An Efficient Knapsack-Based Approach for Calculating the Worst-Case Demand of AVR Tasks

Sandeep Kumar Bijinemula

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Dec. 6, 2018
Arlington, Virginia

Keywords: Adaptive variable rate task, demand bound function, worst-case demand, knapsack problem

An Efficient Knapsack-Based Approach for Calculating the Worst-Case Demand of AVR Tasks

Sandeep Kumar Bijinemula

(ABSTRACT)

Engine-triggered tasks are real-time tasks that are released when the crankshaft arrives at certain positions in its path of rotation. This makes the rate of release of these jobs a function of the crankshaft's angular speed and acceleration. In addition, several properties of the engine triggered tasks like the execution time and deadlines are dependent on the speed profile of the crankshaft. Such tasks are referred to as adaptive-variable rate (AVR) tasks. Existing methods to calculate the worst-case demand of AVR tasks are either inaccurate or computationally intractable. We propose a method to efficiently calculate the worst-case demand of AVR tasks by transforming the problem into a variant of the knapsack problem. We then propose a framework to systematically narrow down the search space associated with finding the worst-case demand of AVR tasks. Experimental results show that our approach is at least 10 times faster, with an average runtime improvement of 146 times for randomly generated task sets when compared to the state-of-the-art technique.

This research was supported in part by the US National Science Foundation (CNS Grant No. 1618979)

This thesis is based on the paper by Bijinemula et al. [1]

An Efficient Knapsack-Based Approach for Calculating the Worst-Case Demand of AVR Tasks

Sandeep Kumar Bijinemula

(GENERAL AUDIENCE ABSTRACT)

Real-time systems require temporal correctness along with accuracy. This notion of temporal correctness is achieved by specifying deadlines to each of the tasks. In order to ensure that all the deadlines are met, it is important to know the processor requirement, also known as demand, of a task over a given interval. For some tasks, the demand is not constant, instead it depends on several external factors. For such tasks, it becomes necessary to calculate the worst-case demand.

Engine-triggered tasks are activated when the crankshaft in an engine is at certain points in its path of rotation. This makes their activation rate dependent on the angular speed and acceleration of the crankshaft. In addition, several properties of the engine triggered tasks like the execution time and deadlines are dependent on the speed profile of the crankshaft. Such tasks are referred to as adaptive-variable rate (AVR) tasks. Existing methods to calculate the worst-case demand of AVR tasks are either inaccurate or computationally intractable. We propose a method to efficiently calculate the worst-case demand of AVR tasks by transforming the problem into a variant of the knapsack problem. We then propose a framework to systematically narrow down the search space associated with finding the worst-case demand of AVR tasks. Experimental results show that our approach is at least 10 times faster, with an average runtime improvement of 146 times for randomly generated task sets when compared to the state-of-the-art technique.

Dedicated to my parents.

Acknowledgements

First of all, I would like to thank Dr. Chantem for being an amazing advisor. It has been a great pleasure to work with and learn from her. She ticks all the boxes for a perfect mentor. She was always open to new ideas while being patient when I messed up. I'm fortunate to have received 2 years of her mentorship.

I am grateful to Dr. Fisher for his invaluable suggestions and ideas. He played a crucial role in shaping my research. I am also thankful to Aaron, my research collaborator, for his hard work and dedication in completing our project.

I'd like to thank my committee members, Dr. Ryan Gerdes and Dr. Guoqiang Yu, for their wise advice and insightful comments, which greatly helped me prepare for the RTSS conference.

I'm blessed to have the greatest parents. They supported me in all my endeavors. If not for them I'd not be the person I am today. Thank you, Mom and Dad for being who you are. I dedicate this thesis to you. I'm thankful to my brother, Santhosh, for being my partner in crime and for always being there for me.

Last but not least, I'd like to thank my girlfriend, Priya, for her unconditional support through this journey. She has always encouraged me even when I'm in doubt. She has seen me succeed, she has seen me fail but she never failed to support me. I'm lucky to have her in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In real-systems both the accuracy of the operation and the time instant at which the operation is executed are important. An example of such a system is the breaking system of a car.

The timing constraints in real-time systems are enforced by assigning deadlines to each task. While meeting deadlines is an important criteria for a system to be classified as a real-time system, not all the deadlines are equally critical. For some real-time systems, missing a deadline can cause failure of the system while for others it might be tolerable to miss a few of the deadlines without any major degradation of the service. For example, the breaking system of a car is considered as a hard real-time system as a delay in the breaking of a car is catastrophic while missing a few deadlines in a video streaming application is acceptable.

In several real-time systems like the power-train control module (PCM) of cars, the processing power is limited. Such systems require good strategies for efficient resource management. One of the important factors to be considered for devising resource management strategies is the workload characterization. Workload characterization especially in the worst-case gives a measure of the maximum resource requirement of a task. This measure is essential when designing resource distribution strategies in hard real-time systems. It is relatively easy to calculate the worst-case workload of tasks which spawn at fixed intervals of time. Such tasks whose time period is constant are referred to as periodic tasks. PCM of a car manages some tasks that are periodic and some other tasks like fuel injection and ignition that are event-triggered and not time-triggered. Some of these event triggered tasks, referred to as the engine-triggered tasks, are initiated when the car engine's crankshaft is at certain positions with respect to fixed points in its path of rotation. The rate of release of these tasks is proportional to the speed of the crankshaft-as the crankshaft rotates faster, more of these tasks are released.

Each instance of a task is called a job. If the execution time of the jobs is assumed to be the same at all speeds, the worst-case resource requirement, also referred to as the maximum demand, of these tasks can be obtained when the crankshaft rotates at its highest speed. However, the crankshaft is rarely operated at the highest speed, scheduling tasks assuming this as the worst-case demand will lead to underutilization of the resources. Moreover, the engine is stable at high crankshaft speeds due to frequent sensor and actuation updates [5]. Hence, jobs that are released at high speeds can skip certain checks and functions. Reflecting this, a model called the AVR task model was proposed to describe the engine controlled tasks [5]. An AVR task is characterized by a set of execution modes, where each mode is defined over a range of speeds with each mode expressing a different functionality. An example AVR task is shown in Fig. 1.1. Further, several properties like inter-arrival times and deadlines of the jobs are also dependent on the crankshaft's speed profile.

To determine whether an AVR task is schedulable using EDF, the demand bound function (dbf) is often used [6, 7] to measure the resource requirement over a given time interval. In a nutshell, dbf determines the worst-case aggregate execution time of the jobs that have both the arrivals and deadlines within a time interval $[t_1, t_2]$. In general, the calculation of the worst-case demand of an AVR task is not straightforward. Let us consider an example. In a time interval $[t_1, t_2]$, assume 15 jobs are released at the highest allowable speed with each job having an execution time of $50\,\mu s$. On the other hand, during the same duration, assume 10 job releases are possible at the lowest speed with each job having an execution

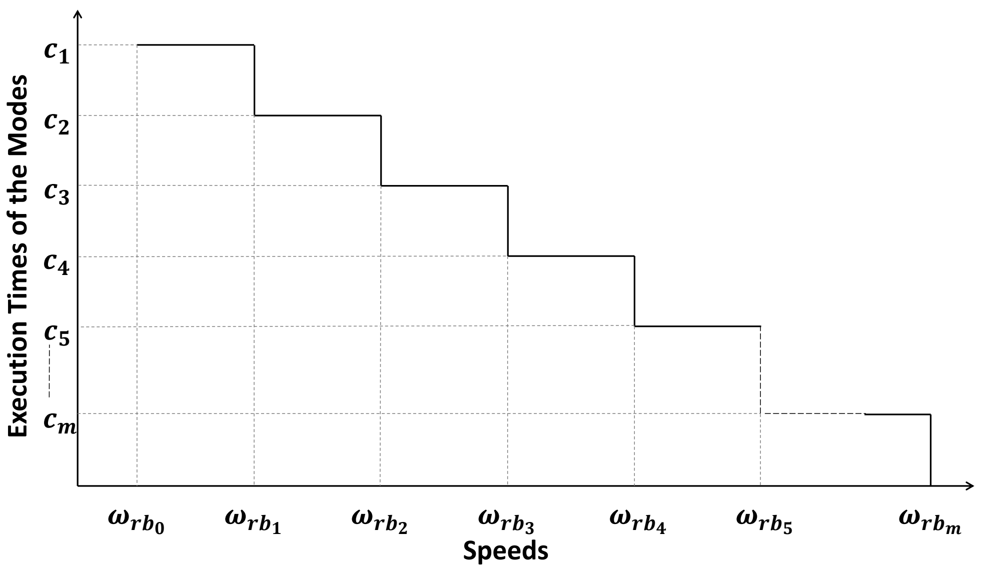


Figure 1.1: Different modes of an AVR task where c_i and ω_{rb_i} are the execution time and the right boundary speed of the i^{th} mode, respectively.

time of $100\,\mu$s. The demand when the jobs are released at the lowest speed (1,000 μs) is greater than when the jobs are released at the highest speed ($750\,\mu s$). Suppose instead that the jobs that are released at the highest speed have an execution time of $70\,\mu s$. In this case, the demand of these jobs is greater than the demand of the jobs that are released at the lowest speed. Hence, the demand depends on the relationship between the task's execution times and speeds, as well as the acceleration profiles of the engine.

Several methods have been proposed to calculate the dbf of an AVR task. Mohaqeqi et al. [4] proposed an exact analysis, using the Digraph model [8], to transform an AVR task into a digraph to calculate the exact worst-case demand assuming that the crankshaft can have multiple acceleration values during a rotation. While this approach represents the state-of-the-art technique, it is computationally intensive and unlikely to be suitable for large problem instances. In this thesis, we propose a knapsack-based method to efficiently calculate the exact worst-case demand of an AVR task.

Chapter 2

Related Work

Buttle [5] in his keynote at ECRTS 2012 conference, was the first to identify the problem of analysis of rate dependent tasks in the ECUs of vehicles. Kim et al. [9] then studied the usage of multiple worst-case execution times and periods for engine-controlled tasks. They proposed the rhythmic task model and obtained basic schedulability results. However, their analysis is limited to a single AVR task scheduled along with periodic tasks using fixed-priority scheduling in which the AVR task has the highest priority. Pollex et al. [10], perform sufficient real-time analysis on rate dependent tasks assuming arbitrary but a constant angular speed. In a later work, Pollex et al. [11] expand their analysis by assuming a non-zero acceleration, hence allowing speed variations and mode changes and present sufficient schedulability tests. However, they quantize the speed domain. Since speed is a continuous variable, quantization of the speed domain introduces pessimism in the analysis. David et al. [12] present several sufficient schedulability tests for rate-dependent tasks. Guo and Baruah [13] proposed a sufficient schedulability test for EDF scheduling. Evaluations show that employing these simple tests provides a better performance than treating AVR tasks as sporadic tasks [14].

Biondi et al. [3] presented the calculation of the processor requirement of AVR tasks in fixed priority scheduling as a search problem in the speed domain. They transformed the problem into a search tree, where the nodes at each level of the tree are obtained based on the acceleration and speed limits of the crankshaft. The infinite paths from the root to the leaves in the search tree were narrowed down by employing certain criteria. A similar method of filtering through the infinite state space was applied and exact schedulability tests were dervied for rate monotonic [6] and EDF scheduling [7]. These works make a simplying assumption that the acceleration of a crankshaft remains constant through a rotation.

In a recent paper, Biondi et al. [15] propose a task model to express some realistic features of AVR tasks and present linear-time and quadratic-time schedulability tests based on utilization bounds.

In a different line of research, Biondi et al. [16] propose methods to calculate the speed of the crankshaft at each point in its point of rotation. In yet another direction of research, Biondi et al. [17] present optimization techniques to model the switching states of AVR tasks.

Mohaqeqi et al. [4] partitioned the speed ranges into sub-ranges and represented each of these sub-ranges as a node of a directed graph. The worst-case demand was then calculated by employing a search from each of these nodes. Mohaqeqi et al. [4] were also the first to relax the assumption of constant acceleration within a rotation. Though their approach gives the accurate value of the worst-case demand, many of the paths that are considered in the digraph are redundant and add unnecessary overhead to the calculation of the demand. In this thesis, we present a method to eliminate all the unnecessary paths and compute the worst-case demand significantly faster.

Chapter 3

Preliminaries

In this chapter, we introduce the task model and provide some background on the physical properties of the crankshaft. We then formally define the problem.

3.1 Task Model

The PCM of an automotive engine performs certain tasks at each of the stages of the rotation of a crankshaft as shown in Fig. 3.1. For the optimal performance of the engine, these tasks should occur at precise angles with respect to the top dead center position of the engine. Further, as the rate of arrival of the crankshaft at a given angle varies with the speed of the engine, the rate of release of the above tasks is not fixed, instead it depends on the speed profile of the crankshaft. At high speeds, a large number of instances of these tasks are released and the PCM could be overloaded which can lead to several deadline misses. In order to schedule using Earliest deadline first (EDF scheduling), it is necessary to calculate the worst-case resource requirement of these tasks over time. For these tasks, the worst-case demand can be obtained by assuming that the crankshaft rotates at the maximum speed always. While this may give the accurate value, it is not practical to

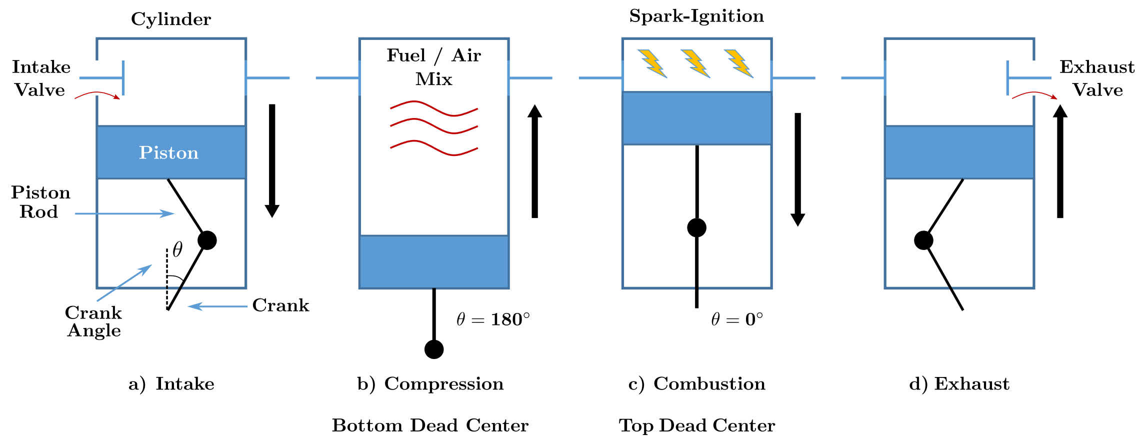


Figure 3.1: Different stages of fuel ignition in a vehicle.

assume that the crankshaft rotates at the maximum speed always. This assumption results in the underutilization of resources. In order to tackle this problem, the execution time of these tasks is made a function of the speed of the crankshaft. Taking into account the fact that the engine is stable at higher speeds [5], the functionality of these tasks and hence their execution time is reduced as the speed increases as shown in Fig. 1.1. Since their execution time varies with the speed, they are referred to as Adaptive Variable Rate (AVR) tasks.

The state space is divided into modes where the range of speeds having the same execution time belong to a mode. The border speeds of the modes are called the *boundary speeds* of the mode. The speeds where the crankshaft switches modes are referred to as the *right boundary speeds* of the modes and they are collectively defined as $\mathbf{\Omega_{rb}} = \{\omega_{rb_1}, \ldots, \omega_{rb_m}\}$, $\forall \omega \in (\omega_{rb_{i-1}}, \omega_{rb_i}]$, speed ω is in the i^{th} mode. The execution time of the i^{th} mode is represented by c_i. Additionally, the execution time corresponding to a speed ω, is denoted by $c(\omega)$.

The minimum ($\omega_{\min}$) and maximum ($\omega_{\max}$) allowable speeds of rotation of the crankshaft are equal to ω_{rb_0} and ω_{rb_m} respectively. The maximum allowable acceleration and deceleration are denoted by $\alpha_{\max}$ and $\alpha_{\min}$, respectively. In this thesis, we make a simplifying assumption, $\alpha_{max} = |\alpha_{min}|$. For consistency we assume speed (ω) to be in rev/min and acceleration (α) to be in rev/min^2. A table of notations used in the thesis can be found in Appendix A.1.

We assume that a single instance of an AVR task, i.e a single job, is released in a rotation. Further, without the loss of generality, the jobs are assumed to be released at the top dead center position. So, the speed at the top dead center position determines the execution time of the job.

3.2 System Dynamics

In this section, we review some basic kinematics formulae.

The angular distance traversed by the crankshaft when the initial speed is ω and a constant angular acceleration of α is applied for a time t, $\Delta\theta$ is [18],

$$\Delta\theta = \omega t + \frac{1}{2}\alpha t^2 \tag{3.1}$$

The total angular distance covered by the crankshaft over a time interval $[t_1, t_2]$ can also be calculated as [18],

$$\Delta\theta = \int_{t_1}^{t_2} \omega(t)dt. \tag{3.2}$$

This implies that the angular distance, $\Delta\theta$, is equal to the area under the ω vs. *time* curve. Since a single job is assumed to be released for every rotation, the number of jobs released increases for every 2π radian (= 1 rotation) covered by the crankshaft.

The speed after an angular displacement of $\Delta\theta$ when the initial speed is ω and a constant angular acceleration of α is applied is [18],

$$\Omega(\omega, \alpha, \Delta\theta) = \sqrt{\omega^2 + 2\alpha\Delta\theta}. \tag{3.3}$$

Since speed (ω) is in rev/min and acceleration (α) is in rev/min^2, angular displacement is measured in number of rotations. So, $\Delta\theta = 1$ indicates a complete rotation. Hence, from Equation 3.3, the speed after one complete rotation when an initial speed of ω, and a constant acceleration of α is assumed is, $\Omega_1(\omega, \alpha) = \sqrt{\omega^2 + 2\alpha}$. In general, the speed after n complete rotations of constant acceleration of α is [4],

$$\Omega_n(\omega, \alpha) = \sqrt{\omega^2 + 2n\alpha}. \tag{3.4}$$

Multiple AVR tasks that have the same phase and period and which are activated by the same source can be modeled as a single AVR task, called the representative AVR task [6]. Hence, the analysis in this thesis can also be extended to multiple AVR tasks.

Definition 1 (Reachable Speeds). *Consider two speeds ω_1 and ω_2 such that $\omega_2 \geq \omega_1$. ω_2 is said to be* reachable *from ω_1 if $\Omega_1(\omega_1, \alpha_{max}) \geq \omega_2$. In other words, ω_2 is said to be* reachable *from ω_1 if it is possible to accelerate from ω_1 to ω_2 in a single rotation while obeying the acceleration bounds.*

3.3 Minimum Job Inter-arrival Times

Consider two speeds ω_1 and ω_2 such that $\omega_2 \geq \omega_1$ and ω_2 is reachable from ω_1. The minimum inter-arrival time ($\widetilde{T}(\omega_1, \omega_2)$) is the shortest time it takes for the crankshaft to initiate the release of a job at ω_1 and the next job at ω_2. Since the speed at the end of the current rotation is the initial speed of the next rotation, mininum inter-arrival time can also be defined as the minimum time duration from the start of a rotation at ω_1 till the end of the rotation at ω_2. Most existing work [3, 6, 7] assume a constant acceleration within a rotation. However Mohaqeqi et al. [4] prove that assuming a constant acceleration does not always give the minimum inter-arrival time. So, we consider the possibility of acceleration variations within a single rotation.

We present the minimum inter-arrival time equations by Mohaqeqi et al. [4] below for readability. Minimum inter-arrival time is obtained by first accelerating from ω_1 to a speed $\omega_p(\omega_1, \omega_2)$, or simply ω_p, using α_{max} and then using α_{min} to decelerate from ω_p to ω_2. The value of ω_p is given by,

$$\omega_p(\omega_1,\omega_2) = \omega_p = \sqrt{\frac{\alpha_{max}\cdot\omega_2^2 - \alpha_{min}\cdot\omega_1^2 - 2\alpha_{min}\cdot\alpha_{max}}{\alpha_{max}-\alpha_{min}}} \tag{3.5}$$

If $\omega_p(\omega_1,\omega_2) \le \omega_{\max}$,

$$\widetilde{T}(\omega_1,\omega_2) = \frac{\omega_p-\omega_1}{\alpha_{max}} + \frac{\omega_2-\omega_p}{\alpha_{min}}$$

If $\omega_p > \omega_{\max}$, then minimum time is obtained by using α_{max} from ω_1 till $\omega_{\max}$ is reached for a time duration of (t_1^*). Then the acceleration is made zero for some time (t_2^*). Finally, α_{min} is used from $\omega_{\max}$ till ω_2 for a duration of (t_3^*).

$$t_1^* = (\omega_{\max}-\omega_1)/\alpha_{max}$$
$$t_2^* = \frac{1}{\omega_{\max}}\left(1 - \frac{\omega_{\max}^2-\omega_1^2}{2\alpha_{max}} - \frac{\omega_2^2-\omega_{\max}^2}{2\alpha_{min}}\right)$$
$$t_3^* = (\omega_2-\omega_{\max})/\alpha_{min}$$
$$\widetilde{T}(\omega_1,\omega_2) = t_1^*+t_2^*+t_3^*$$

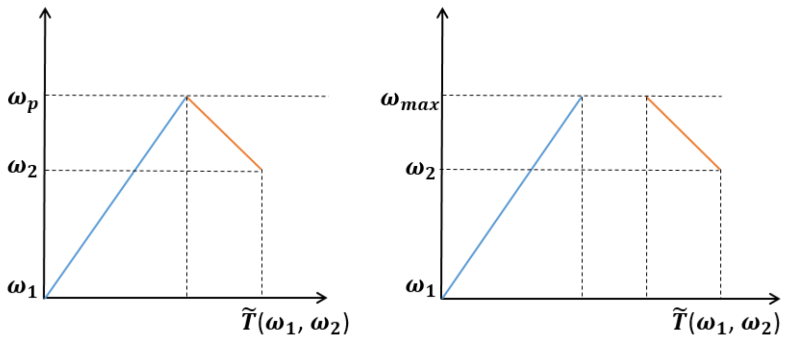
The two cases are defined below and Figure 3.2,

$$\widetilde{T}(\omega_1,\omega_2) = \begin{cases} \frac{\omega_p-\omega_1}{\alpha_{max}} + \frac{\omega_2-\omega_p}{\alpha_{min}} & \omega_p(\omega_1,\omega_2) \le \omega_{\max} \\ \\ t_1^*+t_2^*+t_3^* & \omega_p(\omega_1,\omega_2) > \omega_{\max} \end{cases} \tag{3.6}$$

Note that the value of Equation 3.6 is not altered when ω_1 and ω_2 are interchanged.

3.4 EDF scheduling [2]

We first briefly describe the relatively simple case of calculating demand of a periodic task scheduled by EDF scheduling. Consider a system with only periodic tasks, τ_i. Each of these



(a) $\widetilde{T}(\omega, \omega_2)\,|\omega_p \leq \omega_{\max}$ (b) $\widetilde{T}(\omega, \omega_2)\,|\omega_p > \omega_{\max}$

Figure 3.2: Minimum interarrival time $\widetilde{T}(\omega, \omega_2)$ between speeds ω and ω_2 where (a) $\omega_p \leq \omega_{\max}$ and (b) $\omega_p > \omega_{\max}$. Ascending, flat, and descending lines represent periods of maximum ($\alpha_{\max}$), zero, and minimum ($\alpha_{\min}$) acceleration, respectively.

tasks has a time period (T_i), an absolute deadline ($\widetilde{D_i}$) and an execution time (C_i). Each job of a task has a release time (R_{ij}), and a relative deadline ($\tilde{d}_{ij}$). EDF scheduling is a dynamic priority scheduling policy where the job with the earliest absolute deadline is given the highest priority.

The processor time demanded by all the jobs of a task that are released on or after t_a and end on or before t_b is called the demand of a task in a time interval $[t_a, t_b]$. Formally, the demand can be mathematically expressed as,

$$D_{\tau_i}(t_a, t_b) = \sum_{j:(R_{ij} \geq t_a) \wedge (R_{ij} + \tilde{d}_{ij} \leq t_b)} c_i \tag{3.7}$$

According to the processor demand criterion [19], if the demand of all the tasks in a uniprocessor environment does not exceed the available time, then the task is schedulable under EDF scheduling. Mathematically a taskset is schedulable if,

$$\sum_i D_{\tau_i} \leq t_b - t_a \tag{3.8}$$

To measure the schedulability of AVR tasks under EDF scheduling, we need to calculate their demand. Since the demand of AVR tasks might change depending on the speed profile of the crankshaft, we need to calculate the worst-case demand of AVR tasks to ensure schedulability in all the scenarios. In the following sections of this chapter, we present a mathematical formula for the worst-case demand of AVR tasks and also formally define the problem.

3.5 AVR Task Demand [1]

Let $\omega(t)$ denote the speed at time t and $\mathbb{W}$ represent the set of all feasible speed functions (i.e., $\omega(t)$ is any continuous function following the physical constraints of the crankshaft). For any job released at time t', we assume that the relative deadline is given by $\tilde{d}(\omega) = \widetilde{T}(\omega(t'), \min(\Omega_1(\omega(t'), \alpha_{\max}), \omega_{max}))$ and the absolute deadline is $t' + \widetilde{T}(\omega(t'), \min(\Omega_1(\omega(t'), \alpha_{\max}), \omega_{max}))$. That is, any job must be completed before the first possible release of the next job. AVR tasks that set deadlines this way are referred to as **minimum angular deadline AVR task** [15]. Assuming that consecutive jobs are released at times $\{t_1, t_2, \ldots\}$, Equation 3.7 can be modified for AVR tasks as,

$$D_{\omega(t)}(t_a, t_b) = \sum_{i:(t_i \geq t_a) \wedge (t_i + \tilde{d}(\omega(t_i)) \leq t_b)} c(\omega(t_i)). \tag{3.9}$$

The upper envelope on the demand over any interval of length $\delta > 0$ for $\omega(t)$ is given by,

$$\mathsf{dbf}_{\omega(t)}(\delta) = \max_{t' \geq 0} \{D_{\omega(t)}(t', t' + \delta)\}. \tag{3.10}$$

The worst-case demand of an AVR task over any δ-length interval can be obtained by choosing the speed schedule that maximizes the upper envelope in Equation 3.10:

$$\mathsf{dbf}(\delta) = \max_{\omega(t) \in \mathbb{W}} \{\mathsf{dbf}_{\omega(t)}(\delta)\}. \tag{3.11}$$

If any speed function $\omega(t)$ with job releases at speeds/times $\omega(t_1), \omega(t_2), \ldots$ is modified so that the crankshaft completes a rotation in the minimum time given by Equation 3.6, the resulting demand will be at least as much as the original. So, the problem of finding the function $\omega(t)$ that gives the worst-case demand can be reduced to identifying the *sequence of job release speeds* that gives the worst-case demand.

The objective of this thesis can be reiterated as finding the sequence of job release speeds that give the worst-case AVR task demand (dbf) defined in Equation 3.11.

Definition 2 (Valid Sequence [1]). *A sequence of jobs released at speeds $\omega_1, \omega_2, \ldots, \omega_k$ is said to be valid if $\forall i \in \{1, 2, \ldots..k\}$, ω_i is reachable from ω_{i-1}.*

3.6 Problem Definition

Given a minimum angular deadline AVR task triggered by a crankshaft with feasbile speed $[\omega_{\min}, \omega_{\max}]$ and acceleration $[\alpha_{min}, \alpha_{max}]$ ranges, such that $\alpha_{max} = |\alpha_{min}|$ and is characterized by a set of modes bounded by $\mathbf{\Omega_{rb}} = \{\omega_{rb_1}, \ldots, \omega_{\max}\}$ and their corresponding execution times $c(\omega_{rb_i})$, $i = 1, \ldots, m$, the objective of this thesis is to present efficient methods to calculate its worst-case demand ($\mathsf{dbf}(\delta)$ (Equation 3.11)) while assuming that the acceleration of the crankshaft can change within a single rotation.

Chapter 4

Knapsack-based approach for deriving the worst-case demand

In this chapter, we describe the problem transformation from calculating $\mathsf{dbf}(\delta)$ to a variant of the knapsack problem. This chapter both provides context for and relies upon several properties and lemmas defined in the next chapter (Chapter 5). Briefly, *dominant speed sequences* are sequences of job release speeds whose demand is equivalent to the maximum demand of a task over a given interval length (i.e., the demand of a dominant speed sequence coincides with the value of Equation 3.11. These dominant speed sequences are non-decreasing (see Lemma 1, Chapter 5) and start at right boundary speeds (see Lemma 2, Chapter 5).

We start off with the traditional knapsack problem. Consider a set of items each having a weight (w_i) and a profit (p_i) associated with it. Further, assume that there is a container (or a knapsack) of a fixed size W_c. The aim is to fill the knapsack with the items such that the profit is maximized. As the size of the knapsack is finite, the number of items that can fit is limited and it is clear that to obtain the maximum profit inclusion of an item in the knapsack might lead to the removal of another. So, an optimal allocation of items is required to maximize the profit. As such, it is a combinatorial optimization problem. Mathematically it can be expressed as,

$$maximize \sum_i p_i \tag{4.1}$$

$$\text{s.}t. \sum_i w_i \leq W_c \tag{4.2}$$

In a knapsack problem, called the bounded knapsack problem, it is possible to have multiple copies of an item i. Assuming that item i has b_i identical copies of itself, the optimization problem can be restated as,

$$maximize \sum_i b_i \cdot p_i \tag{4.3}$$

$$\text{s.}t. \sum_i b_i \cdot w_i \leq W_c \tag{4.4}$$

In yet another variant of the knapsack problem, an item might require the inclusion of another item in the knapsack. That is, for an item to be inserted in the knapsack there can be another item or a series of other items that need to be put in the knapsack. This variant of the knapsack problem is called the precedence constraint knapsack problem.

We now transform the problem of finding $\mathsf{dbf}(\delta)$ into a variant of the knapsack problem that is a combination of the bounded and precedence constraint knapsack problems. We consider a job as an item. Since we assume that a single job is released in a rotation, an item can also be considered to be equivalent to a rotation. The minimum inter-arrival time of a job is the weight of the item and the job's execution time is the profit. Further, the length of the time-interval is considered as the size of the knapsack. So, the goal is to maximize the demand (profit) over a time-interval (δ).

Since each speed in a dominant sequence should be reachable from its preceding speed, the number of jobs that can follow a given job is a finite set. So, our problem is a bounded precedence constraint knapsack problem (BPCKP) [20, 21]. Our problem is represented by

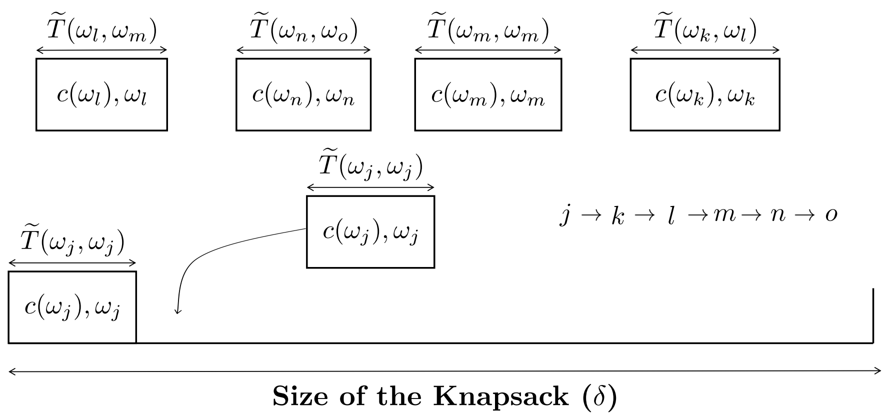

$\widetilde{T}(\omega_l, \omega_m)$
$c(\omega_l), \omega_l$
$\widetilde{T}(\omega_n, \omega_o)$
$c(\omega_n), \omega_n$
$\widetilde{T}(\omega_m, \omega_m)$
$c(\omega_m), \omega_m$
$\widetilde{T}(\omega_k, \omega_l)$
$c(\omega_k), \omega_k$
$\widetilde{T}(\omega_j, \omega_j)$
$c(\omega_j), \omega_j$
$j \to k \to l \to m \to n \to o$
$\widetilde{T}(\omega_j, \omega_j)$
$c(\omega_j), \omega_j$
Size of the Knapsack (δ)

Figure 4.1: Example items for our knapsack problem. A job that is higher in the precedence relation is preceded by a job lower in the relation.

a visual representation in Fig. 4.1. Precedence constraints are represented by the use of subscripts.

An effective way to model the precedence constraints among jobs is by using out-trees. The sequences beginning at each of the root nodes are independent of each other (i.e., sequences beginning at each of the right boundary speeds according to Lemma 2). An example is shown in Fig. 4.2. Each vertex in the tree ($G_I = (V_I, A_I)$) corresponds to a job in a dominant sequence. An edge (represented by $(j, k) \in A_I$) is drawn from a job j to the next job k in a dominant sequence and its value denotes the inter-arrival time. A path from the root to the

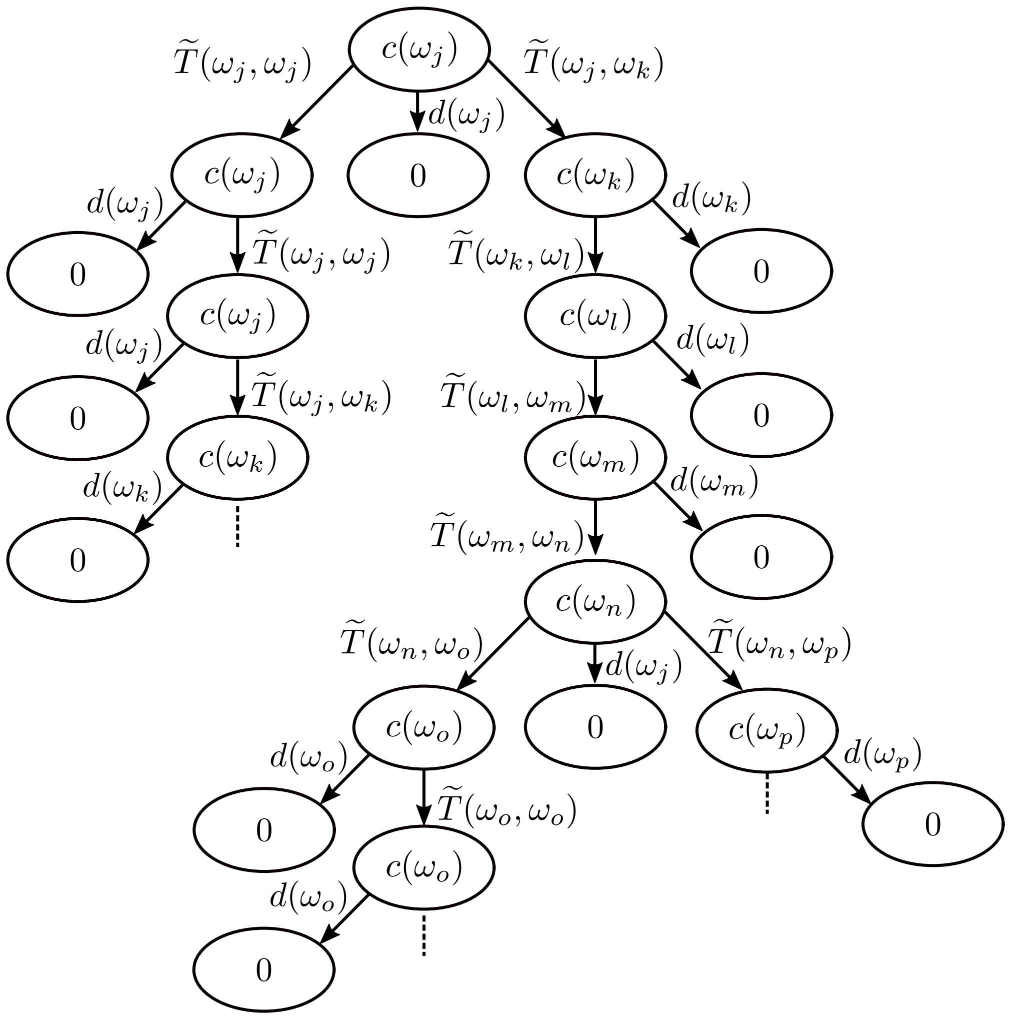

$c(\omega_j)$
$\widetilde{T}(\omega_j,\omega_j)$
$d(\omega_j)$
$\widetilde{T}(\omega_j,\omega_k)$
$c(\omega_k)$
$d(\omega_k)$
$\widetilde{T}(\omega_k,\omega_l)$
$c(\omega_l)$
$d(\omega_l)$
$\widetilde{T}(\omega_l,\omega_m)$
$c(\omega_m)$
$d(\omega_m)$
$\widetilde{T}(\omega_m,\omega_n)$
$c(\omega_n)$
$\widetilde{T}(\omega_n,\omega_o)$
$\widetilde{T}(\omega_n,\omega_p)$
$c(\omega_o)$
$d(\omega_o)$
$\widetilde{T}(\omega_o,\omega_o)$
$c(\omega_p)$
$d(\omega_p)$
0

Figure 4.2: Precedence constraints among jobs are expressed using out-trees. Nodes represent WCETs for the initial speeds and arrows represent minimum inter-arrival times. The tree demonstrates the various precedence relations and possible paths. Leaves represent completion of the parent job without adding subsequent jobs (i.e. items). Furthermore, each of the leaves has zero execution since its parent is the final job included in the knapsack.

leaf constitutes a dominant sequence. Our BPCKP can be formally defined as follows:

$$\underset{x}{\text{maximize}} \quad \sum_{j} \sum_{r=1}^{M_\delta} c(\omega_j) \cdot x_j^r \tag{4.5}$$

$$\text{subject to} \quad \sum_{j,k|(j,k)\in A_I} \sum_{r=1}^{M_\delta} \tilde{T}(w_j, w_k) \cdot x_j^r \leq \delta \tag{4.6}$$

$$\sum_{k|(j,k)\in A_I} x_k^1 \leq 1,\ \forall j \tag{4.7}$$

$$x_j^1 \geq x_k^1,\ \forall (j,k) \in A_I \tag{4.8}$$

$$x_j^r \geq x_j^{r+1},\ \forall j, r \in \{1,2,3,...,M_\delta - 1\} \tag{4.9}$$

$$x_j^r \in \{0,1\},\ \forall j, r \in \{1,2,3,...,M_\delta\} \tag{4.10}$$

where M_δ represents an upper bound on the number of jobs released in a time-interval of length δ and the binary variable x_j^r is 1 if at least r jobs are included at speed ω_j in the knapsack; else, x_j^r is 0 (Equations 4.9 and 4.10). Equation 4.5 gives the worst-case AVR task demand defined in Equation 3.11. Equation 4.6 restricts the selection of only those jobs whose deadlines fall within the δ-length interval. Equation 4.7 ensures that only a single child of a parent node is added to the knapsack i.e. it ensures that a single path from the root node to the tail is selected. Equation 4.8 enforces the precedence constraint that a child node be added only if its parent is present.

Algorithm 1 shows our pseudocode for the dynamic programming approach to solve the bounded precedence constraint knapsack problem. The *CalculateDemand* function is initially called with the parameters $\omega \in \mathbf{\Omega_{rb}}$ and δ, the total time length. The *maxDemand* parameter keeps track of the highest demand computed until the current instance of the recursive loop. In each recursive instance, the next speed is chosen from the list of possible next job release speeds according to Theorem 1 (Chapter 5). The variable D_w (Equation 3.9) tracks the accumulated demand of the current sequence.

Algorithm 1 DP for Calculating $\mathsf{dbf}(\delta)$

```
function CalculateDemand($\omega$,$\delta$):
    maxDemand $\leftarrow 0$

    if StoredDemand$(\omega, t) \neq \phi$ then
        return StoredDemand($\omega$,$\delta$)

    if $\delta < \tilde{d}(\omega)$ then
        return 0

    for $\omega'$ in nextPossibleSpeed$(\omega)$ do                ▷ See Theorem 1
        $\delta \leftarrow \delta - \widetilde{T}(\omega, \omega')$
        $D_w \leftarrow c(\omega)+$ CalculateDemand($\omega'$,$\delta$)

        if $D_w >$ maxDemand then
            maxDemand $\leftarrow D_w$

    StoredDemand($\omega$,$\delta$) $\leftarrow$ maxDemand
    return maxDemand
```

Chapter 5

Dominant Speed Sequence

The previous section outlined how the problem of calculating the $\mathsf{dbf}(\delta)$ can be transformed to BPCKP using dominant speed sequences. This section defines them and presents the Lemmas 1,2 and 3 used in the transformation to BPCKP.

Variation of execution time with angular speed is one of the main challenges in determining the worst-case demand of an AVR task. It was shown in previous work, e.g., Mohaqeqi et al. [4] that speed sequences from a finite set of speeds are enough to maximize the demand. These methods however, still consider a large number of speed permutations. We propose a method that greatly reduces the number of permutations.

In this chapter, we restate some of the lemmas from [1]. To describe them, we present some definitions. A sequence whose demand is equal to the maximum demand of an AVR task over a given interval is defined as a *dominant speed sequence.* A set of speed sequences that contain a *dominant speed sequence* is defined as a *dominant sequence set.*

Lemma 1 (Dominant Non-Decreasing Speed Sequences [1])**.** *Given a valid, dominant speed sequence S over interval $[t_a, t_b]$, the sequence S_A, obtained from reordering the speeds of S in a non-decreasing order, is valid and has equivalent demand.*

Proof. Refer to the Appendix for intuition. Please refer to [1] for a formal proof. □

Lemma 2 (Starting Speeds of a Dominant Sequence). *For any non-decreasing dominant sequence S_{orig} the sequence obtained by replacing the first k jobs in the i^{th} mode with jobs released at the right boundary speed of mode i, i.e., ω_{rb_i}, does not decrease the demand of the sequence.*

Proof. Please refer to the Appendix for intuition. Refer to [1] for a formal proof. □

It is worth noting that the above lemma eliminates all the speeds from the first step except the first right boundary speed from being a part of the dominant speed sequence.

Lemma 3 (Speeds Between Right Boundary Speeds in a Dominant Sequence). *Consider a non-decreasing dominant speed sequence, S, such that job, J_a, released at ω_a is the last job in the $i-1^{\text{th}}$ mode ($i > 1$) and job, J_c, is the first job in the $i+1^{\text{th}}$ mode. The jobs in the i^{th} mode, J_b ($a < b < c$), should be released at speeds, ω_b, such that $\forall\, b \in (a, c)$,*

$$\omega_b = \min(\omega_{rb_i}, \Omega_1(\omega_{b-1}, \alpha_{\max})) \tag{5.1}$$

Proof. Please refer to the Appendix for an intuition. Refer to [1] for a formal proof. □

Lemma 1 eliminates all the decreasing speed sequences. Lemma 2 states that a dominant speed sequence starts at a right boundary speed. Lemma 3 gives the speeds that a dominant speed sequence can contain. Now we show what the actual dominant sequence set is.

Let Ψ be a non-decreasing ordered set of speeds called the *dominant speed set* formally defined as follows:

$$\Psi = \left\{\Omega_n(\omega_{rb_i}, \alpha_{\max}) | (n \in \mathbb{Z}^+) \wedge (\omega_{rb_i} \in \mathbf{\Omega_{rb}})\right\}. \tag{5.2}$$

where, $\mathbb{Z}^+$ is the set of positive integers including 0.

Let $\mathsf{nextPossibleSpeed}(\omega_k)$ be a function that returns the set of speeds that can be chosen for the next job given the current speed, $\omega_k (\in \Psi)$. Let $\mathsf{nextPossibleSpeed}(\omega_0)$ denote that we are choosing the first speed in the dominant sequence. Since Lemma 2 states that a dominant sequence starts from a right boundary speed, $\mathsf{nextPossibleSpeed}(\omega_0)$ returns the right boundary speeds. For any ω_k, $k > 0$, according to Equation 5.1, $\mathsf{nextPossibleSpeed}(\omega_k)$ indicates that the crankshaft should maximally accelerate to the next speed. In addition, if ω_k is a right boundary, it should also return ω_k and the next reachable speed, ω_l ($< \Omega(\omega_k, \alpha_{max})$), if $\omega_l \in \Psi$. Formally,

$$\mathsf{nextPossibleSpeed}(\omega_k) = \begin{cases} \mathbf{\Omega_{rb}} & \text{if } k = 0 \\ \{\Omega_1(\omega, \alpha_{\max})\} \cup \{\omega_{rb_i} \in \mathbf{\Omega_{rb}}(\omega_k)\} & \text{if } k > 0 \end{cases} \tag{5.3}$$

where, $\mathbf{\Omega}_{rb}(\omega_k)$ denotes the set of reachable right boundary speeds from ω_k.

Let $\mathbb{S}(\delta)$ be a set of speed sequences defined as follows:

$$\left\{ \begin{array}{l} (\omega_1, \ldots, \omega_{|S|}) \in 2^{\Psi} \mid \\ \quad \left(\forall k \in \mathbb{N}_0^{|S|-1}, \omega_{k+1} \in \mathsf{nextPossibleSpeed}(\omega_k)\right) \\ \quad \wedge \left(\sum_{\ell=1}^{|S|-1} \widetilde{T}(\omega_\ell, \omega_{\ell+1}) + \Omega_1(\omega_{|S|}, \alpha_{\max}) \le \delta\right) \end{array} \right\} \tag{5.4}$$

Theorem 1. *The set $\mathbb{S}(\delta)$ must contain a dominant speed sequence for any interval of length $\delta > 0$.*

Proof. The correctness of the theorem lies in proving that $\mathsf{nextPossibleSpeed}(\omega_k)$ always returns a dominant sequence. By Lemma 1 only non-decreasing sequences are considered.

In Equation 5.3, $k \ge 0$. According to Lemma 2, the first speed of a dominant sequence must be a right boundary speed, this proves Equation 5.3 when $k = 0$. We need to prove it when $k > 0$.

When $k > 0$, there are several possibilities for the k^{th} speed:

Case 1 (k^{th} job is the first job in mode $i > 1$): In this case, $(k-1)^{\text{th}}$ speed may or may not be a right boundary speed. Applying Equation 5.1, k^{th} speed will be replaced by $\min(\Omega_1(\omega_{k-1}, \alpha_{\max}), \omega_{rb_i})$, proving Equation 5.3 for Case 1.

Case 2 (k^{th} speed is the right boundary speed of mode i): Equation 5.1, when applied for ω_k will guide us to replace ω_k by ω_k itself because we assumed $\omega_k = \omega_{rb_i}$, proving Case 2.

Case 3 (k^{th} speed is an intermediate speed in the middle of mode i): This case follows on the application of Equation 5.1.

In addition, only jobs that are released and whose deadlines fall within an interval of length δ can be part of the dominant sequence. This can be verified by examining the definition of the demand bound function $\mathsf{dbf}(\delta)$ in Equation 3.11.

Together, Lemmas 1, 2, and 3 allow for elimination of unnecessary sequences from the dynamic programming search while Theorem 1 ensures the sequences produced by Algorithm 1 are dominant speed sequences whose demand coincide with Equation 3.11. $\square$

Chapter 6

Approximation Algorithms

The current AVR task model, which has only two dimensions, can be extended to multiple dimensions by considering the effect of physical properties like temperature and pressure on the speed and execution time. Another direction of extending the work is to relax the assumption of symmetric accelerations ($\alpha_{max} == |\alpha_{min}|$) and achieving an improvement in runtime over the state-of-the-art, the DRT approach. Each modification to the current problem increases its complexity. To handle this complexity and obtain a pessimistic solution in a shorter time (milliseconds or less), we present an approximation algorithm. It can be used as a first line of check before using the exact algorithms.

The traditional knapsack problem is a well studied problem and several approximation algorithms have been proposed over the years to calculate sub-optimal solutions. Since our problem is a variant of the traditional knapsack problem, some of the approximation algorithms applicable to the traditional knapsack can be applied to our BPCKP. In this chapter, we will describe Linear Programming Relaxation approximation of our BPCKP.

First, we'll describe approximation algorithms in the context of the traditional knapsack problem and then extend it to our BPCKP. The approximation algorithms for the traditional knapsack problem are present in the book [22]. Here we briefly describe them for readability.

Consider a set of items, $b_j \in \mathbb{B}$. Assume that an item b_j has a profit p_j and a weight w_j. Efficiency of an item is defined as the amount of profit per unit weight of the item, i.e. $\frac{p_j}{w_j}$. For ease of explanation, we assume that the items $\{b_1, b_2, \ldots, b_n\}$ are arranged in the decreasing order of efficiency for the rest of the chapter. Now, we will describe one of the basic approximation algorithms for a Knapsack, called the Greedy-1 algorithm.

Greedy-1:

Add the items to the knapsack in the decreasing order of efficiency. When an item that does not fit in the Knapsack is encountered, move on to the next item until all the items are considered. Let us call this solution G_1.

There is a variant of this algorithm called the Greedy-2 approximation algorithm.

Greedy-2:

Greedy-2 is mostly the same as Greedy-1, except that the process of adding items into the knapsack is stopped the first time an item that cannot fit in the knapsack is encountered. Let the solution be represented by G_2. Let us assume that the first $s-1$ elements can fit in the knapsack. So, the solution can be represented as,

$$G_2 = \sum_{j=1}^{s-1} p_j \tag{6.1}$$

Now, we describe the linear programming approximation of the knapsack problem which is derived from Greedy-2 algorithm.

Linear Programming Relaxation (LP-relaxation):

In LP-relaxation, the procedure is same as Greedy-2 in that the items are added in the decreasing order of the efficiency but when the first item that does not fit in the knapsack is encountered, the fraction of the item that fits in the knapsack is added, i.e., a fraction of the s^{th} element is added to the knapsack. So, the solution of LP-approximnation, z^{LP} can be represented as,

$$z^{LP} = \sum_{j=1}^{s-1} p_j + \left(c - \sum_{j=1}^{s-1} w_j \right) \cdot \frac{p_s}{w_s} \tag{6.2}$$

where, c is the capacity of the knapsack.

It can be shown that z^{LP} gives a 2-factor approximation to the traditional knapsack problem, i.e., $2z^* \geq z^{LP}$, where z^* is the exact solution of the traditional knapsack problem.

6.1 LP-relaxation for BPCKP

Now, we describe the application of LP-relaxation on our BPCKP. The first step is to arrange the items in the increasing order of efficiency. Since, we know that in each mode, the speed of the crankshaft is the highest at the right boundary speeds, the inter-arrival time, which is the weight of the item, is the least for the right boundary speeds. So, in a mode, the efficiency is the highest for the items released at the right boundary speeds. Since multiple jobs can be released at the right boundary speeds, intuitively, the LP-relaxation of our BPCKP consists of only the items released at the right boundary speed which has the highest efficiency.

From Lemma 3, there are two choices of next jobs from a right boundary speed - release the next job at the right boundary speed (say item-1) or use maximum acceleration through the rotation and release the next job in the next mode (say item-2). Among the two item types possible, item-2 has a higher efficiency because, it has a smaller inter-arrival time. So it has to be the first item in the knapsack. If precedence constraints are to be followed, items of the next mode, which have a smaller efficiency than item-1 have to be filled in the knapsack. However, the LP-relaxation is only a bound, precedence constraints need not be followed. So, after item-2 is added to the knapsack, the knapsack is filled with copies of item-1 and the last item when the capacity exceeds is replaced by a fraction of item-1.

Chapter 7

Evaluation

In this section, we compare our algorithm with the state-of-the-art for solving the problem at hand, the DRT algorithm proposed by Mohaqeqi et al. [4]. As explained in the related work chapter, Mohaqeqi et al. partition each of the modes into several sub-ranges each defined as a node in the Digraph and employ a search for the sequence with the highest demand starting at each of these nodes. Our approach explores a subset of speeds considered by Mohaqeqi et al. [4], and so we inherit the same upper bound on number of speeds: $O\left(m \cdot \frac{\omega_{max}^2 - \omega_{min}^2}{2 \cdot \alpha_{max}}\right)$. Note that computing these speeds takes a negligible amount of time (less than 1%) when compared to exploring through all the permutation of these speeds. Even though our algorithm shares similar time complexity with the DRT algorithm with respect to computing these speeds, our algorithm significantly reduces the search space resulting in a much faster computation of the worst-case demand. Since the final search space in our algorithm is a BPCKP, the search complexity is the same as any bounded precedence constraint knapsack problem. We compare the accuracy and runtime of the algorithms using two experiments. For all experiments, a maximum acceleration of 600,000 rev/min^2 and maximum deceleration of -600,000 rev/min^2 were assumed as in previous work [4,6,7,12]. In each experiment, the worst-case demand is calculated for 100 time intervals in $[0, 1s]$ in steps of 10 ms. The experiments were performed using Python 3.6.5 on a 3.40 GHz, quad-core

Table 7.1: Task set used by existing work [3, 4]

i^{th} mode	1	2	3	4	5	6	ω_{rb_m}
ω_{rb_i}	500	1500	2500	3500	4500	5500	6500
$c(\omega_{rb_i})$	965	576	424	343	277	246	

Table 7.2: A more general task set.

i^{th} mode	1	2	3	4	5	6	ω_{rb_m}
ω_{rb_i}	1200	2200	3200	4200	5200	6200	7200
$c(\omega_{rb_i})$	965	576	424	343	277	246	

processor with 8 GB of RAM. While the results are platform dependent, the general trend showing the relative performance of the proposed approach against the DRT algorithm should be representative. Each experiment is run 10 times and the average values are reported. The code and the original data used for this publication can be found on the artifact evaluation page [23].

In the first experiment, two task sets were used. The first task set appeared in existing publications [3, 4] (Table 7.1). Note that this task set is not ideal, as some boundary speeds can be reached from the others in an integer number of rotations using maximum acceleration, which reduces the number of speed partitions simplifying the search for the worst-case demand. The results are shown in Table 7.3(a). Though both approaches were able to find the worst-case demand, our algorithm is 13.5 times faster.

Since the first task set is not ideal, as described earlier, we created another task set (Table 7.2) to better illustrate the improvement achieved by our algorithm. This task set is more general in the sense that the right boundary speeds are not reachable from one another in integer number of rotations when using $\alpha_{\max}$. Since the number of dominant speeds for this taskset are higher, we expect that the algorithms will require longer running times to find

Table 7.3: Runtime comparison of different algorithms

	Demand (in μs) over [0,1s]	Runtime
DRT Alg.	26,568	3 min. 31 sec.
Our Alg.	26,568	15.63 sec.

(a) An existing task set

	Demand (in μs) over [0,1s]	Runtime
DRT Alg.	35,892	17 min. 2 sec.
Our Alg.	35,892	19.24 sec.

(b) A more general task set

the worst-case demand of this task set. The results are shown in Table 7.3(b). As before both approaches were able to find the worst-case demand but our algorithm is 53.1 times faster.

For the second experiment, we generated multiple random AVR tasks using an algorithm presented by Biondi et al. [6]. The execution times and modes of these tasks are combined and they are modeled as a single task, called the representative AVR task. The modes of the representative AVR task are generated by assuming a fixed value of 0.25 for the maximum utilization factor of the modes. Again, the same worst-case demands were found by both algorithm, but overall, our approach is 146 times faster on average and up to 250 times faster, as shown in Fig. 7.1.

Now, the above two experiments are repeated to compare our algorithm and the LP-relaxation of BPCKP. Since the LP-relaxation typically finishes in less than a few milli seconds, only the percentage error plots (Fig. 7.2) are generated. In the figures, the error is large for smaller time intervals and it decreases as the length of the time intervals increases. This is because for smaller time intervals, when only a few job releases are possible, a fraction of an item can generate a huge error in the output.

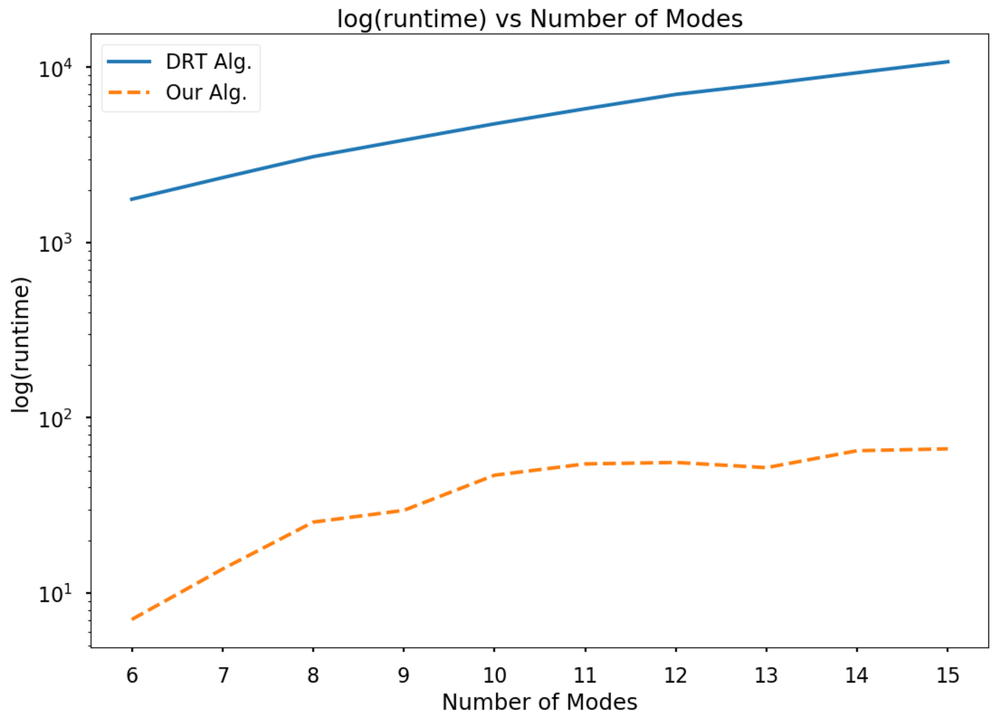
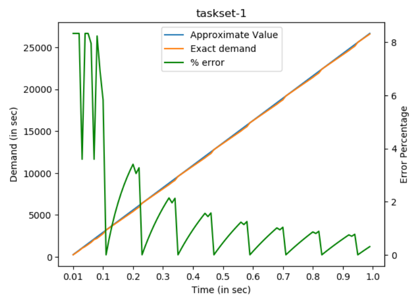
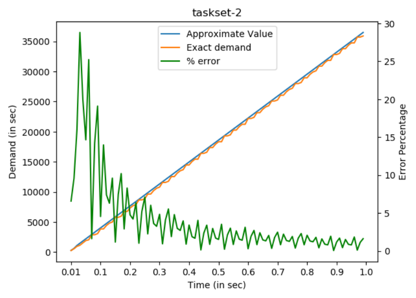


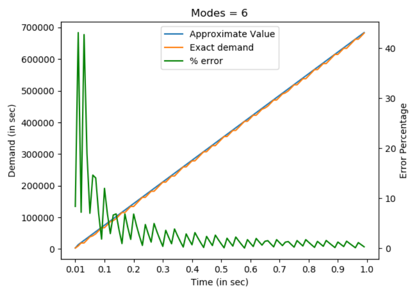
Figure 7.1: Graph depicting the log of runtime of different algorithms as a function of the number of modes of randomly generated AVR task sets.



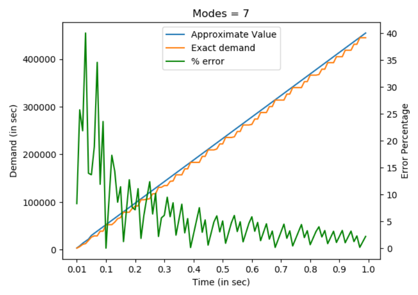
(a)



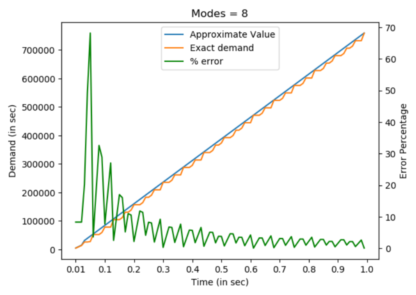
(b)



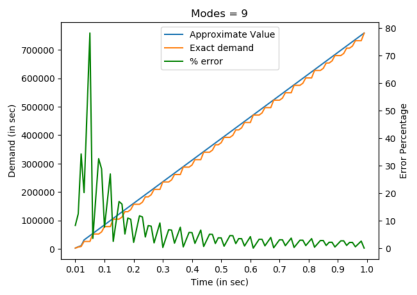
(c)



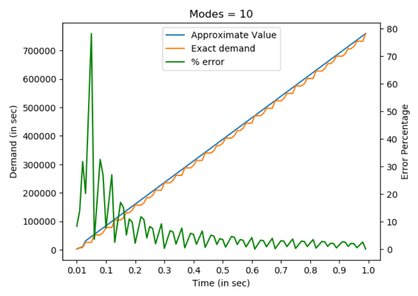
(d)



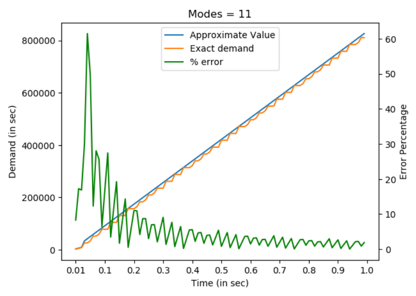
(e)



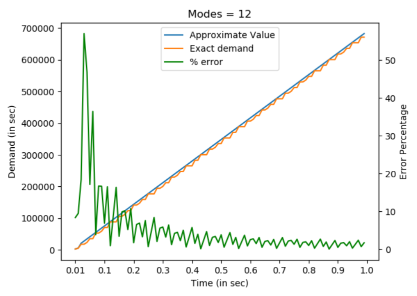
(f)



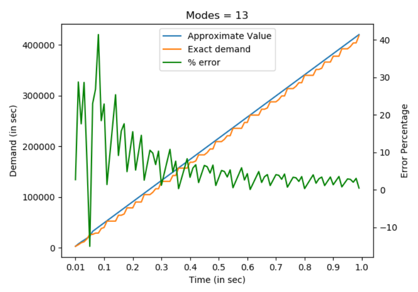
(g)



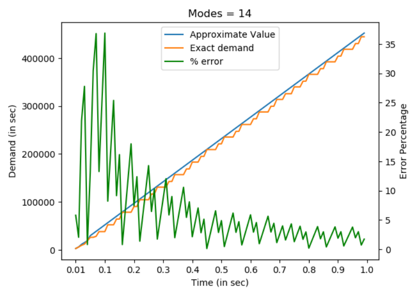
(h)



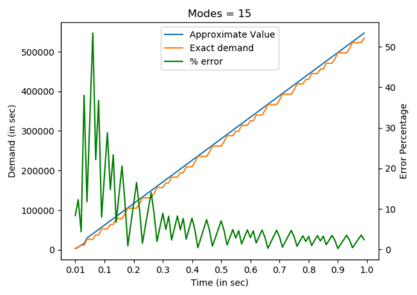
(i)



(j)



(k)



(l)

Figure 7.2: Demand and Percentage error plots of (a) taskset-1, (b) taskset-2 and (c-l) Random tasksets

Chapter 8

Conclusions and Future Work

This thesis presented an efficient method for calculating the exact worst-case demand and also its approximate solution. First, a knapsack-based dynamic programming approach was proposed to efficiently find the worst-case demand given this smaller dominant sequence set. Second, a collection of necessary conditions were presented to reduce the search space and find the dominant sequence set. Then, an approximation algorithm was presented to compute an approximate solution. Experimental results confirm that the proposed approach is much faster than the state-of-the-art technique.

This work can be extended by relaxing the assumption of $\alpha_{max} == |\alpha_{min}|$. More dimensions like pressure and temperature can also be added to the AVR model. Another direction of extension can be studying AVR tasks that have different phases and which are released by independent sources.

Bibliography

[1] S. K. Bijinemula, A. Willcock, T. Chantem, and N. Fisher, "An efficient knapsack-based approach for calculating the worst-case demand of AVR tasks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2018.

[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973. [Online]. Available: http://doi.acm.org/10.1145/321738.321743

[3] A. Biondi, A. Melani, M. Marinoni, M. D. Natale, and G. Buttazzo, "Exact interference of adaptive variable-rate tasks under fixed-priority scheduling," in *2014 26th Euromicro Conference on Real-Time Systems*, July 2014, pp. 165–174.

[4] M. Mohaqeqi, J. Abdullah, P. Ekberg, and W. Yi, "Refinement of Workload Models for Engine Controllers by State Space Partitioning," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Bertogna, Ed., vol. 76. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 11:1–11:22. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7159

[5] D.Buttle, "Real-time in prime-time," keynote speech at the 24th Euromicro Conference on Real-Time Systems, Pisa, Italy, July 12, 2012.

[6] A. Biondi, M. Di Natale, and G. Buttazzo, "Response-time analysis for real-time tasks in engine control applications," in *Proceedings of the ACM/IEEE Sixth International*

Conference on Cyber-Physical Systems, ser. ICCPS '15. New York, NY, USA: ACM, 2015, pp. 120–129. [Online]. Available: http://doi.acm.org/10.1145/2735960.2735963

[7] A. Biondi, G. Buttazzo, and S. Simoncelli, "Feasibility analysis of engine control tasks under edf scheduling," in *2015 27th Euromicro Conference on Real-Time Systems*, July 2015, pp. 139–148.

[8] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2011, pp. 71–80.

[9] J. Kim, K. Lakshmanan, and R. R. Rajkumar, "Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems," in *Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, ser. ICCPS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 55–64. [Online]. Available: http://dx.doi.org/10.1109/ICCPS.2012.14

[10] V. Pollex, T. Feld, F. Slomka, U. Margull, R. Mader and G. Wirrer, "Sufficient real-time analysis for an engine control unit with constant angular velocities," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 1335–1338. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485607

[11] V. Pollex, T. Feld, F. Slomka, U. Margull, R. Mader, and G. Wirrer, "Sufficient real-time analysis for an engine control unit," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, ser. RTNS '13. New York, NY, USA: ACM, 2013, pp. 247–254. [Online]. Available: http://doi.acm.org/10.1145/2516821.2516838

[12] R. I. Davis, T. Feld, V. Pollex, and F. Slomka, "Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2014, pp. 51–62.

[13] Z. Guo and S. Baruah, *Uniprocessor EDF scheduling of AVR task systems.* Association for Computing Machinery, Inc, 4 2015, pp. 159–168.

[14] T. Feld, A. Biondi, R. I. Davis, G. Buttazzo, and F. Slomka, "A survey of schedulability analysis techniques for rate-dependent tasks," *Journal of Systems and Software*, vol. 138, pp. 100 – 107, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217303102

[15] A. Biondi and G. Buttazzo, "Modeling and analysis of engine control tasks under dynamic priority scheduling," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2018.

[16] A. Biondi and G. Buttazzo, "Real-time analysis of engine control applications with speed estimation," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 193–198.

[17] A. Biondi, M. Di Natale, and G. Buttazzo, "Performance-driven design of engine control tasks," in *Proceedings of the 7th International Conference on Cyber-Physical Systems*, ser. ICCPS '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 45:1–45:10. [Online]. Available: http://dl.acm.org/citation.cfm?id=2984464.2984509

[18] H. C. Verma, *Concepts of Physics - Vol. 1.* Bharati Bhawan Publishers and Distributors, 2010.

[19] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Systems*, vol. 2, no. 4, pp. 301–324, Nov 1990. [Online]. Available: https://doi.org/10.1007/BF01995675

[20] G. Cho and D. X. Shaw, "A depth-first dynamic programming algorithm for the tree knapsack problem," *INFORMS Journal on Computing*, vol. 9, no. 4, pp. 431–438, 1997. [Online]. Available: https://doi.org/10.1287/ijoc.9.4.431

[21] D. S. Johnson and K. A. Niemi, “On knapsacks, partitions, and a new dynamic programming technique for trees,” *Mathematics of Operations Research*, vol. 8, no. 1, pp. 1–14, 1983. [Online]. Available: https://doi.org/10.1287/moor.8.1.1

[22] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems.* Springer Berlin Heidelberg, 01 2004.

[23] S. K. Bijinemula, A. Willcock, T. Chantem, and N. Fisher, “Code for the paper-efficient knapsack-based approach for calculating the worst-case demand of avr tasks,” 2018. [Online]. Available: https://github.com/bsk1410/Efficient-Knapsack-for-AVR-tasks-RTSS2018

Appendix A

Appendix

A.1 Table of Notation and Units

ω	Angular speed (rev/min)
ω_{rb}	Right boundary speed (rev/min)
$\mathbf{\Omega_{rb}}$	Set of right boundary speeds
$\omega_{\max}$	Maximum angular velocity (rev/min)
α	Angular acceleration (rev/min^2)
$\alpha_{\max}$	Maximum angular acceleration (rev/min^2)
$\alpha_{\min}$	Minimum angular acceleration (rev/min^2)
t	Time (sec)
$\omega(t)$	Instantaneous angular velocity (rev/min)
$c(\omega(t))$	Worst-case execution time (sec.)
θ	Angular position (rev)
$\Delta\theta$	Change in angular distance (rev)
Ω_n	Angular velocity at the end of ‘n’ rotations (rev/min)
$\widetilde{T}$	Minimum interarrival time (sec.)
ω_p	Peak angular velocity (rev/min)

f	Angular velocity at the end of a rotation (rev/min)
$d(\omega)$	Relative deadline (sec.)
D_ω	Demand (sec.)
dbf	Demand Bound Function (sec.)
δ	Time interval length (sec.)
G_I	Dependency graph
V_I	Vertices
A_I	Edges
$\mathbb{Z}^+$	Positive integers
S	Speed sequence
p_j	Profit of an item
w_j	Weight of an item
G_1	Solution of Greedy Approximation Algorithm - 1
G_2	Solution of Greedy Approximation Algorithm - 2

A.2 Filtering the Sequences

Here, we give some intuition to the lemmas that are presented in chapter A.2.4. We provide some properties that a dominant speed sequence has and eliminate all those sequences that do not follow these properties from the dominant sequence set.

Lemma 4 (Highest demand in a mode)**.** *For a sequence of jobs released in the* i^{th} *mode, the highest demand in a time interval* $[t_1, t_2]$*, is obtained when the jobs are released at* ω_{rb_i}*.*

Proof. Consider the speed vs. time graph in Fig. A.1. The horizontal line indicates the jobs released at ω_{rb_i} while the other lines represent the other possible sequences in a time interval $[t_1, t_2]$. As can be seen from the figure, the area under the curve is the highest if the jobs are released at ω_{rb_i}. As a single mode is considered, the execution time is the same. So, the highest demand in a mode i is obtained if the jobs are released at ω_{rb_i}. □

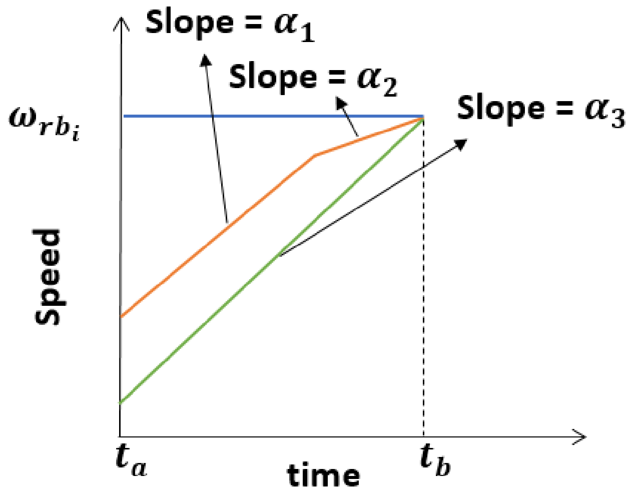


Figure A.1: In a mode, the area under the curve is maximum if jobs are released at the right boundary speed, ω_{rb_i}.

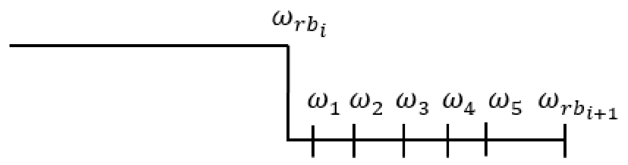

ω_{rb_i}
ω_1 ω_2 ω_3 ω_4 ω_5 $\omega_{rb_{i+1}}$

Figure A.2: Speeds obtained when maximum acceleration is used from the right boundary speed of a mode.

A.2.1 Finding the Dominant Sequences across Modes

Intuitively, from Lemma 4, it might seem logical to only consider the jobs released at the right boundary speeds in all the modes. However, this might not always give the worst-case demand when multiple modes are considered. For example, assume a sequence as shown in the Fig A.3a. ω_{rb_i} and $\omega_{rb_{i+1}}$ are the right boundary speeds of two consecutive modes (as shown in Fig. A.2).

ω_1, ω_2, ω_3, ω_4, ω_5 are speeds in the next step such that

$$\begin{aligned}\omega_1 = \Omega(\omega_{rb_i}, \alpha_{max}), \omega_2 = \Omega(\omega_1, \alpha_{max}),\\ \omega_3 = \Omega(\omega_2, \alpha_{max}), \omega_4 = \Omega(\omega_3, \alpha_{max}),\\ \omega_5 = \Omega(\omega_4, \alpha_{max})\end{aligned} \tag{A.1}$$

The demand in the case of A.3a is given by,

$$(D_\omega)_{A.3a} = 4 \cdot c(\omega_{rb_i}) \tag{A.2}$$

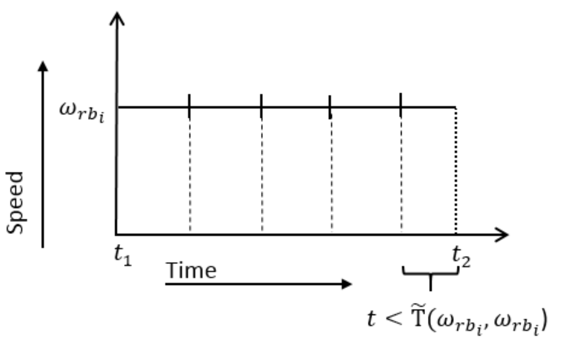
Assume that the time left after the last job is released in A.3a is just short of the relative deadline of that job. Now, if the last job in the sequence A.3a is replaced by a job that is released when the crankshaft rotates at the maximum acceleration (α_{max}) i.e. at ω_1, the time left might be just more than the inter-arrival time of the next job, ω_2 (as shown in the Fig. A.3b). In this case, the demand is given by,

$$\begin{aligned}(D_\omega)_{A.3b} = 3 \cdot c(\omega_{rb_i}) + c(\omega_1) + c(\omega_2)\\ c(\omega_1) = c(\omega_2)\\ (D_\omega)_{A.3b} = 3 \cdot c(\omega_{rb_i}) + 2 \cdot c(\omega_1)\end{aligned} \tag{A.3}$$

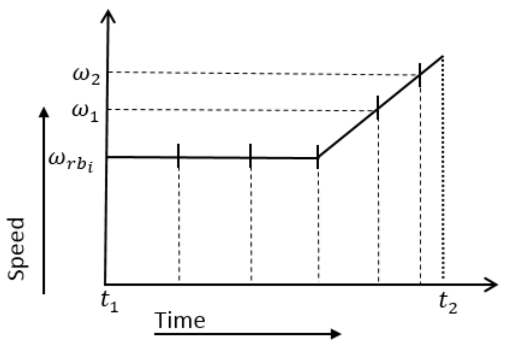
Depending on the values of $c(\omega_{rb_i})$ and $c(\omega_1)$, $(D_\omega)_{A.3b}$ can be greater than $(D_\omega)_{A.3a}$. There can be several other combinations of job releases spanning across multiple modes as shown in Fig. A.4. So we need to consider speeds other than the right boundary speeds in our analysis. Since there is no mode before the first mode, this reasoning does not apply to the speeds of the first mode. Next we show that only the right boundary speed of the first mode is to be considered to be a part of the dominant speed sequence.

Lemma 5. *In mode* 1*, only the jobs released at* ω_{rb_1} *can be a part of the dominant sequence.*

Proof. According to lemma 4, all the speeds of the first mode are dominated by its right boundary speed. Hence the lemma follows. □

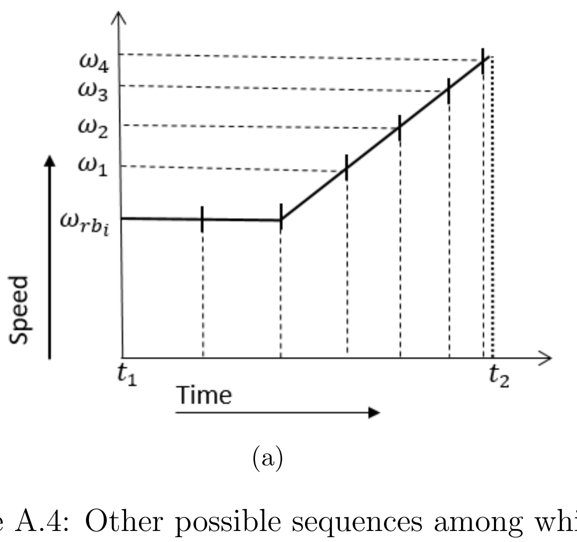


(a)

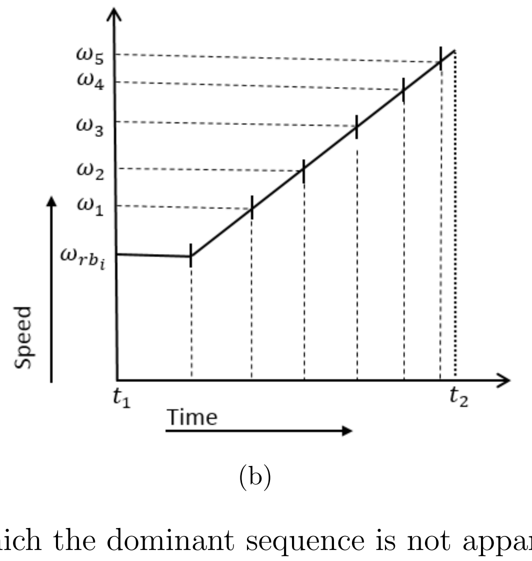


(b)

Figure A.3: (a) An example sequence in which the time left after the release of the last job is less than the deadline of the next job. (b) An example sequence in which the last job released at a right boundary speed is replaced by jobs released when maximum acceleration is used from the right boundary speed.



(a)



(b)

Figure A.4: Other possible sequences among which the dominant sequence is not apparent.

A.2.2 Non-decreasing Speed Sequences

Property 1. *For every non-increasing speed sequence, s_1, there exits a corresponding non-decreasing speed sequence, s_2, during the same duration, such that s_2 is a mirror image of s_1.*

Lemma 6. *Consider a non-increasing speed sequence speed, s_1, and a corresponding non-decreasing speed sequence, s_2, during the same duration, such that s_2 is a mirror image of s_1. The demand of s_2 is always greater than or equal to the demand of s_1.*

Proof. Since the jobs are released at the beginning of the rotation, the execution times of the jobs depends on the initial speed of rotation. As the execution time vs. speed graph is a decreasing step function, the jobs that are released during rotation from one mode to the other in s_2 are released at higher execution times than the corresponding jobs in s_1. Since the number of jobs is the same in both the cases, the demand of s_2 is no less than s_1. □

The previous lemma eliminates all the non-increasing speed sequences but sequences in which there is mixture of increasing and decreasing speeds are not eliminated. We need to consider speeds other than the right boundary speeds and from Lemma 4 we know that the highest demand in a mode is obtained if the jobs are released at the right boundary speed, so if the crankshaft is at any speed ω, we need to find the sequence of accelerations that increase the speed of the crankshaft to the right boundary speed of the mode in the shortest duration possible. However, depending on the choice of accelerations, the right boundary speed may or may not occur at at the end of a rotation. So, we divide the sequences into two types-those that reach the right boundary speed in an integer number of rotations and the others that reach the right boundary speed in the middle of a rotation as discussed next.

A.2.3 Fastest way to reach the right boundary speed

In this segment, we find the sequence of accelerations that increase the crankshaft's speed from ω_1 at time t_1 in the i^{th} mode to the right boundary speed of the mode, ω_{rb_i} in the shortest time. If there is no restriction of integer number of rotations, the minimum number of rotations taken to reach ω_{rb_i} from ω_1 is obtained when α_{max} is used from ω_1 to ω_{rb_i} which is given by,

$$n'_{min} = \frac{\omega_{rb_i}^2 - \omega_1^2}{2\pi\alpha_{max}} \tag{A.4}$$

Unless n'_{min} is an integer, ω_{rb_i} is obtained at some point in the middle of the $\lceil n'_{min} \rceil^{th}$ rotation, in which case, the crankshaft enters the next mode.

Finding the sequence of accelerations to reach the right boundary speed in integer number of rotations

In any mode i, we need to find the sequence of accelerations that increase the crankshaft's speed from ω_1 at time t_1 to ω_{rb_i} in integer number of rotations. Consider the example in Fig. A.5. s_1 is obtained when maximum acceleration is used from the speed ω_1 until ω_{rb_i} is reached at time t' and zero acceleration is applied from t' to t_2 to finish the current rotation. Ideally, if the number of rotations from ω_1 to ω_{rb_i} is an integer, the value of $t_2 - t'$ will be 0. s_2 is an arbitrary sequence during the same time interval $[t_1, t_2]$ such that it starts at ω_1 and uses accelerations α_1,α_2,α_3 during $[t_1, t_2]$.

Since, the shortest distance to travel to reach ω_{rb_i} is $2\pi n'_{min}$ from Equation A.4, the minimum number of integer rotations is the next highest integer, which is given by,

$$n_{min} = \lceil n'_{min} \rceil \tag{A.5}$$

This value of n_{min} signifies the minimum number of rotations required to traverse when

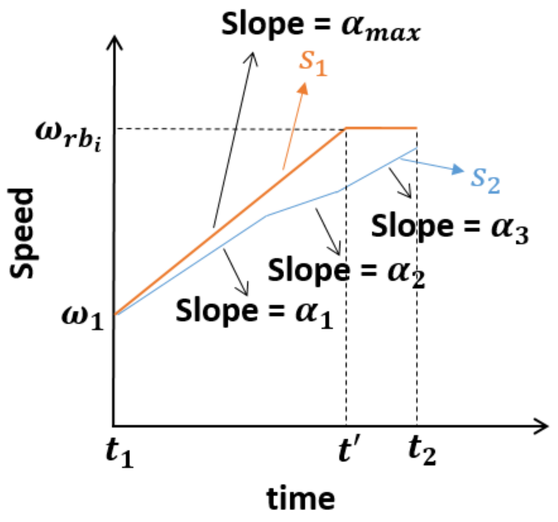


Figure A.5: s_1 takes the least amount of time to reach ω_{rb_i} from ω_1 among the sequences that do have a speed overshoot over ω_{rb_i} in the middle of a rotation.

accelerating from ω_1 to ω_{rb_i}. In other words, it is not possible to reach ω_{rb_i} from ω_1 without traveling a minimum angular distance of n_{min} rotations. From t_1 to t', the maximum area under the *speed* vs. *time* curve is obtained when maximum acceleration is used from ω_1. If the speed at any point in the middle of a rotation does not exceed the right boundary speed, the maximum area under the curve from t' to t_2 is given by the rectangle formed by the horizontal lines $speed = \omega_{rb_i}$ and the *time* axis and, the vertical lines $time = t'$ and $time = t_2$. So, s_1 has a higher area than any sequence that does not go over the right boundary speed. From Eqn. A.4, we know that the number of rotations completed by sequence s_1 from t_1 to t' is n'_{min}. If we assume that the number of rotations traversed from t' to t_2 is $n_{min} - n'_{min}$, the total distance which is also the area under the curve cannot be n_{min} for any other curve. Since, n_{min} is the minimum number of complete rotations required to be traversed to reach ω_{rb_i} from ω_1, any curve that does not go over the right boundary speed and that does not complete n_{min} rotations cannot take shorter amount of time than s_1 to accelerate from ω_1 to ω_{rb_i}.

The above discussion proves that s_1 in Fig. A.5 takes the shortest amount of time to

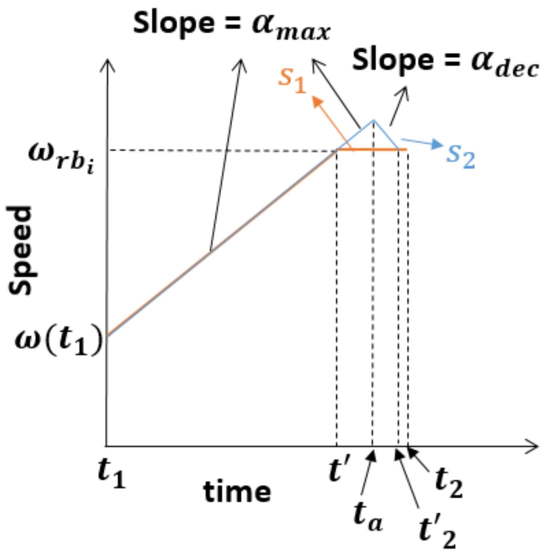


Figure A.6: s_2 takes the least time to accelerate from ω_1 to ω_{rb_i} in integer number of rotations.

accelerate from a speed ω_1 of mode i to its right boundary, ω_{rb_i}, among the sequences in which, the speed does not overshoot over ω_{rb_i} while ensuring that integer number of rotations are completed. Here, we used the variable acceleration in the last rotation. However, we know that to achieve the minimum time to complete a rotation we need to use $\widetilde{T}$ (from Equation 3.6). This is obtained when α_{max} is used for a part of the rotation and α_{min} is used for the rest of the rotation. This is shown in the example in Fig. A.6. s_1 is the same as in Fig. A.5 and s_2 is equivalent to s_1 till t' and the crankshaft continues with an acceleration of α_{max} from t' to t_a and then decelerates till it reaches ω_{rb_i}.

The above discussion shows that in a dominant sequence α_{max} is used when the crankshaft is at a non-boundary speed until the right boundary speed of the current mode is reached. Intuitively, this restricts that the next job in the sequence should not be released at speeds lower than the current speed.

A.2.4 Starting Speed of a Dominant Sequence

We now show that only those sequences that start at a right boundary speed can be a part of the dominant sequence set.

Definition 3 (Isolated sub-sequence)**.** *The sub-sequence from the release of the first job of the sequence to the instant where there is a mode change is called an Isolated sub-sequence.*

Lemma 7 (Starting Speeds of a Dominant Sequence)**.** *For any non-decreasing dominant sequence s_1 over the interval $[t_a, t_b]$ where the first k jobs (denoted $s_1 = (\omega_1, \omega_2, \ldots \omega_k)$) are released in the i^{th} mode, the sequence s_2 obtained by replacing this first k jobs with jobs released at the right boundary speed of mode i, i.e., ω_{rb_i}, does not decrease the demand of the sequence in $[t_a, t_b]$.*

Proof. Assume that s_1 and s_2 in Fig. A.7 are two isolated sub-sequences during a time interval $[t_1, t_2]$, such that the sub-sequences following s_1 and s_2 are the same. In the isolated sub-sequence of s_2, all the jobs are released at the right boundary speed of i^{th} mode whereas in the isolated sub-sequence s_1, accelerations $\alpha_1, \alpha_2, \ldots, \alpha_k$ are used to release jobs at $\omega_1, \omega_2, \ldots \omega_k$. The area under s_2 is greater than that in $s_1 \, \forall \, \alpha_1, \alpha_2, \ldots, \alpha_k$. Since the area under the curve is a measure of the distance traversed by the crankshaft and since a job is released for every 2π radian distance travelled by the crankshaft, the demand of s_2 is not less than that of s_1. □

Definition 4. *In a speed sequence any job released at speed $\omega(t_1)$ at time t_1 such that $\omega < \omega_{rb_i} < \Omega(\omega, \alpha_{max}), \forall i$ is said to be released at a special speed. It is represented by ω_{sp} (refer Fig. A.8). It is worth noting that the speed at the beginning of the last rotation in s_1 and s_2 in Figures A.5 and A.6 is a special Speed.*

The next job in a dominant sequence when the current speed is a special speed is either the right boundary speed of the current mode or accelerating maximally. Proved in Theorem 1.

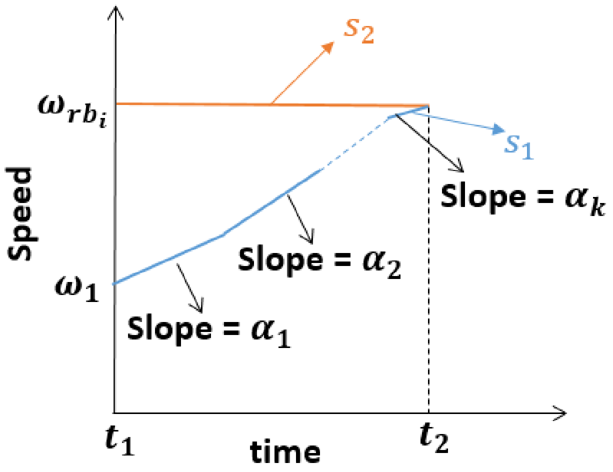


Figure A.7: The demand does not decrease if all the speeds in the isolated sub-sequence of the i^{th} mode are replaced by ω_{rb_i}

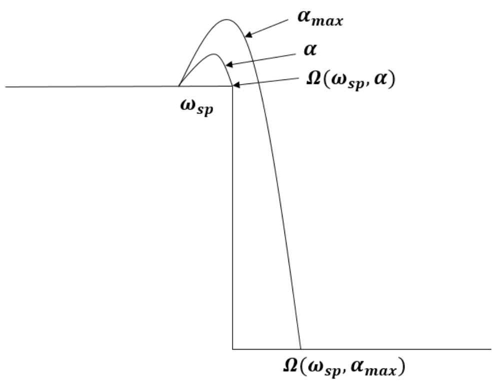

α_{max}
α
$\Omega(\omega_{sp}, \alpha)$
ω_{sp}
$\Omega(\omega_{sp}, \alpha_{max})$

Figure A.8: In a dominant sequence, the speed following a special speed (ω_{sp}) is either the right boundary speed (using $\alpha < \alpha_{max}$) of the current step or $\Omega(\omega_{sp}, \alpha_{max})$ (using α_{max}).

A.3 Summarizing the properties of dominant sequences

A dominant sequence has the following properties.

1. It starts at a right boundary speed.
2. Maximum acceleration is used at all the speeds except at special speeds and right boundary speeds.
3. At special speeds, two acceleration values are possible - maximum acceleration and a non-max acceleration. This is equivalent to a sequence splitting into two.
4. At right boundary speeds, two values of acceleration are possible - maximum acceleration and zero acceleration.
5. Multiple jobs are released only at right boundary speeds.

Fig. A.9 gives two examples of non-decreasing speed sequences that can potentially give the worst-case demand during a time interval [0,t]. In Fig. A.9(a), the sequence starts at ω_{rb_i} at $t = 0$ and α_{max} is used till the splitting speed, ω_{sp} and a non-maximum value of α_1 is used till the next right boundary speed, $\omega_{rb_{i+1}}$ is reached. Finally, α_{max} is used till 't'. Similarly, in A.9(b), the sequence starts at ω_{rb_i} and crankshaft rotates at this speed till t_1. Then the maximum acceleration is used till the splitting speed, ω_{sp}. At ω_{sp}, a non-maximum acceleration of α is used till $\omega_{rb_{i+1}}$ is reached. The crankshaft stays at this speed till 't'.

Fig. A.10(a) and Fig. A.10(b) cannot be a dominant sequence as multiple jobs are released at a $\omega \notin \mathbf{\Omega_{rb}}$ and a combination of non-decreasing and non-increasing speed sequences is used in Fig. A.10(a) and Fig. A.10(b) respectively.

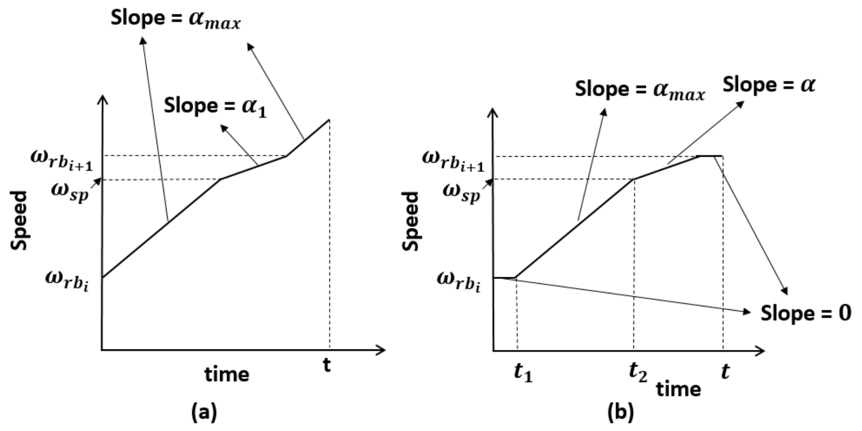


Figure A.9: Examples of non-decreasing speed sequences that can potentially be dominant sequences.

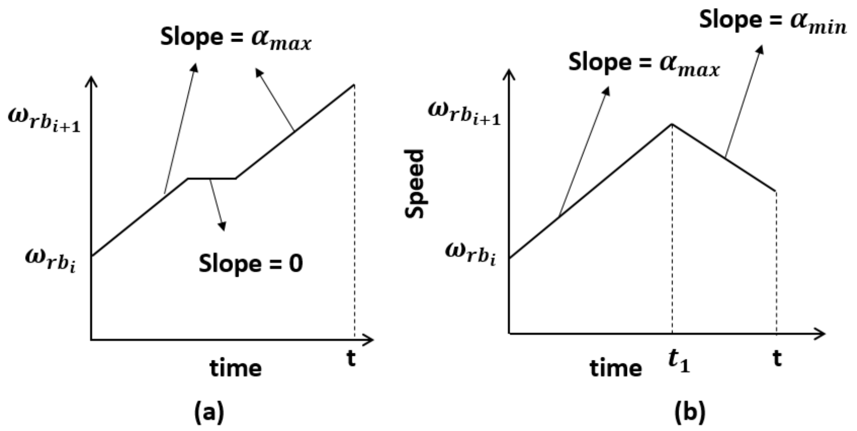


Figure A.10: Examples of sequences that cannot be dominant sequences.