show that the heuristic performs satisfactorily and in
many cases finds the global optimal solution. Fourth, we provide and motivate the
use of our framework for solving the deadline selection problem, which can be ap-
p lied to control systems with fixed sampling times. Last, our flexible framework can
be used to solve other problems where the schedulability-performance trade-off is
central. The framework also permits the development and comparisons of efficient
algorithms.

Since the algorithm presented in this work is best-effort, it would be interesting
to study whether there exists a way to select the value of $L$ at every iteration such that the solution found will always be optimal. It would also be useful to explore different classes of objective functions and constraints that may be even harder to solve. Finally, while the task models used in this work allow for either the task periods or task deadlines to be adjustable, in some systems, changing the task periods imply changing the task deadlines. That is, there is a tight coupling between task periods and task deadlines and an overload management scheme for this type of tasks still need to be researched on.


