Abstract

Title of Dissertation: Dynamic Time-Based Scheduling for Hard Real-Time Systems

Seonho Choi, Doctor of Philosophy, 1997

Dissertation directed by: Professor Ashok K. Agrawala
Department of Computer Science

In traditional time-based scheduling schemes for real-time systems time line is explicitly managed to obtain a feasible schedule that satisfies all timing constraints. In the schedule the task attributes, such as task start time, are statically decided off-line and used without modification throughout system operation time. However, for dynamic real-time systems, in which new tasks may arrive during the operation, or tasks may have relative constraints based on information only known at run-time, such static schemes may lack the ability to accommodate dynamic changes. Clearly a solution of dynamic real-time scheduling has to reflect the knowledge about tasks and their execution characteristics. In this dissertation we present a *dynamic time-based scheduling scheme* and show its applicability for three problem domains.

In dynamic time-based scheduling scheme attributes of task instances in the schedule may be represented as functions parameterized with information available at task dispatching time. These functions are called *attribute functions* and may denote any attribute of a task instance, such as lower and upper bound of its start time, its execution mode, etc. Flexible resource management becomes possible in this scheme by utilizing the freedom provided by the scheme.

First, we study the problem of dynamic dispatching of tasks, reflecting relative timing constraints among tasks. The relative constraints may be defined across the boundary of two consecutive scheduling windows as well as within one scheduling window. We present the solution approach with which we are not only able to test the schedulability of a task set, but also able to obtain maximum slack time by postponing static task executions at run-time.

Second, new framework is formulated for designing real-time control systems in which the assumption of fixed sampling period is relaxed. That is, sampling time instants are found adaptively based on physical system state such that a new cost function value is minimized which incorporates computational costs. We show, for linear time-invariant control systems, that the computation requirement can be reduced while maintaining the quality of control.

Third, acceptance tests are found for dynamically arriving aperiodic tasks, and for dynamically arriving sporadic tasks, respectively, under the assumption that an Earliest Deadline First scheduling policy is used for resolving resource contention between dynamic and static(dynamic) tasks.

Dynamic time-based scheduling scheme can be applied as solution approaches for these problems as will be shown in this dissertation, and its effectiveness will be demonstrated.

Dynamic Time-Based Scheduling for Hard Real-Time Systems

by

Seonho Choi

Dissertation submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1997

Advisory Committee:

Professor Ashok K. Agrawala, Chairman/Advisor
Professor Satish K. Tripathi
Professor Moon-Jhong Rhee
Associate Professor David Mount
Assistant Professor Jeff Hollingsworth

Dedication

To my parents and my wife

Contents

List of Tables

List of Figures

Dynamic Time-Based Scheduling for Hard Real-Time Systems

Seonho Choi

July 29, 1997

This comment page is not part of the dissertation.

Chapter 1

Introduction

Real-time computer systems are characterized by the existence of timing constraints on computations they carry out. The timing constraints are statically determined at pre-runtime from the characteristics of physical systems they interact with. In some real-time systems, called *hard real-time systems*, a timing failure is considered catastrophic and a guarantee should be given prior to execution that every timing constraint will be satisfied. Examples are found in application domains such as avionics, process control, automated manufacturing, robotics, etc.

Real-time systems of the next generation will be required to interact with more complex and dynamic systems [40, 2]. In such environments it will be required that a mechanism be provided to support high degree of concurrency, and to adapt itself to dynamically changing system state. Dynamic tasks such as aperiodic tasks with or without timing constraints may arrive at any time instant during system operation. Transient system overload may occur from dynamic nature of the system. Dynamic resource discovery and allocation methods, and methods of dynamically adapting to changing system conditions to assure or re-negotiate quality of service have to be supported by the real-time systems.

In this dissertation, we concentrate on the issues concerning how to achieve flexibility for hard real-time systems while not sacrificing the required quality of service. The new scheduling scheme, called *dynamic time-based scheduling*, is developed for this purpose. Then, this scheme is applied to three problems. First, it is addressed how to dynamically dispatch tasks in the presence of complex timing constraints such as relative timing constraints. Second, the issues are studied regarding dynamic adjustment of timing constraints, such as dynamic selection of task periods based on physical system state. Finally, it is studied how to incrementally schedule dynamic tasks such as aperiodic or sporadic tasks. The dynamic time-based scheduling scheme provides a sound basis for realizing the solution approaches derived.

1.1 Motivation

1.1.1 Scheduling with Relative Constraints

In some real-time systems complex timing constraints exist, such as jitter, separation, and relative deadline constraints, in addition to release time and deadline constraints [23]. Those constraints are usually specified between event occurrence times and are based on information(e.g. task completion time) which is only available at run-time. Such timing constraints make it more difficult to enhance the system with a capability to dynamically allocate CPU times to dynamic tasks while not hampering the schedulability guarantee given to tasks with complex timing constraints.

In priority-based preemptive systems, one of the approaches to schedule the tasks with jitter constraints is to separate the constrained event in the task into another task, and to associate highest priority with it. By doing this, the event occurrence times in consecutive periods can be made to be more predictable since higher priority tasks preempt lower priority tasks [36]. However, this approach can not be used efficiently when there exist many periodic tasks with jitter constraints, or when other types of relative constraints exist such as separation, or relative deadline constraints. Moreover, it is quite difficult to flexibly incorporate aperiodic task executions by postponing static task executions, when possible. It is the lack of ability to explicitly control task executions over a time line that causes these problems in priority-based systems. Some work has been done on scheduling aperiodic tasks and slack stealing algorithms in priority based scheduling systems [47, 18, 49, 32, 17, 24, 28, 48, 31]. However, most of their work assumes that only release time and deadline constraints are present. The complexity of optimal slack stealing algorithms in priority based systems is high [18, 17].

1.1.2 Dynamic Adjustment of Timing Constraints

Usually, the timing constraints of tasks are statically determined prior to system operation time from the characteristics of the physical system. Periodic task model is widely used and assumed in most real-time systems. One of the reasons for its popularity is that almost every control algorithm is formulated under the assumption of periodicity since it is easy to derive control laws under that assumption. Regardless of the current state of the system being controlled, the same period is used for a control task. The usual determination rule for task period is to select a task frequency to be 5-10 times the corresponding system's *characteristic frequency*. We study the issue of relaxing the periodicity assumption and propose a new control formulation, called *dynamic temporal control*, which dynamically decides the periods based on the system information such as current system state. To show the feasibility and benefit of this scheme, a solution approach is presented for a linear-time invariant control systems.

1.1.3 Scheduling Dynamic Tasks

A lot of work has been done on scheduling dynamic tasks such as aperiodic or sporadic tasks for priority-based scheduling systems [47, 18, 49, 32, 17, 24, 28, 48, 31]. However, only recently some results have been reported on scheduling aperiodic tasks on the basis of time-based scheduling scheme [22] in the presence of release time and deadline constraints. But, the solution approach presented in the paper is incomplete as explained in Chapter 2. We apply a dynamic time-based scheduling scheme for this problem, and develop acceptance tests for dynamically arriving aperiodic tasks, and for dynamically arriving sporadic tasks.

1.2 Our Approach

Two categories of tasks are considered in this dissertation:

- Static Tasks: Tasks whose invocation times are known at design time. Usually, these are periodic tasks, or those that have been converted to periodic tasks as in [38].

- Dynamic Tasks: Tasks whose executions are dynamically requested at run-time. These may be aperiodic or periodic.

In this dissertation every static task is executed in non-preemptive manner. That is, once a CPU is assigned to a task no preemption occurs until the task voluntarily releases the CPU or a maximum execution time allowed for the task expires.

1.2.1 Dynamic Time-based Scheduling Scheme

The dynamic time-based scheduling scheme consists of two components, an off-line scheduler that generates a dynamic calendar for static tasks, and a dynamic dispatcher that is responsible for scheduling both static and dynamic tasks while maintaining the total order for static tasks found by an off-line scheduler. The architecture of this scheduling system is shown in Figure 1.1.

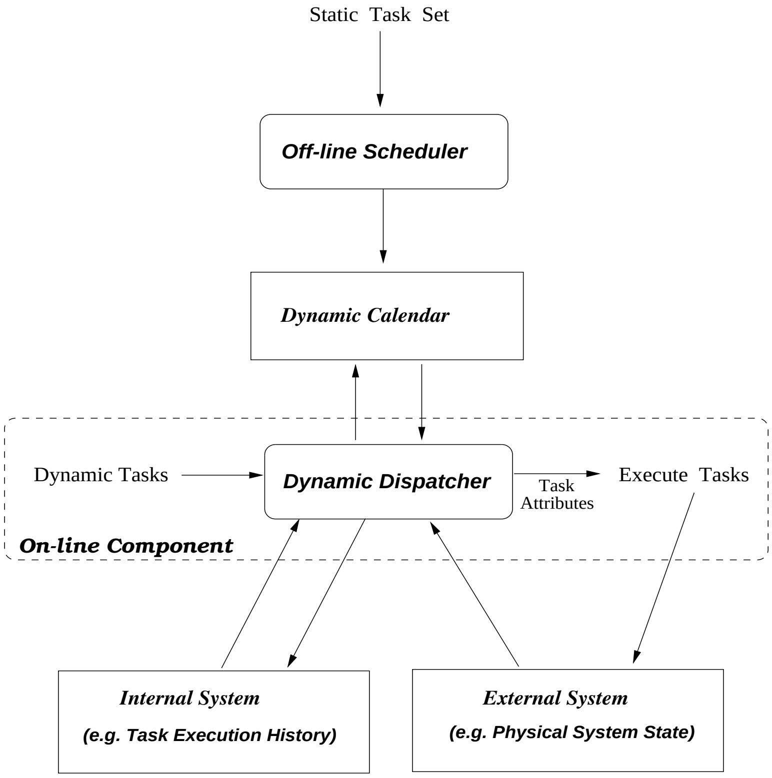

Figure 1.1: Overview of dynamic time-based scheduling scheme

A dynamic calendar is constructed from a totally ordered static task set found by an off-line scheduler. Each task in a dynamic calendar may have functions denoting any of its attributes, such as the task's valid start time range, its execution mode denoting which version of the task will be executed, etc. Those functions are called *attribute functions*. The functions may be parameterized with any information available to the system at the function evaluation time such as any attribute values of previously executed tasks, or current physical system state, etc.

At system operation time, the dynamic dispatcher makes use of any information available to it at a current time instant to dynamically evaluate the attribute functions for the next task in the dynamic calendar, and it decides the attributes of the next task(s), such as actual task start time, actual execution mode of the task if multiple task versions exist, etc. Then, it records the decided attributes of the next task for future usage.

1.2.2 Dynamic Dispatching with Complex Timing Constraints

The problem of scheduling tasks is studied when there exist complex timing constraints, such as relative inter-task constraints. The min/max constraints may be given between start or finish times of any two tasks. In this dissertation, it is assumed that a lower and upper bound of each task's execution time is known at pre-runtime, and the actual execution time may vary within those bounds. The non-deterministic execution times may make it infeasible to assign static start times to tasks at pre-runtime in the presence of the relative constraints between start or finish times of tasks.

To incorporate realistic relative constraints such as jitter constraints, a cyclic task model is defined with cyclic constraints allowed to be specified across the boundaries of scheduling windows. To apply the dynamic time-based scheduling scheme to this problem, the following questions have to be answered:

- How a total ordering among tasks can be found by the off-line scheduler?
- How a schedulability of a totally ordered task set can be checked in the presence of complex timing constraints?
- What is the structure of a dynamic calendar?
- How a dynamic calendar should be constructed for the total ordering such that all timing constraints imposed on tasks are guaranteed to be satisfied at run-time?
- How much is the overhead of dynamic dispatching at run-time?

The problem of deterministic scheduling has been well addressed in the literature. The solution approaches are based on either heuristic or approximation algorithms, or optimal schemes using implicit enumeration methods such as branch and bound search. In this dissertation, it is assumed that a total ordering of static tasks is given, and the rest of the issues mentioned above are addressed.

The dynamic time-based scheme is elaborated as follows for this problem:

- Each task in a dynamic calendar has two attribute functions, denoting lower and upper bounds for the task's start time.
- The attribute functions may be parameterized with start and finish times of already executed tasks.

With this refined dynamic time-based scheduling scheme, the solution approach is presented in Chapter 4.

1.2.3 Dynamic Dispatching with Variable Sampling Periods

Traditional control systems have been designed to exercise controls at regularly spaced time instants. When a discrete version of the system dynamics is used, a constant sampling interval is assumed and a new control value is calculated and exercised at each time instant.

In Chapter 5, we propose a new control scheme, *dynamic temporal control*, in which we not only calculate the control value at control computation time but also decide the time instant when the next control computation is done. The system state at control computation time is also used for obtaining the next control computation time as well as for calculating a new control value. Taking a discrete, linear, time-invariant system, and a cost function which reflects a cost for computation of the control values, as an example, we show the feasibility of using this scheme. We implement the dynamic temporal control scheme in a rigid body satellite control example and demonstrate the significant reduction in cost.

Also, the dynamic temporal control technique proposed can be implemented by using the dynamic time-based scheduling scheme under the assumptions given in Chapter 5.

1.2.4 Scheduling Dynamic Tasks

We present a solution approach in Chapter 6 for scheduling dynamically arriving aperiodic and sporadic tasks. It is assumed that a total ordering among static tasks is given at pre-runtime, and that only release time and deadline constraints are allowed. The total ordering among static tasks given initially is assumed to be maintained at run-time. Under this assumption, an EDF [1] scheduling algorithm is assumed to be used, and acceptance tests for dynamically arriving aperiodic tasks, and for dynamically arriving sporadic tasks, are derived, respectively. This solution approach seems to be a sound basis for extending the problem to allow more complex timing constraints.

1.3 Contributions

The main contributions of this dissertation are:

- We propose a new dynamic time-based scheduling scheme in which the dispatcher has the capability to dynamically decide the parameters(attributes) of the future tasks, such as start time or sampling period, while not affecting the guaranteed schedulability of future tasks. The dynamic decision is done based on the information available to the system at the decision time, such as start times and execution times of already executed tasks, or current physical system state and current system load.

- We develop a scheduling scheme which handles relative constraints, not only those defined between tasks within a scheduling window but also those cyclically defined across the boundaries of consecutive scheduling windows. Jitter constraint is a good example of such constraints.

- An algorithm is developed for checking the schedulability of a totally ordered cyclic task set. If the task set is schedulable, a dynamic calendar is constructed during the execution of the algorithm. The algorithm is based on variable elimination techniques. Also, we show that the run-time dispatching overhead is small, at most $O(N)$ execution time is spent for each

[1] EDF stands for Earliest Deadline First.

task instance for evaluating attribute functions where N is the number of task instances in one scheduling window.

- We present a new method for designing control systems in which the sampling periods are adaptively selected based on system states. Traditionally, control processes are implemented under the assumption of fixed sampling period. It is shown that, by dynamically selecting timing instants when new controls are calculated, the computational resource requirement can be greatly reduced while not sacrificing the quality of control. Linear time-invariant control system is used as an example to show the feasibility of the scheme. This result can be effectively used in an environment where computational resources can become scarce, e.g., in an overloaded situation.

- The acceptance tests are developed for dynamically arriving aperiodic or sporadic tasks when a time-based scheduling scheme is used to schedule static tasks. EDF scheduling algorithm is assumed to be used to resolve the conflicts between static and dynamic tasks.

1.4 Outline

The rest of the dissertation is organized as follows. We summarize prior work on real-time scheduling theory in Chapter 2. Then, in Chapter 3, we formally introduce the problem of scheduling tasks with relative timing constraints, present more detailed prior works related to the problem. In Chapter 4, we present a solution approach for the problem defined in Chapter 3 by utilizing dynamic time-based scheduling scheme. In Chapter 5, the dynamic temporal controller is developed for linear time-invariant control systems. In Chapter 6, acceptance tests are found for dynamic tasks such as aperiodic or sporadic tasks on the basis of a dynamic time-based scheduling scheme. Finally, concluding remarks and directions for future work are presented in Chapter 7.

Chapter 2

Prior Work

In this chapter we review previous work on real-time scheduling, and on real-time control systems. In Section 2.1, some relevant prior work on real-time scheduling theory is presented. In Section 2.2, prior work on scheduling with relative timing constraints is given. The previous work on design of real-time control systems is presented in Section 2.3. Finally, previous work on scheduling dynamic tasks in a time-based scheduling scheme is presented in Section 2.4. Some prior work is presented in more detail in the second part of Chapter 3 since they are directly related to our solution approach which will be presented in Chapter 4.

2.1 Real-Time Scheduling Theory

Scheduling algorithms in hard real-time systems may be classified in several ways. One way is to classify them in terms of how the scheduling is done. Priority-based scheduling schemes resolve the resource(CPU) contention between different tasks by making use of the fixed or dynamic priorities of the tasks. Another scheduling approach is time-based scheduling scheme in which the explicit time line is used to schedule the tasks. In traditional time-based scheduling schemes, all resource requirements are satisfied by statically assigning time intervals to the task instances at pre-runtime.

Each approach has its own advantages and disadvantages. Even though scheduling in priority based approach can be done in a simple manner, it lacks the capability of scheduling tasks with complex constraints such as precedence relations, and relative timing constraints, while the time-based approaches have that capability.

2.1.1 Fixed Priority Scheduling

In this scheme, fixed priority is assigned to each task which is used at runtime to resolve the resource contention. A task with a higher priority can preempt any lower priority task and thus the currently executing task has the highest priority among the tasks currently active(released). It is well known that rate monotonic scheduling algorithm is optimal for scheduling a set of independent periodic tasks with deadlines at the end of their periods [36]. It is optimal in a sense that it can schedule any set of tasks if that is schedulable by any fixed priority scheduling scheme. Any set of n tasks is schedulable according to rate monotonic scheduling scheme if the total utilization of the tasks doesn't exceed $n(2^{1/n}-1)$ which converges to $ln(2) \cong 0.69314718$ as n goes to ∞. This is a sufficient condition for a given set of tasks and not a necessary condition. The exact schedulability condition which is necessary and sufficient is found in [30] with the statistical simulation results showing that in general the utilization of the schedulable task set is higher than $ln(2)$.

A deadline monotonic scheduling algorithm is shown to be optimal for a set of tasks which have deadlines less than or equal to their periods. It assigns priorities according to their deadlines, the shorter the deadline, the higher priority is assigned regardless of the periods [33, 3]. For a set of tasks with arbitrary deadlines, it is shown that the optimal priority assignment can't be done in a simple priority assignment method, but requires a pseudo polynomial time algorithm [50].

A synchronization protocol becomes necessary when tasks use shared resources such as semaphores. Sharing resources may lead to a possible *priority inversion* when a higher priority task is blocked due to the lower priority task possessing the resource required by a higher priority task. This priority inversion may cause an unbounded blocking times. To solve this problem, several synchronization protocols have been developed. In a priority ceiling protocol [45], a priority ceiling is first assigned to each semaphore, which is equal to the highest priority of the tasks that may use this semaphore. Then, a task, τ, can start a new critical section only if τ's priority is higher than all priority ceilings of all the semaphores locked by tasks other than τ. In stack-based protocol [5], the concept of *preemption level* is used instead of the priorities to derive the protocol suitable for both fixed priority and dynamic priority based systems. Also, sharing of multiple-unit resources becomes possible with this protocol. The word "stack" is used in the sense that a task with higher preemption level can only preempt and thus block tasks with lower preemption level. Preemption levels are found statically reflecting synchronization constraints and resource requirements.

2.1.2 Dynamic Priority Scheduling

The priorities of tasks in dynamic priority scheme are decided at runtime. This means that the task instances from the same task may have different priorities at runtime while in the fixed priority scheme the same priority is used for scheduling them. The *earliest deadline first*(EDF) scheduling algorithm which assigns the highest priority to a task instance with the closest deadline is known to be optimal for a set of periodic or aperiodic tasks [36, 19]. The necessary and sufficient schedulability condition for a set of independent tasks with their deadlines equal to their periods is that the total processor utilization of the tasks should be less than or equal to 1 [36]. A dynamic priority ceiling protocol [10] and a stack-based protocol [5] have been developed for dynamic priority systems to enable the use of shared resources and to bound the blocking times. Note that the stack based resource allocation protocol may be used for both fixed priority and dynamic priority scheduling algorithms. Also, in [5], it is shown that the stack-based protocol provides a better schedulability test than that of dynamic priority ceiling protocol.

2.1.3 Static Time-based Scheduling

In a static time-based scheduling scheme, a calendar for a set of task instances is constructed at pre-runtime. At runtime this calendar is referred to execute each task instance at a scheduled time instant. Through off-line scheduling, we can schedule any set of tasks with various constraints, such as complex precedence relation, relative timing constraints, and other synchronization constraints. Even though the complexity of the off-line scheduling is NP-Complete in general, the scheduling can be done in a reasonable amount of time in most cases using techniques such as branch and bound or heuristic search algorithms [52, 21, 12, 56]. It has been shown that the complexity of non-preemptive scheduling can be dramatically reduced in many cases by *decomposition scheduling* approach where task instances are decomposed into a sequence of subsets, which are scheduled independently [54]. Also, the time based scheduling scheme can efficiently schedule task sets with relative timing constraints which can't be easily accommodated in priority-based systems [23, 12]. Because of these reasons, it is claimed that the time-based scheduling scheme is the most appropriate scheduling approach for *hard* real-time systems [53].

2.2 Scheduling with Relative Timing Constraints

In some hard real-time systems, *relative timing constraints* should be satisfied between event occurrence times. as well as *release time* and *deadline* constraints on tasks. For example, control output events in two successive instances of a periodic task may have to occur with the jitter requirement satisfied. That is, the difference of two event occurrence times, called *jitter*, should lie between a lower and an upper bound. The occurrences of events in different tasks may also be constrained from the requirements and characteristics of the environment by separation or relative deadline constraints [23]. These relative constraints have to be enforced in many real-time control systems such as process control systems and flight control systems [9], etc. For example, in process control systems, it has been shown that jitter constraints have more influence on control systems performance than the frequency constraints [29].

Usually, the relative constraints between events are transformed into relative constraints between start or finish times of the tasks to make feasible the process of scheduling and dispatching of task instances [26, 23]. In [26] a preemptive fixed priority scheduling algorithm is developed to schedule periodic tasks with relative deadline constraints between finish times of two successive instances of periodic tasks. However, other types of relative constraints are not allowed in that work and it is not possible to flexibly manage slack times at runtime for dynamic tasks. In [23] dispatching of a totally ordered non-preemptive task instance set with any min/max constraints is studied and a new scheduling mechanism called *parametric scheduling* is developed. In that paper, it is also shown that a traditional static scheduling approach, in which task instance start times are statically scheduled under the assumption that every task instance spends its worst case execution time, doesn't work any more for task instance sets with general min/max constraints even when a total ordering among them is given. Furthermore, in parametric scheduling scheme, it is possible to effectively schedule aperiodic tasks at run-time by dynamically managing task instance start times. However, the task instance set in parametric scheduling scheme consists of a finite number of task instances with a finite number of constraints. This implies that the approach cannot be applied to a periodic task model, since periodic tasks may invoke an infinite number of task instances with an infinite number of relative constraints. In a traditional time-based scheduling scheme the task start times are statically decided in a scheduling window, and this static schedule is cyclically used at run-time. In the presence of jitter constraints between start times of non-preemptive task instances, the problem of finding a static schedule has been addressed in [11]. However, this static cyclic scheduling approach only allows certain types of min/max constraints to be specified, and it only works under low utilization. Moreover, it is very difficult to flexibly manage task start times at run-time to incorporate any dynamic tasks such as aperiodic tasks into the schedule.

2.3 Control

Rich literature exists on the design of controllers. However, nearly all the results develop control laws under the assumption of equal sampling periods. In addition, taking computation time delay into consideration for real-time computer control has been studied in several research papers [6, 25, 27, 41, 46, 55]. However, to the best of our knowledge, the *dynamic temporal control* approach which is explained in Chapter 5 has not been studied in the past.

In dynamic temporal control, the computational cost is incorporated into the cost function and the time instants for performing control computations are chosen to minimize this cost function. With this new approach, we can perform the same quality of control with fewer control computations compared to the traditional approaches [1].

2.4 Scheduling Aperiodic and Sporadic Tasks

Scheduling of dynamic tasks such as aperiodic or sporadic tasks has been studied extensively for priority-based scheduling systems. In this section, those works are summarized as well as a recent work on aperiodic task scheduling problem on the basis of time-based scheduling scheme [22].

2.4.1 Scheduling Dynamic Tasks in Fixed Priority Systems

Hard and non-realtime aperiodic tasks can be scheduled within a fixed priority scheduling scheme. One approach is to utilize the *aperiodic server* concept in which a certain percentage of the processor utilization is reserved for servicing the aperiodic tasks. That is, one or several periodic tasks are reserved for servicing aperiodic tasks. Several algorithms have been developed and their performances have been compared [31, 47]. Another approach is slack stealing approach which tries to utilize as much processor time as possible by postponing the execution of hard periodic task executions as long as the schedulability of the hard tasks is not affected [18, 32, 49]. The optimal slack stealing algorithm is found to be pseudo polynomial [18] and several approximation algorithms have been proposed [17].

2.4.2 Scheduling Dynamic Tasks in Dynamic Priority Systems

An aperiodic task scheduling problem has been studied under the assumption that only hard periodic tasks exist [28, 24]. The $O(N)$ acceptance test for a hard aperiodic task is given when a set of independent periodic tasks is scheduled by EDF where N is the total number of task instances in an LCM[1] of the periods of periodic tasks [14, 13, 15]. Aperiodic scheduling schemes for EDF have been proposed and studied and the Improved Priority Exchange Algorithm is shown to perform well [48].

[1] LCM stands for Least Common Multiple.

2.4.3 Scheduling Dynamic Tasks in Time-based Systems

The aperiodic task scheduling problem in time-based scheduling scheme has been addressed in the paper by Fohler *et al.* [22]. The initial schedule is assumed to be given and the arriving aperiodic tasks are scheduled at runtime. First, the deadlines of task instances, τ_j, in the time-based schedule are sorted and the schedule is divided into a set of *disjoint execution intervals*, I_i. Then, the *spare capacities* are defined for these intervals, which may be used to schedule arriving aperiodic tasks. Several task instances with the same deadline constitute one interval and the end of an interval, $end(I_i)$, is defined to be the deadline of the last task instance in the interval. The earliest start time of an interval is defined to be the minimum of the earliest start times of its constituting task instances. And, the start time of the interval, $start(I_i)$ is defined to be the maximum of its earliest start time or the end of the previous interval. The length of an interval, $\mid I_i \mid$ is defined to be $end(I_i) - start(I_i)$. Then, the spare capacity is defined recursively as:

$$\begin{aligned} sc(I_i) &= \mid I_i \mid - \sum_{\tau_j \in I_i} C_j + \min(sc(I_{i+1}), 0) \\ sc(I_e) &= \mid I_e \mid - \sum_{\tau_j \in I_e} C_j \end{aligned}$$

where C_j denote the worst case execution time of τ_j and I_e is the last interval in the schedule. Note that the spare capacity may have a negative value reflecting the fact that the borrowed spare capacity from the previous interval is used to schedule the task instances in the current interval. Figure 2.1 shows an example case of this. In this example, the spare capacities for I_2 and I_1 are found to be:

$$sc(I_2) = 2 - 3 = -1$$

$$sc(I_1) = 8 - 3 + \min(-1, 0) = 4$$

These spare capacities are used to schedule arriving aperiodic tasks and adjusted whenever the aperiodic tasks are accepted.

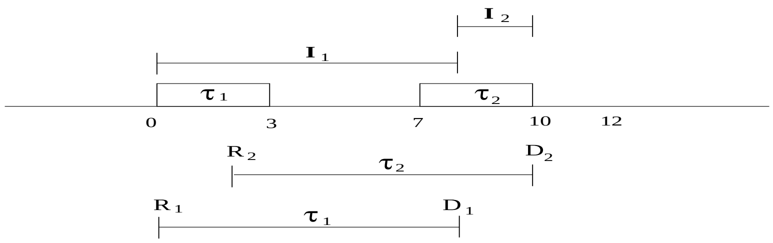

Figure 2.1: Example case.

However, no consideration is given about how to obtain correct spare capacities when the deadlines of the task instances are not in increasing order in the schedule. For example, no correct spare capacity can be obtained in the example case shown in Figure 2.2.

According to the algorithm presented in that paper, we have two execution intervals, $[10, 12]$ and $[0, 10]$. The spare capacities in these intervals are:

$$\begin{aligned} sc([10, 12]) &= 2 - 7 = -5 \\ sc([0, 10]) &= 10 - 3 - 5 = 2 \end{aligned}$$

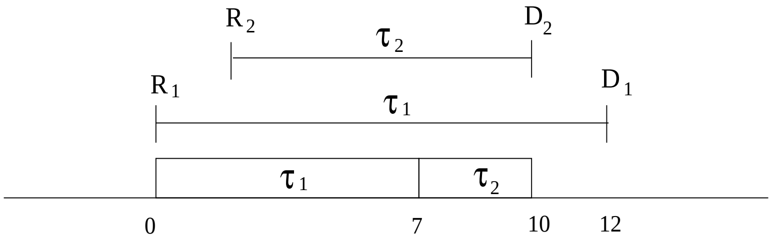

Figure 2.2: No spare capacities can be found.

This result shows that, in an execution interval $[0, 10]$, a spare capacity of 2 is found. However, as can be seen in Figure 2.2, zero spare capacity should have been found in an interval $[0, 10]$. This shows that their solution approach is incomplete.

2.5 Summary

We have presented a brief overview of the related work on real-time scheduling and control systems design. The works by Cheng *et al.* [11] and Gerber *et al.* [23] are combined and extended in Chapter 4 for scheduling tasks with relative timing constraints. Our solution approach overcomes the limitations of those previous approaches and provides more flexible and unified ways for scheduling tasks with complex timing constraints. Also, Fohler *et al.* [22] propose a mechanism to flexibly manage slack times in time-based scheduling scheme. However, their approach is un-necessarily complicated and incomplete as shown in the previous section. Our approach presented in Chapter 6 provides more intuitive and complete solution. Instead of spare capacities, we define slack values which can be obtained by postponing static tasks in the schedule, and make use of them to schedule dynamic tasks.

Chapter 3

Scheduling with Relative Constraints

We formulate the problem of scheduling a set of tasks with relative constraints, and present its solution in the next chapter. We also present some prior works in detail since our solution approach is based on parametric dispatching approach [23] developed for a transaction scheduling problem.

In Section 3.1, the problem is formally defined. Then two prior works are presented on scheduling with relative constraints, static approach and dynamic parametric approach. Finally, a brief summary is presented.

3.1 Problem Description

A task instance is called a *job* and these two terms will be used inter-changeably throughout the dissertation. Let $\Gamma^j = \{\tau_i^j \mid i = 1, \ldots, N\}$ denote an ordered set of N jobs to be dispatched sequentially in a j-th scheduling window $[(j-1)L, jL]$ where L is a positive integer denoting a scheduling window size.. The jobs are executed non-preemptively in this order. At runtime, this job set will be cyclically scheduled in consecutive scheduling windows. In other words, τ_i^j and τ_i^k are jobs of the same task.

Then, let $\Gamma^{1,k} = \Gamma^1 \cup \Gamma^2 \cup \ldots \cup \Gamma^k$ denote a set of jobs to be executed in a time interval $[0, kL]$. Each job τ_i^j ($j \geq 1$, $1 \leq i \leq N$) has the following set of parameters that may have integer values:

- A runtime variable s_i^j denoting the actual start time of τ_i^j
- A runtime variable e_i^j representing the actual execution time spent for τ_i^j
- A runtime variable $f_i^j = s_i^j + e_i^j$ denoting the actual finish time of τ_i^j
- A constant l_i^j corresponding to the minimum execution time of τ_i^j
- A constant u_i^j denoting the maximum execution time of τ_i^j.

Note that it is simply assumed that execution times of jobs are nondeterministic and bounded from above and below, which is a realistic assumption in many real-time systems.

Standard constraints are defined next that may be imposed on $\{s_i^j, e_i^j \mid 1 \leq j \leq k,\ 1 \leq i \leq N\}$ for $\Gamma^{1,k}$.

Definition 3.1 (Standard Constraints) *A standard constraint involves the variables of at most two jobs, τ_a^j and $\tau_b^l (1 \leq a \leq b \leq N, \mid j - l \mid \leq 1)$, where s_a^j (or $s_a^j + e_a^j$) appears on one side of "$\leq$," and s_b^l (or $s_b^l + e_b^l$) appears on the other side of the "$\leq$." For two jobs, τ_a^j, τ_b^l, the following constraints are permitted(where c_i is an arbitrary constant) and called* relative *standard constraints:*

$$
\begin{array}{rcl}
s_a^j - s_b^l & \leq & c_1 \\
s_a^j - (s_b^l + e_b^l) & \leq & c_2 \\
s_a^j + e_a^j - s_b^l & \leq & c_3 \\
s_a^j + e_a^j - (s_b^l + e_b^l) & \leq & c_4
\end{array}
\qquad
\begin{array}{rcl}
s_b^l - s_a^j & \leq & c_5 \\
s_b^l - (s_a^j + e_a^j) & \leq & c_6 \\
s_b^l + e_b^l - s_a^j & \leq & c_7 \\
s_b^l + e_b^l - (s_a^j + e_a^j) & \leq & c_8
\end{array}
\tag{3.1}
$$

In addition, each job has release time and deadline constraints. These constraints are called absolute *standard constraints. A job τ_a^j has the following absolute constraints:*

$$
c_9 \leq s_a^j \qquad\qquad s_a^j + e_a^j \leq c_{10} \tag{3.2}
$$

We also include as standard any constraint that can be rewritten in one of the above forms; e.g., $s_a^j \geq s_b^l + e_b^l - e_a^j + c$ falls into this category.

Next, the *k-fold cyclically constrained job set* is formally defined. [1] Any $\Gamma^{1,k}$ considered in this dissertation belongs to this class.

Definition 3.2 (k-fold Cyclically Constrained Job Set) *A job set $\Gamma^{1,k} = \Gamma^1 \cup \Gamma^2 \cup \ldots \cup \Gamma^k$ $(k = 1, 2, \ldots, \infty)$ is classified as a k-*fold cyclically constrained job set *if it has the following linear constraints:*

1. *The set of standard* relative *constraints:*

$$
\forall j \in [1, k) :: A_1 \mathbf{x}^j + A_2 \mathbf{x}^{j+1} \leq \mathbf{a} \tag{3.3}
$$

where $\mathbf{x}^j$ is a $2N$-dimensional column vector $[s_1^j, e_1^j, s_2^j, e_2^j, \ldots, s_N^j, e_N^j]^T$. A_1, A_2 are $m_1 \times 2N (m_1 \geq 0)$ matrices of 0, 1, or -1, and $\mathbf{a}$ is an m_1-dimensional column vector whose elements are integers. Included in the m_1 constraints are those denoting the total ordering on jobs:

$$
\forall j \in [1, k] :: \forall i \in [1, N) :: s_i^j + e_i^j \leq s_{i+1}^j
$$

$$
\forall j \in [1, k) :: s_N^j + e_N^j \leq s_1^{j+1}
$$

2. *The set of release time and deadline constraints:*

$$
\forall j \in [1, k] :: B \mathbf{x}^j \leq \mathbf{b}^j \tag{3.4}
$$

$$
\forall j \in [1, k] :: D \mathbf{x}^j \leq \mathbf{d}^j \tag{3.5}
$$

[1]Note that k may be equal to ∞.

where $\mathbf{b}^j$ *is an* m_2*-dimensional column vector of non-positive integers satisfying:*

$$\mathbf{b}^j = \mathbf{b}^1 + (1-j)L$$

and $\mathbf{d}^j$ *is an* m_3*-dimensional column vector of non-negative integers satisfying:*

$$\mathbf{d}^j = \mathbf{d}^1 + (j-1)L$$

We define $\mathcal{C}^{1,k}$ *to represent the logical conjunction of the constraints induced by each row of (3.3), (3.4), and (3.5).*

In the above definition, the same matrices A_1, A_2, B, D are cyclically used to represent the standard constraints on consecutive job sets.

The example job set shown in Figure 3.1 is presented here with corresponding matrices and vectors defined in (3.3), (3.4), and (3.5).

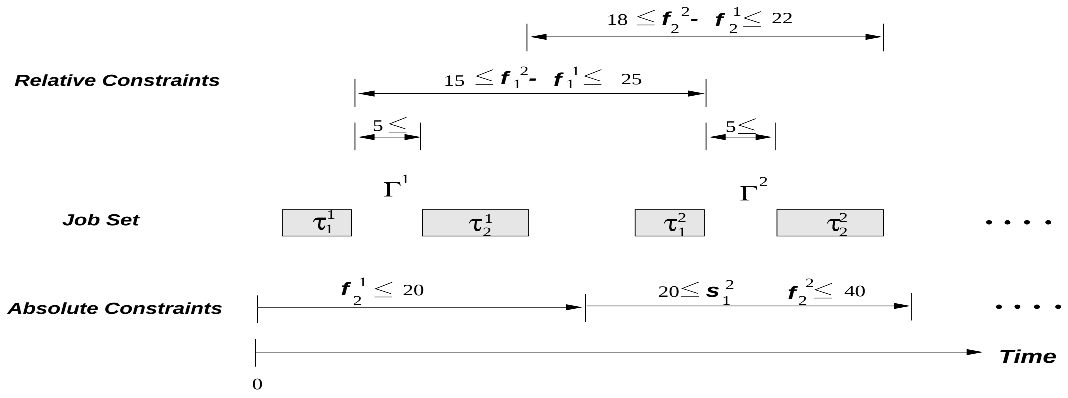

Figure 3.1: Example Job Sequence

Example 3.1 Consider the example job set depicted in Figure 3.1. Each job set Γ^j, $1 \leq j \leq k$, consists of two jobs, τ_1^j and τ_2^j (i.e. $N = 2$), whose execution time bounds are:

$$\begin{array}{rclcrcl} l_1^j & = & 5 & \quad & u_1^j & = & 8 \\ l_2^j & = & 8 & \quad & u_2^j & = & 10 \end{array}$$

The standard relative constraints defined within Γ^j or within Γ^{j+1} are:

$$\begin{array}{rcl} 5 & \leq & s_2^j - (s_1^j + e_1^j) \\ s_1^j + e_1^j & \leq & s_2^j \end{array} \qquad \begin{array}{rcl} 5 & \leq & s_2^{j+1} - (s_1^{j+1} + e_1^{j+1}) \\ s_1^{j+1} + e_1^{j+1} & \leq & s_2^{j+1} \end{array} \tag{3.6}$$

Similarly, the set of standard relative constraints across the boundary of Γ^j and Γ^{j+1} are:

$$\begin{array}{rcl} s_1^j + e_1^j + 15 & \leq & s_1^{j+1} + e_1^{j+1} \\ s_1^{j+1} + e_1^{j+1} & \leq & s_1^j + e_1^j + 25 \\ s_2^j + e_2^j & \leq & s_1^{j+1} \end{array} \qquad \begin{array}{rcl} s_2^j + e_2^j + 18 & \leq & s_2^{j+1} + e_2^{j+1} \\ s_2^{j+1} + e_2^{j+1} & \leq & s_2^j + e_2^j + 22 \end{array} \tag{3.7}$$

Finally, the absolute constraints on Γ^j and Γ^{j+1} are:

$$\begin{array}{rclrcl} 20(j-1) & \leq & s_1^j & 20j & \leq & s_1^{j+1} \\ 20(j-1) & \leq & s_2^j & 20j & \leq & s_2^{j+1} \\ s_1^j + e_1^j & \leq & 20j & s_1^{j+1} + e_1^{j+1} & \leq & 20(j+1) \\ s_2^j + e_2^j & \leq & 20j & s_2^{j+1} + e_2^{j+1} & \leq & 20(j+1) \end{array} \tag{3.8}$$

All standard relative constraints can be denoted by the following inequality:

$$\begin{bmatrix} 1 & 1 & -1 & 0 \\ 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} s_1^j \\ e_1^j \\ s_2^j \\ e_2^j \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & 1 & -1 & 0 \\ -1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} s_1^{j+1} \\ e_1^{j+1} \\ s_2^{j+1} \\ e_2^{j+1} \end{bmatrix} \leq \begin{bmatrix} -5 \\ 0 \\ -5 \\ 0 \\ -15 \\ 25 \\ 0 \\ -18 \\ 22 \end{bmatrix}$$

And, the set of absolute constraints is represented by the following inequality:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} s_1^j \\ e_1^j \\ s_2^j \\ e_2^j \end{bmatrix} \leq \begin{bmatrix} -20(j-1) \\ -20(j-1) \\ 20j \\ 20j \end{bmatrix}$$

One traditional approach for scheduling with complex timing constraints is a time-based scheduling scheme that assigns static start times to the jobs in the scheduling window such that the relative constraints are satisfied if the static schedule is cyclically repeated at runtime. However, this approach can't be used in the presence of arbitrary relative constraints between start or finish times of jobs [23]. Also, this approach suffers from the loss of schedulability problem. Some task sets are not schedulable in this approach, even though they are schedulable if our approach is employed. This will be explained through an example later. To cope with some of the above limitations the parametric scheduling scheme was developed in the context of real-time transaction scheduling [23]. However, as far as we know, the solution approach has not been found for general periodic task models where jobs in different scheduling windows may have relative constraints. The objective of the next chapter is to develop a schedulability test for $\Gamma^{1,\infty}$, and to develop a flexible job dispatching mechanism for schedulable job sets, $\Gamma^{1,\infty}$.

3.2 Prior Work

In this section, we briefly describe two scheduling schemes closely related to ours. The first one is the static cyclic scheduling scheme [11] and the second one is the parametric scheduling scheme [23].

3.2.1 Static Cyclic Scheduling

The static cyclic scheduling problem has been studied in [11]. The periodic task model is used, which means that every job has a release time and a deadline constraints, and only the jitter constraints between two job start times are allowed. An important assumption made in the work is that the start times of jobs in Γ^j are statically determined as offsets from the start of the j-th scheduling window $[(j-1)L, jL]$, and this schedule is invoked repeatedly by wrapping around the end point of the current schedule to the start point of the next. In other words, $s_i^{j+1} = s_i^j + L$ holds for all $1 \leq j$.

In the presence of jitter constraints, the job start times should be chosen carefully such that the jitter constraints are satisfied at run-time as well as the absolute constraints. Obtaining the ordering and job start times is an NP-hard problem, since non-preemptive scheduling problem with release time and deadline constraints is NP-hard. Several priority based non-preemptive scheduling algorithms are presented and their performances are compared in [11].

Suppose that a job $\tau_{i_1}^j$ belongs to Γ^j, and a job $\tau_{i_2}^{j+1}$ belongs to Γ^{j+1}, and they have jitter constraints $c_1 \leq s_{i_2}^{j+1} - s_{i_1}^j \leq c_2$ $(0 < c_1 \leq c_2 \leq L)$. From the above assumption, $s_{i_2}^{j+1} = L + s_{i_2}^j$ holds. Thus, a new constraint is created, $c_1 - L \leq s_{i_2}^j - s_{i_1}^j \leq c_2 - L$, which is again equal to $L - c_2 \leq s_{i_1}^j - s_{i_2}^j \leq L - c_1$. Therefore, the jitter constraints across the boundary of Γ^j and Γ^{j+1} are transformed into jitter constraints between two jobs in Γ^j. As a consequence, if we can find a static schedule for Γ^j that satisfy the above transformed constraints and the constraints between jobs within Γ^j, it is clear that all timing constraints will be satisfied if that schedule is repeatedly used at run-time. This approach is depicted in figure 3.2.

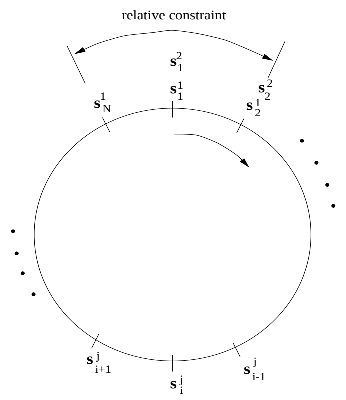

Figure 3.2: Static Cyclic Scheduling

However, this approach suffers from the following limitations:

- The relative constraints allowed are limited to jitter type constraints between start times of two jobs.
- The schedulability of job sets are reduced due to the static start time assignments.
- It is very difficult to effectively incorporate dynamic tasks, such as aperiodic tasks, into a schedule by dynamically adjusting the start times of the jobs.

In some real-time applications, the jitter constraints may be imposed between the finish times of the jobs rather than between the start times [26]. Furthermore, a periodic task may be decomposed into several subtasks and any kind of standard constraints may be defined between these subtasks [23]. In these cases this static scheduling approach is no more applicable without sacrificing the schedulability [23].

By transforming the jitter constraints across the boundary of Γ^j and Γ^{j+1} into those between jobs within Γ^j, we are affecting the schedulability of job sets. We will show that, under our new scheduling scheme in which this transformation is not necessary, the schedulability of job sets is increased, i.e., some job sets are not schedulable according to this scheme whereas it is schedulable by our scheme.

3.2.2 Parametric Scheduling

Gerber *et al.* [23] proposes a parametric scheduling scheme in the context of transaction scheduling, in which any standard constraints may be given between jobs in one transaction. Let $\Pi =< \tau_1, \ldots, \tau_N >$ denote a sequence of jobs constituting one transaction with a set of standard constraints, $\mathcal{C}$. Also, let l_i and u_i denote a lower and upper bound of τ_i's execution time, respectively.

In the presence of standard constraints between start or finish times of tasks, it may not be possible to statically assign start times to tasks in the scheduling window by using the maximum execution time(u_i^j) as the worst case execution time for each job. This is due to the nondeterministic execution times of tasks and the existence of standard constraints involving the finish times of tasks. This is well explained with the following example [42].

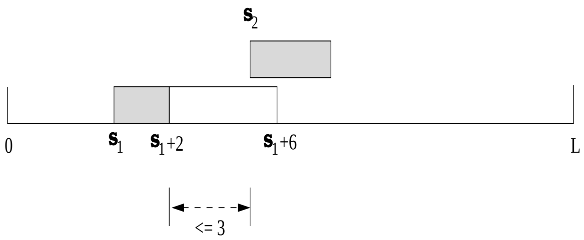

Figure 3.3: Limitation of Static Scheduling Scheme

Consider a simple example shown in Figure 3.3 which consists of two jobs, τ_1 and τ_2. Suppose that $l_1 = 2$, $u_1 = 6$, and there exists a constraint $s_2 - f_1 \leq 3$. In this example, it is not possible to statically assign start times for two jobs due to large variability of first job's execution time and due to the existence of relative deadline constraint between first job's finish time and second job's start time. However, if we allow the start time s_2 for τ_2 be parameterized with f_1, then all the constraints are satisfied under all execution scenarios.

In [42], a parametric schedulability of Π is defined as follows:

$$Sched \equiv \exists s_1 :: \forall e_1 \in [l_1, u_1] :: \ldots :: \exists s_N :: \forall e_N \in [l_N, u_N] :: \mathcal{C} \quad (3.9)$$

From this *Sched* predicate, parametric lower and upper bound functions for each start time s_i are obtained by eliminating the variables in an order e_N, s_N, …, e_i. The parametric lower and upper bound functions, denoted as $\mathcal{F}_{s_i}^{min}$ and $\mathcal{F}_{s_i}^{max}$, are parameterized in terms of the runtime variables, s_1, e_1, ..., s_{i-1}, e_{i-1} of already executed jobs. The parametric calendar structure is shown in figure 3.4.

$\mathcal{F}^{min}_{s_1}()$	$\leq$	s_1	$\leq$	$\mathcal{F}^{max}_{s_1}()$
$\mathcal{F}^{min}_{s_2}(s_1, e_1)$	$\leq$	s_2	$\leq$	$\mathcal{F}^{max}_{s_2}(s_1, e_1)$
	$\vdots$		$\vdots$	
$\mathcal{F}^{min}_{s_N}(s_1, e_1, s_2, \ldots, s_{N-1}, e_{N-1})$	$\leq$	s_N	$\leq$	$\mathcal{F}^{max}_{s_N}(s_1, e_1, s_2, \ldots, s_{N-1}, e_{N-1})$

Figure 3.4: Parametric Calendar Structure

This parametric calendar is obtained from an off-line component of the algorithm by applying variable elimination techniques that will be given later in this section, and the actual bounds of s_i are found at runtime by evaluating the parametric functions in the parametric calendar by using the start and finish times of already executed jobs, τ_1, ..., τ_{i-1}. The actual form of these parametric functions are given in the following proposition which is obtained from the paper by Saksena *et al.* [23]. This proposition will be used in deriving our solution approach in Chapter 4.

Proposition 3.1 (Parametric Bound Functions [23]) *A parametric lower bound function for s_j is of the following form:*

$$\begin{aligned} & \mathcal{F}^{min}_{s_j}(s_1, f_1, \ldots, s_{j-1}, f_{j-1}) \\ = \quad & \max(p_1 + c_1, p_2 + c_2, \ldots, p_a + c_a, \alpha^{min}_j) \end{aligned} \tag{3.10}$$

where each p_i, $1 \leq i \leq a$, belongs to $\{s_1, f_1, \ldots, s_{j-1}, f_{j-1}\}$, and c_i is an arbitrary constant.[2] *And, α^{max}_j is a non-negative integer.*

Similarly, a parametric upper bound function for s_j is of the following form:

$$\begin{aligned} & \mathcal{F}^{max}_{s_j}(s_1, f_1, \ldots, s_{j-1}, f_{j-1}) \\ = \quad & \min(q_1 + d_1, q_2 + d_2, \ldots, q_b + d_b, \alpha^{max}_j) \end{aligned} \tag{3.11}$$

where each q_i, $1 \leq i \leq b$, belongs to $\{s_1, f_1, \ldots, s_{j-1}, f_{j-1}\}$, and d_i is an arbitrary constant..

The main result obtained is that, for an arbitrary set of standard constraints on $\Pi = \{\tau_1, \ldots, \tau_N\}$, we can find the parametric calendar in $O(N^3)$ time and the run-time evaluation of each bound function can be carried out in $O(N)$ time.

By applying this parametric scheduling scheme, we are not only able to schedule any sequence of jobs with standard constraints, but also able to take advantage of the flexibility offered by the scheme. That is, the job start times may be decided dynamically at runtime to incorporate other dynamic activities in the system. Even though this scheme is directly applicable to our k-fold cyclically constrained job sets, if the number of jobs in $\Gamma^{1,k}$ becomes large, the bounds need to be found on the size of parametric functions and for the memory requirements for them.

In the rest of this section, the parametric scheduling scheme is presented with an example.

[2] Note that $f_i = s_i + e_i$.

Elimination of Quantified Variables

Consider a set of linear constraints C in n variables $(x_1, x_2, \ldots, x_n)$,

$$C \equiv H\mathbf{x} \leq \mathbf{h}$$

which must be satisfied with respect to some defined existential and universal quantification over the variables. In this section we show how an innermost universally quantified variable x_n, with associated lower (l_n) and upper (u_n) bounds can be eliminated to obtain a new set of equivalent constraints. The set of constraints C may be partitioned into three subsets, depending on whether the coefficient of x_n is positive, negative or zero. Thus,

$$C \equiv C_P \wedge \; C_N \wedge C_Z$$

where

$$\begin{array}{rcl} C_P & \equiv & \{x_n \geq D_i(\mathbf{x}'),\ 1 \leq i \leq p\} \\ C_N & \equiv & \{x_n \leq E_j(\mathbf{x}'),\ 1 \leq j \leq q\} \\ C_Z & \equiv & \{0 \leq F_k(\mathbf{x}'),\ 1 \leq k \leq r\} \end{array}$$

$D_i(\mathbf{x}'), E_j(\mathbf{x}'), F_k(\mathbf{x}')$ are linear functions of $\mathbf{x}' = [x_1, \cdots, x_{n-1}]^T$. The elimination of variable x_n leads to a new system of constraints C' obtained from C by substituting x_n with l_n or u_n, depending on its coefficient:

$$C' \equiv (C_P)^{x_n}_{l_n} \wedge \; (C_N)^{x_n}_{u_n} \wedge \; (C_Z)$$

The following lemma has been derived and proved in the paper by Saksena *et al.*[23], and shows the validity of the above variable elimination process.

Lemma 3.1 ([23]) *Let* C *be a system of linear constraints and let* C$'$ *be the resulting set of constraints after eliminating a universally quantified variable* x_n *with lower bound* l_n *and upper bound* u_n. *Then the sentence* $\forall x_n \in [l_n, u_n] ::$ C *holds if and only if* C$'$ *holds.*

The existential quantifier can be eliminated by using Fourier-Motzkin variable elimination technique [16].

Fourier-Motzkin Elimination. Consider a system of linear constraints C in n variables $(x_1, x_2, \ldots, x_n)$. We wish to find a system of linear constraints C' over $\mathbf{x}' = [x_1, \ldots, x_{n-1}]^T$, such that $\mathbf{x}'$ is a solution to C' if and only if $\mathbf{x}'$ is a solution to $\exists x_n ::\ C$. As before, the constraints in C may be partitioned into three subsets.

$$C \equiv \left\{ \begin{array}{rcll} x_n & \geq & D_i(\mathbf{x}'), & 1 \leq i \leq p \\ x_n & \leq & E_j(\mathbf{x}'), & 1 \leq j \leq q \\ 0 & \leq & F_k(\mathbf{x}'), & 1 \leq k \leq r \end{array} \right.$$

The elimination of variable x_n leads to a new system of constraints:

$$C' \equiv \exists x_n ::\ C \equiv \left\{ \begin{array}{rcll} D_i(\mathbf{x}') & \leq & E_j(\mathbf{x}'), & 1 \leq i \leq p, \quad 1 \leq j \leq q \\ 0 & \leq & F_k(\mathbf{x}'), & 1 \leq k \leq r \end{array} \right.$$

Again, the following lemma has been derived and proved in the paper by Saksena *et al.*[23], and shows the validity of the above variable elimination process.

Lemma 3.2 ([23]) *Let* C *be a set of linear constraints. Let* C′ *represent the set of constraints as a result of eliminating* x_n *using Fourier Motzkin elimination as described above. Then,*

$$\exists x_n :: \ \mathrm{C}$$

holds if and only if C′ *holds.*

Example

This example is based on the work presented in the paper by Saksena *et al.*[23]. Here, the variable elimination technique is applied to

$$\exists s_1 :: \forall e_1 \in [5,8] :: \exists s_2 :: \forall e_2 \in [8,10] :: \exists s_3 :: \forall e_3 \in [5,8] :: \exists s_4 :: \forall e_4 \in [8,10] :: \mathcal{C}^{1,2}$$

where $\mathcal{C}^{1,2}$ is a constraint set given on $\Gamma^{1,2}$ in Example 3.1. Initially, since e_4 is the innermost universally quantified variable, it can be eliminated first. The constraints involving e_4 in $\mathcal{C}^{1,2}$ are:

$$\begin{array}{lcl} s_4 + e_4 & \leq & 40 \\ s_4 + e_4 - (s_2 + e_2) & \leq & 22 \\ 18 & \leq & s_4 + e_4 - (s_2 + e_2) \end{array}$$

The elimination of e_4 from these constraints results in the following derived constraints:

$$\begin{array}{lcll} s_4 & \leq & 30 & (e_4 := u_4 = 10) \\ s_4 - (s_2 + e_2) & \leq & 12 & (e_4 := u_4 = 10) \\ 10 & \leq & s_4 - (s_2 + e_2) & (e_4 := l_4 = 8) \end{array}$$

Therefore, these three constraints are substituted for the original constraints containing e_4. Thus, the complete set of constraints is given below:

$$\begin{array}{rcl} 0 & \leq & s_1 \\ s_2 + e_2 & \leq & 20 \\ 20 & \leq & s_3 \\ s_4 & \leq & 30 \\ s_1 + e_1 & \leq & s_2 \\ s_2 + e_2 & \leq & s_3 \\ s_3 + e_3 & \leq & s_4 \end{array} \qquad \begin{array}{rcl} s_2 - (s_1 + e_1) & \leq & 5 \\ 15 & \leq & s_3 + e_3 - (s_1 + e_1) \\ s_3 + e_3 - (s_1 + e_1) & \leq & 25 \\ 10 & \leq & s_4 - (s_2 + e_2) \\ s_4 - (s_2 + e_2) & \leq & 12 \\ s_4 - (s_3 + e_3) & \leq & 5 \end{array} \tag{3.12}$$

Next, an existentially quantified variable s_4, which is the innermost one, is eliminated. The constraints containing s_4 in the above constraint set are:

$$\begin{array}{rcl} s_2 + e_2 + 10 & \leq & s_4 \\ s_3 + e_3 & \leq & s_4 \end{array} \qquad \begin{array}{lcl} s_4 & \leq & s_2 + e_2 + 12 \\ s_4 & \leq & s_3 + e_3 + 5 \\ s_4 & \leq & 30 \end{array} \tag{3.13}$$

From these constraints, the parametric lower and upper bound functions are obtained as follows:

$$\begin{array}{lcl} \mathcal{F}_4^{min}(s_1, e_1, s_2, e_2, s_3, e_3) & = & \max(s_3 + e_3, s_2 + e_2 + 10) \\ \mathcal{F}_4^{max}(s_1, e_1, s_2, e_2, s_3, e_3) & = & \min(s_2 + e_2 + 12, s_3 + e_3 + 5, 30) \end{array}$$

And, as a result of eliminating s_4, the constraints in (3.13) are replaced by the following constraints:

$$\begin{array}{rcl} 10 & \le & 12 \\ s_2+e_2+10 & \le & s_3+e_3+5 \\ s_2+e_2+10 & \le & 30 \\ s_3+e_3 & \le & s_2+e_2+12 \\ 0 & \le & 5 \\ s_3+e_3 & \le & 30 \end{array} \quad \rightsquigarrow \quad \begin{array}{rcl} s_2+e_2 & \le & 20 \\ s_2+e_2+5 & \le & s_3+e_3 \\ s_3+e_3 & \le & s_2+e_2+12 \\ s_3+e_3 & \le & 30 \end{array} \tag{3.14}$$

If we continue this process until s_1 is eliminated, then we will obtain all the parametric bound functions, or the predicate will turn out to be false during the process. Figure 3.5 shows the obtained parametric bound functions.

0	$\le$	s_1	$\le$	2
$\max(8, s_1+e_1)$	$\le$	s_2	$\le$	$\min(10, s_1+e_1+5)$
$\max(20, s_1+e_1+10, s_2+e_2)$	$\le$	s_3	$\le$	$\min(22, s_1+e_1+17, s_2+e_2+4)$
$\max(s_3+e_3, s_2+e_2+10)$	$\le$	s_4	$\le$	$\min(30, s_2+e_2+12, s_3+e_3+5)$

Figure 3.5: Parametric Calendar for Example

3.3 Summary

We have formally defined the problem whose solution approach will be given in Chapter 4. The k-fold cyclically constrained job set was defined that allows standard constraints to be specified across the boundaries of two consecutive scheduling windows as well as within one scheduling window.

Also, prior works were presented in detail on scheduling with relative constraints, including the parametric scheduling scheme [23].

Chapter 4

Dynamic Dispatching of Cyclic Real-Time Tasks with Relative Constraints

In this chapter, we present an off-line algorithm to check the schedulability of a job set, $\Gamma^{1,\infty}$. And, if they are schedulable, the parametric lower and upper bound functions for each job start time are found in terms of the start or finish times of the previous jobs. These bounds can be evaluated at runtime within $O(N)$ time. Suppose that τ_i^j belongs to Γ^j, then the parametric lower and upper bound functions of s_i^j, are parameterized in terms of the start and finish times of already executed jobs in Γ^{j-1} and Γ^j. Another important result is that only N pairs of parametric bound functions have to be stored and cyclically used at runtime. The off-line algorithm has a pseudo-polynomial complexity $O(n^2N^3)$, where n is the number of jobs in one scheduling window that have relative constraints with jobs in the next scheduling window. If only jitter constraints on periodic tasks are allowed, it can be shown that the off-line and online components require $O(n^4N)$ and $O(n)$ times, respectively. Also, it is shown that, for a certain class of standard constraints, called *restricted standard constraints*, the off-line algorithm requires at most $O(N^3 + n^5)$.

The rest of this chapter is organized as follows. In Section 4.1, the parametric scheduling approach is developed by using the quantifier elimination techniques, and by transforming the constraint set into an equivalent constraint graph. In Section 4.2, example job sequences are given with parametric calendars found from the off-line algorithm. Finally, a summary of the chapter follows in Section 4.3.

4.1 Dynamic Cyclic Dispatching

As in the parametric scheduling approach developed for transaction scheduling [23], we want to devise a schedulability test and an efficient dispatching mechanism when an ∞-fold cyclically constrained job set, $\Gamma^{1,\infty}$, is given with its constraint matrices and vectors. We say $\Gamma^{1,k}$, is *schedulable* if there exists any method which can successfully dispatch the jobs in $\Gamma^{1,k}$.

Definition 4.1 (Schedulability of $\Gamma^{1,k}$) *The k-fold cyclically constrained job set $\Gamma^{1,k}$ $(1 \le k)$ is schedulable if the following predicate holds:*

$$\begin{aligned} sched^{1,k} \;\equiv\;& \exists s_1^1 :: \forall e_1^1 \in [l_1^1, u_1^1] :: \exists s_2^1 :: \forall e_2^1 \in [l_2^1, u_2^1] :: \ldots \\ & \exists s_N^k :: \forall e_N^k \in [l_N^k, u_N^k] :: \mathcal{C}^{1,k} \end{aligned} \tag{4.1}$$

where $\mathcal{C}^{1,k}$ is a set of standard constraints defined on $\{s_1^1, e_1^1, \ldots, s_N^k, e_N^k\}$.

Then, the following proposition holds for all $k \geq 1$.

Proposition 4.1

$$\forall k \geq 1 :: sched^{1,k+1} \Longrightarrow sched^{1,k}$$

Proof: Obvious from the definition of a cyclically constrained job set and from the definition of $sched^{1,k}$ in (4.1). ■

Hence, once $sched^{1,k}$ turns out to be **False**, then all $sched^{1,j}$, $k \leq j$, are **False**, too. By this proposition, the schedulability of $\Gamma^{1,\infty}$ is defined.

Definition 4.2 (Schedulability of $\Gamma^{1,\infty}$) *$\Gamma^{1,\infty}$ is schedulable if and only if*

$$\lim_{k \to \infty} sched^{1,k} = \textbf{True}$$

In [23], it is shown that checking Predicate (3.9) is not trivial because of the nondeterministic job execution times and because of the existence of standard relative constraints among the jobs. This applies to the above $sched^{1,k}$ predicate, too. The variable elimination techniques are used in [23] to eliminate variables from Predicate (3.9). At the end of the variable elimination process parametric bound functions for s_i, that are parameterized in terms of the variables in $\{s_1, e_1, \ldots, e_{i-1}\}$, are found as well as the predicate value.

However, if we want to apply the variable elimination technique to $sched^{1,k}$, the following problems have to be addressed first:

1. On which subset of $\{s_1^1, e_1^1, \ldots, s_{i-1}^j, e_{i-1}^j\}$ does the parametric bound functions for s_i^j depend?
2. Is it required to store parametric bound functions for every job in $\Gamma^{1,k}$?
3. What parametric bound functions have to be used if k is not known at pre-runtime and dynamically decided at runtime?

Let $\mathcal{F}_{s_i^j}^{min,k}$ and $\mathcal{F}_{s_i^j}^{max,k}$ denote parametric lower and upper bound functions for s_i^j, respectively, that are found after the variable elimination algorithms are applied to $sched^{1,k}$. If the number of variables is unbounded with which $\mathcal{F}_{s_i^j}^{min,k}$ or $\mathcal{F}_{s_i^j}^{max,k}$ is parameterized, then it is not possible to evaluate them at run-time within bounded computation times. Also, if it is required that parametric bound functions for every job in $\Gamma^{1,k}$ be stored at runtime, the scheme is not implementable for large k because of memory requirements. Finally, if the value of k is not known at pre-runtime and is decided dynamically at runtime, which is true in most real-time applications, the parametric bound functions to be used have to be selected.

In this section, the answers to the above questions are sought by first transforming $sched^{1,k}$ into a constraint graph and by investigating the properties of such graphs. In section 4.1.1 the transformation rule is presented with lemmas showing the equivalence relationship between $sched^{1,k}$ and its constraint graph with respect to variable elimination process. In section 4.1.2 several terminologies are defined for constraint graphs, and in section 4.1.3 the properties of constraint graphs are investigated. Then, in section 4.1.4 a complete off-line algorithm is presented to check $sched^{1,\infty}$ and to obtain parametric bound functions for job start times if it is schedulable. In addition, for a certain class of standard constraints, it is shown in section 4.1.5 that the off-line algorithm can be executed within $O(N^3 + n^5)$ time by pre-eliminating certain nodes from the constraint graph.

4.1.1 Transforming a Constraint Set into a Constraint Graph

We want to apply the variable elimination algorithms to $sched^{1,k}$ for some fixed k, and want to find out answers to the previously raised three questions. For that purpose, we first transform the predicate into a constraint graph and apply node elimination algorithms(corresponding to the variable elimination algorithms) to the graph. Then, the properties of the constraint graphs created during the node elimination process are examined. Working on constraint graphs, instead of constraint sets themselves, makes it easier to infer and prove useful properties. In this section, the transformation rules are given for a set of jobs and its associated constraint set.

Let $\Pi = \{\tau_1, \tau_2, \ldots, \tau_N\}$ be a finite set of jobs with a set of standard constraints, $\mathcal{C}$. Consider eliminating quantified variables from the following predicate:

$$Sched \;\equiv\; \exists s_1 :: \forall e_1 \in [l_1, u_1] :: \ldots \exists s_N :: \forall e_N \in [l_N, u_N] :: \mathcal{C}$$

Then, predicates on subsets of $\{s_1, e_1, \ldots, s_N, e_N\}$ are defined next that are found after eliminating variables.

Definition 4.3 *$Sched(s_a)$($1 \le a \le N$) is defined to be a predicate on a set of variables $\{s_1, e_1, \ldots, s_a\}$ that are found after eliminating variables of $< f_N, s_N, \ldots, f_a >$ from $Sched$. $Sched(e_a)$ is defined similarly.*

That is, $Sched(s_a)$ can be expressed as

$$Sched(s_a) \;\equiv\; \exists s_1 :: \forall e_1 \in [l_1, u_1] :: \ldots \exists s_a :: \mathcal{C}(s_a)$$

It will be shown that $Sched$(or $Sched(s_a)$, or $Sched(e_a)$) can be transformed into a directed graph, which is called a *constraint graph*, such that the variable elimination process can be mapped into a corresponding node elimination operation in the graph. Note that, in the following definition of a constraint graph, *semi-exclusive-ORed* edges are defined, which will be used in defining *restricted paths* in constraint graphs. Also, $v_1 \xrightarrow{\text{w}} v_2$ denotes an edge from a node v_1 to a node v_2 with a weight w, and $< v_1 \xrightarrow{\text{w}_1} v_2 \xrightarrow{\text{w}_2} \ldots \xrightarrow{\text{w}_{i-1}} v_i >$ denotes a path from a node v_1 to a node v_i with a weight sum $w = \sum_{j=1}^{i-1} w_j$. $v_1 \rightsquigarrow v_i$ denotes that there exists a path from v_1 to v_i, and $v_1 \overset{\text{w}}{\rightsquigarrow} v_i$ denotes that there exists a path from v_1 to v_2 whose weight sum is w.

The following rule is used to transform a predicate into a constraint graph. Here, the *semi-exclusive-ORed* edges denote a pair of edges that cannot be arbitrarily placed in a *restricted path* that will be defined in Definition 4.6.

Definition 4.4 (Constraint Graph) *A constraint graph $G(V, E)$ is found from $Sched$ (or $Sched(s_a)$, or $Sched(e_a)$) as follows:*

1. *node set V is obtained as follows:*
   - $v_0 \in V$
   - *s_i, $f_i \in V$ for $1 \le i \le N$ where $f_i = s_i + e_i$.*
2. *edge set E is obtained as follows:*
   - *For each tuple $< s_i, f_i >$, add the following* semi-exclusive-ORed *edges to E:*

(a) $s_i \xrightarrow{\mathrm{l_i}} f_i$

(b) $f_i \xrightarrow{\mathrm{-u_i}} s_i$

- *For each constraint in $\mathcal{C}$ that can be converted to:*

  (a) $v_i - v_j \le c$ $(v_i, v_j \in \{s_i, f_i \mid 1 \le i \le N\})$: *add* $v_j \xrightarrow{\mathrm{c}} v_i$ *to* E.

  (b) $v_i \le c$: *add* $v_0 \xrightarrow{\mathrm{c}} v_i$ *to* E.

  (c) $-v_i \le c$: *add* $v_i \xrightarrow{\mathrm{c}} v_0$ *to* E.

Definition 4.5 *The constraint graph found from $Sched(s_a)$ is denoted as $G(s_a)$.*[1] *Similarly, $G(f_a)$ represents a graph found from $Sched(e_a)$.*

Figure 4.1 shows a graph created from the example job set $\Gamma^{1,2}$ defined in Example 3.1. Note that v_0 is an extra node created to represent a constant 0 that is used to specify absolute constraints such as the release time and the deadline constraints. In the figure, the edges connected by $\oplus$ are semi-exclusive-ORed edges.

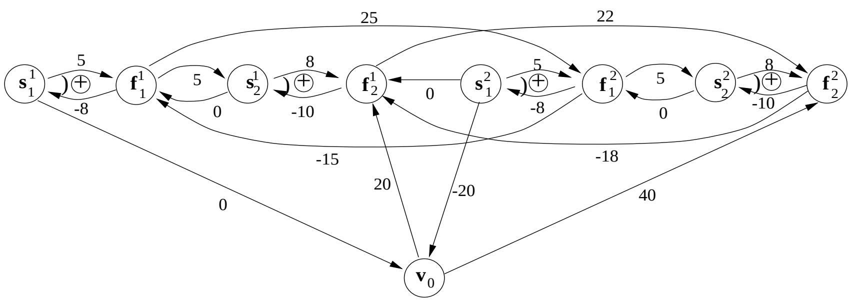

Figure 4.1: Constraint Graph for $\Gamma^{1,2}$

Note that there may exist only one edge from one node to another from the uniqueness of inequality in the constraint set. For example, if there are two constraints $v_1 - v_2 \le c_1$ and $v_1 - v_2 \le c_2$ in $\mathcal{C}$, then one of them is redundant. Therefore, we can denote an edge from v_1 to v_2 in a constraint graph as $v_1 \rightarrow v_2$ without its weight specified. Also, note that any edge from f_i to s_i is semi-exclusive-ORed to any edge from s_i to f_i. That is, even if any of these two edges is created from another constraint in $\mathcal{C}$ rather than from the minimum or maximum execution time constraint, they are semi-exclusive-ORed.

Definition 4.6 (Restricted Path) *In a constraint graph, a path,* $< v_1 \xrightarrow{\mathrm{w_1}} v_2 \xrightarrow{\mathrm{w_2}} \ldots v_{i-1} \xrightarrow{\mathrm{w_{i-1}}} v_i >$, *is called a* restricted path *from v_1 to v_i if the following is satisfied:*

- *If $f_j \rightarrow s_j$ appears in the path, then its semi-exclusive-ORed edge $s_j \rightarrow f_j$ may appear at most once in the path, and vice versa.*
- *If two semi-exclusive-ORed edges, $f_j \rightarrow s_j$ and $s_j \rightarrow f_j$, appear in the path, then they belong to a sub-path $< f_j \rightarrow s_j \rightarrow f_j >$.*

[1]The full notation would be $G(s_a)(V, E)$. But, if no confusion is caused, $G(s_a)$ will be used in this chapter.

Note that if a sub-path $< f_j \to s_j \to f_j >$ appears once in the path, then neither $f_j \to s_j$ nor $s_j \to f_j$ should appear at another place in the path, and vice versa.

Definition 4.7 (Restricted Cycle) *A* restricted cycle *in a constraint graph is defined to be a cycle*[2] *such that*

1. *it satisfies the definition of a restricted path.*
2. *it is not a sub-path of* $< f_j \to s_j \to f_j >$ *for any* $1 \le j \le N$*.*

For example, a path $< f_j \to s_j \to f_j \to s_l \to f_j >$ is a restricted cycle while a path $< f_j \to s_j \to f_j >$ is not. Also, a restricted path without any restricted cycle in it is called an *acyclic restricted path.*

The elimination algorithm of a node f_a from a graph $G(f_a)$ is presented next.

Algorithm 4.1 (Elimination of f_a from a Graph $G(f_a)$) *Elimination of* f_a *from* $G(f_a)$ *is performed by the following algorithm.*

1. *For each edge pair,* $< y \xrightarrow{w_1} f_a, f_a \xrightarrow{w_2} s_a >$*, that are not semi-exclusive-ORed in* $G(f_a)$*:*
   - *create an edge* $y \xrightarrow{w_1+w_2} s_a$*.*
     (a) *If* $y = s_a$ *and* $w_1 + w_2 < 0$*, then return* **False**.[3]
     (b) *If* $y = s_a$ *and* $w_1 + w_2 \ge 0$*, then remove this edge.*[4]
     (c) *If there already exists an edge* $y \xrightarrow{w'} s_a$ *before creating* $y \xrightarrow{w_1+w_2} s_a$*, then the edge with less weight remains, while the other is removed.*
2. *For each edge pair,* $< s_a \xrightarrow{w_1} f_a, f_a \xrightarrow{w_2} z >$*,* $z \ne s_a$*, that are not semi-exclusive-ORed in* $G(f_a)$*:*
   - *create an edge* $s_a \xrightarrow{w_1+w_2} z$*.*
     (a) *If there already exists an edge* $s_a \xrightarrow{w''} z$ *before creating* $s_a \xrightarrow{w_1+w_2} z$*, then the edge with less weight remains, while the other is removed.*
3. *Set* $V = V - \{f_a\}$ *and remove all edges to or from* f_a *in* $G(f_a)$*.*

Let $Elim(G(f_a), f_a)$ denote a new graph created after eliminating f_a from the graph $G(f_a)$ according to Algorithm 4.1 in case **False** is not found. In this case, the following lemma proves the equivalence, with regards to the graph transformation rule, between the elimination of an universal quantifier from the predicate and the elimination of a node, f_a, from the constraint graph.

Lemma 4.1 $Elim(G(f_a), f_a)$ *is equal to* $G(s_a)$*.*

---

[2]A cycle is defined to be a path $< y \to v_1 \ldots \to v_i \to y >$ where $i \ge 1$, or to be a path $< y \to y >$.

[3]This is because $y - y = 0 \le w_1 + w_2 < 0$ is a contradiction.

[4]This is because $y - y = 0 \le w_1 + w_2$ is a tautology.

Proof: Given in appendix.

Next, we show how a node corresponding to an existential quantifier s_a may be eliminated from the graph $G(s_a)$.

Algorithm 4.2 (Elimination of s_a from a Graph $G(s_a)$) *Elimination of s_a from $G(s_a)$ is performed by the following algorithm.*

1. *For each edge pair,* $< y \xrightarrow{w_1} s_a, s_a \xrightarrow{w_2} z >$*, in* $G(s_a)$*:*
   - *create an edge* $y \xrightarrow{w_1+w_2} z$*.*
     (a) *If* $y = z$ *and* $w_1 + w_2 < 0$*, then return* **False**.
     (b) *If* $y = z$ *and* $w_1 + w_2 \geq 0$*, then remove this edge.*
     (c) *If there already exists an edge* $y \xrightarrow{w'} z$ *before creating* $y \xrightarrow{w_1+w_2} z$*, then the edge with less weight remains, while the other is removed.*
2. *Set* $V = V - \{s_a\}$ *and remove all edges to or from* s_a *in* $G(s_a)$*.*

Similarly, let $Elim(G(s_a), s_a)$ denote a new graph created after eliminating s_a from the graph $G(s_a)$ according to Algorithm 4.2 in case **False** is not found. Then, the following lemma shows the equivalence between the elimination of a node in the graph and the elimination of an existential quantifier from the constraint set.

Lemma 4.2 *$Elim(G(s_a), s_a)$ is equal to $G(f_{a-1})$.*

Proof: Given in appendix.

By inductively applying Lemma 4.1 and 4.2, the equivalence relationship between node elimination and variable elimination processes can be established. This relationship is shown in Figure 4.2 with respect to the constraint graph derivation rules.

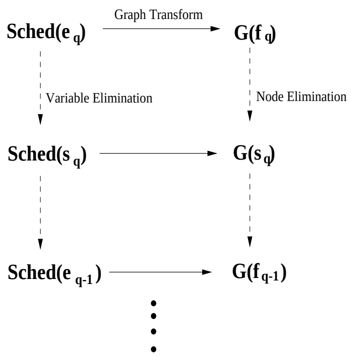

Figure 4.2: Equivalence between Predicates and Graphs

The elimination process of nodes, f_a and s_a, from the graph $G(f_a)$ can be viewed as preserving the connectivity between any two nodes in $\{v_0, s_1, f_1, \ldots, s_{a-1}, f_{a-1}\}$ through f_a and s_a in $G(f_a)$. That is, if there exists any restricted path from y to z only through s_a and f_a in $G(f_a)$, then a new edge from y to z is created to maintain the connectivity from y to z even after f_a and s_a are eliminated. This is formally proved in Lemma 4.3.

Figure 4.3 shows a graph and its node elimination processes for $sched^{1,2}$ that is derived from $\Gamma^{1,2}$ in Example 3.1.

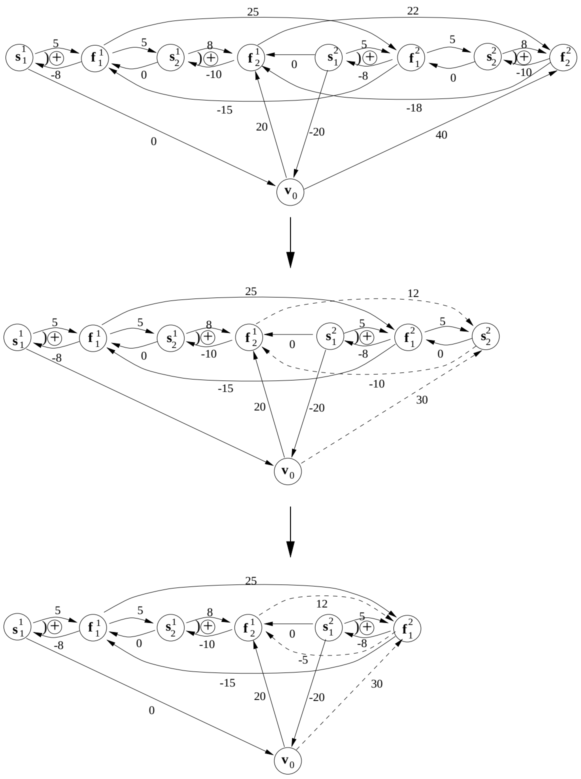

Figure 4.3: Elimination of f_2^2 and s_2^2 from $\Gamma^{1,2}$

The following proposition describes a necessary condition for $Sched$ to be true in terms of its constraint graph.

Proposition 4.2 *If a constraint graph for $Sched$ has a negative weight restricted cycle, then $Sched =$* **False**.

Proof: Given in appendix.

The following lemma shows how the connectivity is maintained during the node elimination process, which is quite an useful property that will be frequently used throughout this chapter.

Lemma 4.3 *Let $\{v_0, s_1, f_1, s_2, f_2, \ldots, s_{a-1}, f_{a-1}, s_a, f_a\}$, $1 \leq a \leq N$, denote a node set of $G(f_a)$ that is found after eliminating nodes of $< f_N,\ s_N,\ f_{N-1},\ s_{N-1},\ \ldots,\ f_{a+2},\ s_{a+2},\ f_{a+1},\ s_{a+1} >$ from $G(f_N)$. Also, assume that no contradiction has been found yet. Then, the following two conditions are equivalent:*

1. $y \xrightarrow{\mathrm{w}} z \in G(f_a)$

2. *there exists a minimum weight acyclic restricted path* $y \overset{w}{\rightsquigarrow} z$ *in* $G(f_N)$ *where all intermediate*[5] *nodes of the path belong to* $\{s_{a+1}, f_{a+1}, \ldots, s_N, f_N\}$.[6]

Proof: Given in appendix.

In the next corollary it is assumed that v and v' denote any two nodes that are located consecutively in a sequence $< v_0, s_1, f_1, \ldots s_N, f_N >$.

Corollary 4.1 *Let* $\{v_0, s_1, f_1, \ldots, v\}$, *denote a node set of* $G(v)$ *that is found after eliminating nodes of* $< f_N, s_N, \ldots, v' >$ *from* $G(f_N)$. *Also, assume that no contradiction has been found yet. If an edge from* y *to* z *exists in* $G(v)$, *then there exists a path from* y *to* z *in* $G(f_N)$ *whose intermediate nodes belong to* $\{v', \ldots, s_N, f_N\}$.

Proof: Given in appendix.

For example, in the example shown in figure 4.3, after eliminating $\{f_2^2, s_2^2\}$ an edge $f_2^1 \xrightarrow{12} f_1^2$ is created since, in the initial graph, there exists a minimum weight acyclic restricted path $< f_2^1 \xrightarrow{22} f_2^2 \xrightarrow{-10} s_2^2 \xrightarrow{0} f_1^2 >$ whose weight sum is 12 and whose intermediate nodes belong to $\{s_2^2, f_2^2\}$. Also, an edge $f_2^1 \xrightarrow{2} f_2^1$ is created in $G^{1,2}(f_1^2)$, since there exists a minimum weight restricted path $< f_2^1 \xrightarrow{22} f_2^2 \xrightarrow{-10} s_2^2 \xrightarrow{8} f_2^2 \xrightarrow{-18} f_2^1 >$ without any intermediate restricted cycle.

4.1.2 Definitions for Constraint Graphs

In this section, we define several terms for constraint graphs. They will be used in deriving and proving the properties of constraint graphs in the next section. In this section, it is assumed that an initial graph is obtained from the predicate $sched^{1,k}$ that is defined in (4.1) for $\Gamma^{1,k}$.

Before defining terminologies for constraint graphs, the following function is defined on node sets of constraint graphs.

Definition 4.8 (g_γ) g_γ *is an one-to-one mapping*

$$\{s_i^j, f_i^j \mid 1 \le i \le N \wedge \max(1, -\gamma + 1) \le j\}$$

$$\longrightarrow \{s_i^j, f_i^j \mid 1 \le i \le N \wedge \max(\gamma + 1, 1) \le j\}$$

by the following rule:

$$g_\gamma(v) = \begin{cases} v_0 & \textit{if } v = v_0. \\ s_i^{j+\gamma} & \textit{if } v = s_i^j \textit{ where } 1 \le i \le N. \\ f_i^{j+\gamma} & \textit{if } v = f_i^j \textit{ where } 1 \le i \le N. \end{cases}$$

$g_\gamma(V)$ *on a node set* V *is defined to be a set of* $g_\gamma(v)$ *where* v *is an element of* V.

[5] $\{v_1, v_2, \ldots, v_i\}$ is a set of intermediate nodes of a path $< y \rightarrow v_1 \rightarrow v_2 \ldots v_i \rightarrow z >$ where $i \ge 1$, or $\{\}$ is an intermediate node set if the path consists of one edge.

[6] $y \rightarrow z$ may also be considered as a path whose intermediate nodes belong to $\{s_{a+1}, f_{a+1}, \ldots, s_N, f_N\}$.

For example, $s_i^{j_1}$ in Γ^{j_1} can be related to a node $s_i^{j_2}$ in Γ^{j_2} by

$$s_i^{j_2} = g_{(j_2-j_1)}(s_i^{j_1})$$

In this case $s_i^{j_2}$ is called a *corresponding node* of $s_i^{j_1}$ in a job set Γ^{j_2}, and vice versa.

As in Definition 4.3, $sched^{1,k}(s_i^j)$ $(1 \leq i \leq N \wedge 1 \leq j \leq k)$ is defined to be a predicate on a set of variables $\{s_1^1, e_1^1, \ldots, s_i^j\}$ that is obtained after eliminating the variables, e_N^k, s_N^k, ..., e_i^j, from $sched^{1,k}$. That is, it can be expressed as

$$sched^{1,k}(s_i^j) \equiv \exists s_1^1 :: \forall e_1^1 \in [l_1^1, u_1^1] :: \ldots \exists s_i^j :: \mathcal{C}^{1,k}(s_i^j)$$

where $\mathcal{C}^{1,k}(s_i^j)$ is a set of standard constraints obtained after variable elimination. $sched^{1,k}(e_i^j)$ is defined similarly. Also, as in Definition 4.5, the graphs found from the above predicates are denoted as follows:

- $G^{1,k}(s_i^j)$ denotes a graph constructed from $sched^{1,k}(s_i^j)$.
- $G^{1,k}(f_i^j)$ denotes a graph constructed from $sched^{1,k}(e_i^j)$.

Note that, from $\mathcal{C}^{1,k}(s_i^j)$(or $G^{1,k}(s_i^j)$), we can find out the parametric lower and upper bound functions for s_i^j in the forms presented in Proposition 3.1.

First, several terms are defined for constraint graphs. Let $\mathcal{E}$ denote a subset of edges in a graph $G^{1,k}(s_i^j)$, (or $G^{1,k}(f_i^j)$) in the following two definitions.

Definition 4.9 (Node Set from $\mathcal{E}$) *$Node(\mathcal{E})$ denotes a set of nodes that are connected by any edge in $\mathcal{E}$.*

Definition 4.10 (Preceding Node Set from $\mathcal{E}$) *$PrecNode(\mathcal{E})$ is defined to be a subset of $Node(\mathcal{E})$ in the graph such that $v \in PrecNode(\mathcal{E})$ if and only if*

- *there exists a node v' that lies after v in the sequence $< v_0, s_1^1, f_1^1, \ldots, s_i^j(, f_i^j) >$ satisfying:*

$$v \rightarrow v' \in \mathcal{E} \ \vee \ v' \rightarrow v \in \mathcal{E}$$

In the example constraint graph shown in Figure 4.1 let $\mathcal{E}$ be $\{f_2^1 \rightarrow f_2^2,\ s_2^2 \rightarrow f_2^2,\ v_0 \rightarrow f_2^2\}$. Then, a node set from $\mathcal{E}$, $Node(\mathcal{E})$ is found to be $\{v_0, f_2^1, s_2^2, f_2^2\}$. Also, the preceding node set, $PrecNode(\mathcal{E})$, is $\{v_0, f_2^1, s_2^2\}$.

In the following definition, let y and z denote any two consecutive nodes in the sequence $< v_0, s_1^1, f_1^1, s_2^1, f_2^1, \ldots, s_N^1, f_N^1, s_1^2, f_1^2, \ldots, \ldots, s_1^k, f_1^k, \ldots, s_N^k, f_N^k >$.

Definition 4.11 (Crossing Edge Set over a Node y) *A crossing edge set $\Phi^{1,k}(y)$ is defined to be a set of edges $v_1 \rightarrow v_2$ in $G^{1,k}(f_N^k)$ satisfying either of the following two conditions:*

1. *$v_1 \in < v_0, s_1^1, f_1^1, \ldots, y >$ and $v_2 \in < z, \ldots, s_N^k, f_N^k >$.*
2. *$v_2 \in < v_0, s_1^1, f_1^1, \ldots, y >$ and $v_1 \in < z, \ldots, s_N^k, f_N^k >$.*

For example, in Figure 4.4, $\Phi^{1,2}(f_2^1)$ is shown with dashed edges. Informally speaking, any edges created in $G^{1,k}(y)$ after eliminating nodes $< z, \ldots, s_N^k, f_N^k >$ may connect only the nodes that belong to $PrecNode(\Phi^{1,k}(y))$. This is proved in Proposition 4.4.

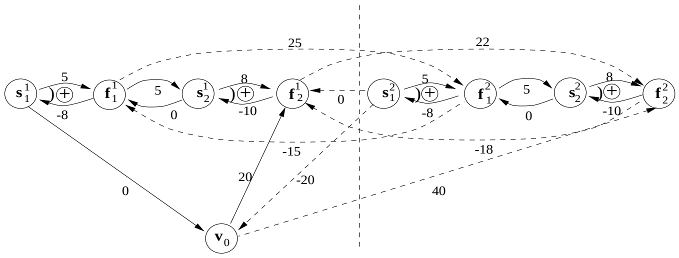

Figure 4.4: $\Phi^{1,2}(f_2^1)$ is denoted as dashed edges meeting with a vertical line.

Definition 4.12 (Created Edge Set in $G^{1,k}(f_i^j)$) *A created edge set $\Psi^{1,k}(f_i^j)$, $1 \le j \le k-1$, is defined to be a set of edges $v_1 \xrightarrow{w} v_2$ in $G^{1,k}(f_i^j)$ where v_1, v_2 satisfy the following condition:*

- *there exists a path $v_1 \rightsquigarrow v_2$ in $G^{1,k}(f_N^k)$ such that*
  1. *it has at least one intermediate node.*
  2. *all of its intermediate nodes belong to $\{v_0, s_1^1, f_1^1, \ldots, s_N^k, f_N^k\} - \{v_0, s_1^1, f_1^1, \ldots, f_i^j\}$.*

$\Psi^{1,k}(s_i^j)$ is defined similarly.

That is, a created edge set in $G^{1,k}(f_i^j)$ contains edges that could be created during the variable elimination process. Note that, if a newly created edge is implied by an already existing edge in $G^{1,k}(f_N^k)$ with a less weight and thus removed during the elimination process as explained in Algorithm 4.1 and 4.2, then the already existing edge is included into the created edge set instead of the removed one that is actually created during the variable elimination process. In figure 4.5, the constraint graph is shown corresponding to Example 3.1. Dashed edges are used to represent $\Psi^{1,3}(s_2^3)$ and $\Psi^{1,3}(s_2^2)$.

Next, the semi-homogeneity and homogeneity relationships are defined between two edge sets in two constraint graphs that are found during variable elimination processes from two job sets, $\Gamma^{1,k}$ and $\Gamma^{1,l}(k \le l)$, respectively.

Definition 4.13 (Semi-homogeneous Edge Sets) *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be subsets of edges in $G^{1,k}(f_i^{j_1})$ and $G^{1,l}(f_i^{j_2})$ (or, $G^{1,k}(s_i^{j_1})$ and $G^{1,l}(s_i^{j_2})$), respectively, where $k \le l \wedge j_1 \le k \wedge j_2 \le l$. Then, $\mathcal{E}_1$ is semi-homogeneous to $\mathcal{E}_2$ if and only if*

$$|\mathcal{E}_1| = |\mathcal{E}_2| \quad \wedge \quad [(v_1 \to v_2 \ \in \mathcal{E}_1) \implies (g_{(j_2-j_1)}(v_1) \to g_{(j_2-j_1)}(v_2) \ \in \mathcal{E}_2)]$$

Here, note that, if $\mathcal{E}_1$ is semi-homogeneous to $\mathcal{E}_2$, then

$$(v_3 \to v_4 \ \in \mathcal{E}_2) \implies (g_{(j_1-j_2)}(v_3) \to g_{(j_1-j_2)}(v_4) \in \mathcal{E}_1)$$

holds, too, since $|\mathcal{E}_1| = |\mathcal{E}_2|$ and $\mathcal{E}_1$ is mapped onto $\mathcal{E}_2$ under the index function $g_{(j_2-j_1)}$ which is one-to-one.

The homogeneity relationship is defined next which is stronger than semi- homogeneity relationship. Again, let $\mathcal{E}_1$ and $\mathcal{E}_2$ be subsets of edges in $G^{1,k}(f_i^{j_1})$ and $G^{1,l}(f_i^{j_2})$ (or, $G^{1,k}(s_i^{j_1})$ and $G^{1,l}(s_i^{j_2})$), respectively, where $k \le l \wedge j_1 \le k \wedge j_2 \le l$.

Definition 4.14 (Homogeneous Edge Sets) *$\mathcal{E}_1$ is homogeneous to $\mathcal{E}_2$, denoted as $\mathcal{E}_1 \sim \mathcal{E}_2$, if and only if*

1. *$\mathcal{E}_1$ and $\mathcal{E}_2$ are semi-homogeneous.*
2. *For two nodes $v_1(\neq v_0)$, $v_2(\neq v_0)$,*
   $(v_1 \xrightarrow{\mathrm{w}} v_2 \ \in \mathcal{E}_1) \iff (g_{(j_2-j_1)}(v_1) \xrightarrow{\mathrm{w}} g_{(j_2-j_1)}(v_2) \ \in \mathcal{E}_2)$
3. *For two nodes v_0, v_2, where $v_2 \neq v_0$,*
   $(v_0 \xrightarrow{\mathrm{w}} v_2 \ \in \mathcal{E}_1) \iff (v_0 \xrightarrow{\mathrm{w+(j_2-j_1)L}} g_{(j_2-j_1)}(v_2) \ \in \mathcal{E}_2)$
4. *For two nodes v_1, v_0, where $v_1 \neq v_0$,*
   $(v_1 \xrightarrow{\mathrm{w}} v_0 \ \in \mathcal{E}_1) \iff (g_{(j_2-j_1)}(v_1) \xrightarrow{\mathrm{w-(j_2-j_1)L}} v_0 \ \in \mathcal{E}_2)$

Homogeneity relations are commutative and transitive, i.e.,

$$\mathcal{E}_1 \sim \mathcal{E}_2 \iff \mathcal{E}_2 \sim \mathcal{E}_1$$

$$(\mathcal{E}_1 \sim \mathcal{E}_2) \wedge (\mathcal{E}_2 \sim \mathcal{E}_3) \implies \mathcal{E}_1 \sim \mathcal{E}_3$$

which can be easily proved from the definition of homogeneity. Two homogeneous created edge sets, $\Psi^{1,3}(s_2^3)$ and $\Psi^{1,3}(s_2^2)$, are shown in Figure 4.5 with dashed edges where $L = 20$.

A constant, n, is defined next that will be used in obtaining a complexity bound of our algorithm.

Definition 4.15 (n) $n = \mid PrecNode(\Phi^{1,k}(f_N^1)) \mid, \quad k \geq 2$

Note that $\mid PrecNode(\Phi^{1,k}(f_N^j)) \mid$ is same for all $2 \leq k$ and all $1 \leq j \leq k-1$ from the definition of a cyclically constrained job set and the definition of a preceding node set. $n-1$ is the number of jobs in one scheduling window that have standard relative constraints with jobs in the next scheduling window.

4.1.3 Characteristics of Constraint Graphs

From now on, the properties of constraint graphs will be examined that remain true during the node elimination process. Note that, from Proposition 4.2 if a negative weight restricted cycle exists in the constraint graph, Algorithm 4.1 or 4.2 will detect it and return **False**. In this case the predicate $sched^{1,k}$ is false and the job set $\Gamma^{1,\infty}$ is not schedulable as well as $\Gamma^{1,k}$. If a constraint graph appears in any of the following propositions, it is assumed that no contradiction has been found in the process of obtaining that graph. First, it is shown that the parametric bound functions for s_i^j found from a constraint graph $G^{1,k}(s_i^j)$ depend on the start or finish times of the jobs in Γ^{j-1} and Γ^j that are already executed. This means that the number of jobs it may actually be dependent on is shown to be bounded by $O(N)$. This bounds the number of variables to $O(N)$ that have to be used in evaluating parametric bound functions at runtime.

Proposition 4.3 *In a graph $G^{1,k}(s_i^j)$, if s_i^j is connected to a node v, then*

$$v \ \in \ PrecNode(\Phi^{1,k}(s_i^j)) \ \cup \ P$$

where $P = \{y \mid \ y \in <v_0, s_1^{j-1}, f_1^{j-1}, \ldots, f_{i-1}^j> \ \wedge (y \rightarrow s_i^j \in G^{1,k}(f_N^k) \ \vee \ s_i^j \rightarrow y \in G^{1,k}(f_N^k))\}$

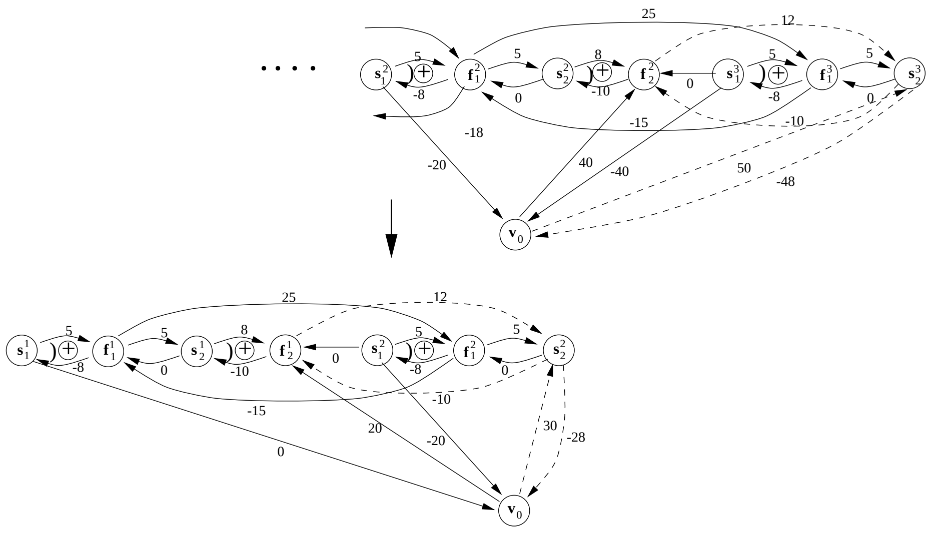

Figure 4.5: Homogeneous edge sets, $\Psi^{1,3}(s_2^3)$ and $\Psi^{1,3}(s_2^2)$

Proof: Given in appendix.

Similar result holds for a graph $G^{1,k}(f_i^j)$.

Then, the following proposition implies that the set of nodes, to which additionally created edges in $G^{1,k}(s_i^j)$(or $G^{1,k}(f_i^j)$) may be connected, is a subset of the set, $PrecNode(\Phi^{1,k}(s_i^j))$(or $PrecNode(\Phi^{1,k}(f_i^j))$).

Proposition 4.4

$$Node(\Psi^{1,k}(s_i^j)) \subseteq PrecNode(\Phi^{1,k}(s_i^j))$$

$$Node(\Psi^{1,k}(f_i^j)) \subseteq PrecNode(\Phi^{1,k}(f_i^j))$$

Also, $\mid \Psi^{1,k}(f_N^j) \mid \leq \; n(n-1)$ *holds.*

Proof: Given in appendix.

These two propositions give an upper bound on the actual number of nodes s_i^j may be connected to in $G^{1,k}(s_i^j)$, which is $O(N)$. If only jitter constraints are allowed from periodic tasks, it is easy to see that s_i^j in $G^{1,k}(s_i^j)$ is connected to at most $O(n)$ number of jobs. This is because $\mid PrecNode(\Phi^{1,k}(s_i^j)) \mid \leq n$ and $\mid P \mid \leq 2$.

Then, an interesting property of an additionally created edge set, $\Psi^{1,k}(f_N^j)$, is given in the following proposition. After eliminating $4N$ variables of the last $2N$ jobs (belonging to Γ^k and Γ^{k-1}) from $sched^{1,k}$, we periodically obtain semi-homogeneous created edge sets once eliminating each $2N$ variables for Γ^j, $2 \leq j \leq k-2$.

Proposition 4.5 *An edge set* $\Psi^{1,k}(f_N^j)$ *is semi-homogeneous to* $\Psi^{1,k}(f_N^{j-1})$ *for* $2 \leq j \leq k-2$.

Proof: Given in appendix.

In addition, the edge weight change patterns between two semi-homogeneous edge sets, $\Psi^{1,k}(f_N^{j-1})$ and $\Psi^{1,k}(f_N^j)$, are presented in the following proposition.

Proposition 4.6 *Consider two semi-homogeneous created edge sets,* $\Psi^{1,k}(f_N^{j-1})$ *and* $\Psi^{1,k}(f_N^j)$*, where* $2 \leq j \leq k-2$*. Suppose* $v_1 \xrightarrow{\text{w}} v_2 \;\in \Psi^{1,k}(f_N^j)$ *and* $g_{(-1)}(v_1) \xrightarrow{\text{w}'} g_{(-1)}(v_2) \;\in \Psi^{1,k}(f_N^{j-1})$*. Then, the following is satisfied:*

1. *If* $v_1 \neq v_0$ *and* $v_2 \neq v_0$*,* $\quad w' \leq w$
2. *If* $v_1 = v_0$ *and* $v_2 \neq v_0$*,* $\quad w' \leq w - L$
3. *If* $v_1 \neq v_0$ *and* $v_2 = v_0$*,* $\quad w' \leq w + L$

Proof: Given in appendix.

Once we find two homogeneous created edge sets, $\Psi^{1,k}(f_N^j)$ and $\Psi^{1,k}(f_N^{j-1})$ for some j, then the following proposition enables us to stop the variable elimination process, since homogeneous created edge sets will be found to the ones already obtained, if the node elimination process continues.

Proposition 4.7 *If an edge set* $\Psi^{1,k}(f_N^j)$ *is homogeneous to an edge set* $\Psi^{1,k}(f_N^{j-1})$*, where* $2 \leq j \leq k-2$*, then*

$$\forall l : 2 \leq l \leq j-1 \;::\; \forall i : 1 \leq i \leq N \;::\; \Psi^{1,k}(f_i^l) \sim \Psi^{1,k}(f_i^j) \;\wedge\; \Psi^{1,k}(s_i^l) \sim \Psi^{1,k}(s_i^j)$$

Proof: Given in appendix.

More generalized result is presented next which holds whenever two homogeneous edge sets, $\Psi^{1,k}(f_N^j)$ and $\Psi^{1,k}(f_N^{j-1})$, are found during the variable elimination process.

Proposition 4.8

$$\Psi^{1,k}(f_N^{j-1}) \sim \Psi^{1,k}(f_N^j) \implies (\forall i : 1 \leq i :: \Psi^{1,k+i}(f_N^{(j-1)+i}) \sim \Psi^{1,k+i}(f_N^{j+i}))$$

Proof: This is obvious from the cyclic structures of constraint graphs, $G^{1,k}(f_N^k)$ and $G^{1,k+i}(f_N^{k+i})$, and from Proposition 4.6 and 4.7. ■

From the definition of homogeneity between edge sets in constraint graphs, the following proposition is derived.

Proposition 4.9 *Suppose* $\Psi^{1,k_1}(s_i^j) \sim \Psi^{1,k_2}(s_i^l)$ *holds. Then,*

1. *the set of edges to* s_i^j *in* $G^{1,k_1}(s_i^j)$ *is homogeneous to the set of edges to* s_i^l *in* $G^{1,k_2}(s_i^l)$*.*
2. *the set of edges from* s_i^j *in* $G^{1,k_1}(s_i^j)$ *is homogeneous to the set of edges from* s_i^l *in* $G^{1,k_2}(s_i^l)$*.*

Note that from the set of edges to s_i^j in $G^{1,k_1}(s_i^j)$ we can obtain the parametric upper bound function $\mathcal{F}_{s_i^j}^{max,k_1}$ for s_i^j, and from the set of edges from s_i^j in $G^{1,k_1}(s_i^j)$ we can obtain the parametric lower bound function $\mathcal{F}_{s_i^j}^{min,k_1}$ for s_i^j by inversely transforming the edge set into constraints. Two parametric lower (upper) bound functions for s_i^j and s_i^l are defined to be *homogeneous* if they satisfy condition 1(2) in the above proposition. If $\mathcal{F}_{s_i^j}^{min,k_1}$ and $\mathcal{F}_{s_i^l}^{min,k_2}$ are homogeneous, it is denoted as:

$$\mathcal{F}_{s_i^j}^{min,k_1} \sim \mathcal{F}_{s_i^l}^{min,k_2}$$

We have the following lemma from Proposition 4.7 and 4.8.

Lemma 4.4 *If $\Psi^{1,k}(f_N^{j-1}) \sim \Psi^{1,k}(f_N^j)$ holds for $2 \leq j \leq k-2$, then*

1. $\forall l : 2 \leq l \leq j-1 \;::\; \forall i : 1 \leq i \leq N \;::\; \mathcal{F}_{s_i^l}^{min,k} \sim \mathcal{F}_{s_i^j}^{min,k} \;\wedge\; \mathcal{F}_{s_i^l}^{max,k} \sim \mathcal{F}_{s_i^j}^{max,k}$

2. $\forall a : 1 \leq a :: \mathcal{F}_{s_i^{(j-1)+a}}^{min,k+a} \sim \mathcal{F}_{s_i^{j+a}}^{min,k+a} \;\wedge\; \mathcal{F}_{s_i^{(j-1)+a}}^{max,k+a} \sim \mathcal{F}_{s_i^{j+a}}^{max,k+a}$

This lemma enables us to obtain *asymptotic* [7] parametric bound functions, $\mathcal{F}_{s_i^j}^{min,\infty}$ and $\mathcal{F}_{s_i^j}^{max,\infty}$, once we find two homogeneous created edge sets during node elimination process from the constraint graph. By using asymptotic parametric bound functions at run-time we can guarantee that the constraint set $\mathcal{C}^{1,k}$ will be satisfied with any arbitrary value of k.

Note that asymptotic parametric bound functions, $\mathcal{F}_{s_i^j}^{min,\infty}$ and $\mathcal{F}_{s_i^j}^{max,\infty}$, are parameterized in terms of the variables in $\{s_1^{j-1}, f_1^{j-1}, \ldots, f_{i-1}^j\}$ and in terms of the index variable j. By knowing the scheduling window(job set) index j at run-time, only one pair of asymptotic parametric bound functions need to be stored for all s_i^j where i is fixed and $j \leq 2$. In addition to this, another pair of parametric bound functions needs to be stored for s_i^1.

4.1.4 Off-line Component

In this section, a $4N$-node graph, called *basis graph*, is obtained to which we can cyclically apply variable elimination algorithm without explicitly obtaining a large constraint graph $G^{1,k}(f_N^k)$ for large k. That is, by recursively applying variable elimination algorithm to this smaller graph, it can be decided whether the created edge set sequence, $\Psi^{1,k}(f_N^j)$, $j = k, k-1, \ldots$, will converge or not.

Definition 4.16 (Basis Graph) *A basis graph $G_b(V_b, E_b)$ is defined as a subgraph of $G^{1,2}(f_N^2)$ as follows.*[8]

1. $V_b = V_{b,1} \cup V_{b,2} \cup \{v_0\}$ *where:*

$$V_{b,1} = PrecNode(\Phi^{1,2}(f_N^1)) - \{v_0\}$$

$$V_{b,2} = \{s_1^2, f_1^2, \ldots, s_N^2, f_N^2\}$$

[7] "Asymptotic" means "converging" in the sense that homogeneous parametric bound functions will be found to the ones already obtained, if the variable elimination process continues.

[8] $G^{1,2}(f_N^2)$ is found from $\Gamma^{1,2}$.

2. All edges in $G^{1,2}(f_N^2)$ connecting any two nodes in V_b are included into E_b.

Then, the variable elimination process for a graph $G^{1,k}(f_N^k)$ can be transformed into an equivalent one by using a basis graph as follows:

Algorithm 4.3 *Cyclic algorithm to obtain $G^{1,k}(f_N^2)$.*

- *Input: k, Basis Graph $G_b(V_b, E_b)$*
- *Output: $G^{1,k}(f_N^2)$*

1. Initialize $i = 1$.

2. Initialize $G_{in}^1(V_b, E_{in}^1) = G_b(V_b, E_b)$.

3. From $i = 1$ to $i = k - 2$ repeat the following:

   (a) Eliminate, from $G_{in}^i(V_b, E_{in}^i)$, the nodes of $V_{b,2}$ by alternately using Algorithm 4.1 and 4.2.

   (b) If **False** *is returned from Algorithm 4.1 or 4.2, then return* **False**.

   (c) Let $G_{out}^i(V_{b,1} \cup \{v_0\}, E_{out}^i)$ denote the resulting graph.

   (d) If $i \geq 2$ and $G_{out}^i(V_{b,1} \cup \{v_0\}, E_{out}^i) = G_{out}^{i-1}(V_{b,1} \cup \{V_0\}, E_{out}^{i-1})$, then return $G_{in}^i(V_b, E_{in}^i)$.

   (e) Let $G_{in}^{i+1}(V_b, E_{in}^{i+1}) = G_b(V_b, E_b)$

   (f) For each edge $v_1 \xrightarrow{\mathrm{w}_{12}} v_2$ in $G_{out}^i(V_{b,1} \cup \{v_0\}, E_{out}^i)$,

      i. If $v_1 \neq v_0$ and $v_2 \neq v_0$, add an edge $g_{(1)}(v_1) \xrightarrow{\mathrm{w}_{12}} g_{(1)}(v_2)$ to $G_{in}^{i+1}(V_b, E_{in}^{i+1})$.

      ii. If $v_1 = v_0$, add an edge $g_{(1)}(v_1) \xrightarrow{\mathrm{w}_{12}+\mathrm{L}} g_{(1)}(v_2)$ to $G_{in}^{i+1}(V_b, E_{in}^{i+1})$.

      iii. If $v_2 = v_0$, add an edge $g_{(1)}(v_1) \xrightarrow{\mathrm{w}_{12}-\mathrm{L}} g_{(1)}(v_2)$ to $G_{in}^{i+1}(V_b, E_{in}^{i+1})$.

   (g) Set $i = i + 1$.

At step $3-(d)$ the graph $G_{in}^i(V_b, E_{in}^i)$ is returned. By utilizing Proposition 4.7, this graph can be shown to be equal to $G^{1,k}(f_N^2)$. Once we find homogeneous created edge sets on $V_{b,1} \cup \{v_0\}$ at step $3-(d)$, asymptotic parametric bound functions for job start times can be found from the graph $G^{1,k}(f_N^2)$. From this graph the variables in the sequence $< f_N^2, s_N^2, \ldots, f_1^2, s_1^2 >$ are eliminated to obtain the parametric bound functions for each s_i^2, $1 \leq i \leq N$. During this elimination process, the weights of edges connected to or from v_0 have to be modified appropriately to reflect scheduling window index $j \geq 2$ as well as the node index of the graph. For example,

- if an edge $v_0 \xrightarrow{\mathrm{w}} s_i^2$ is obtained after eliminating $< f_N^2, s_N^2, \ldots, f_i^2 >$, then a formula $s_i^j \leq w + (j-2)L$ must be used in deriving asymptotic parametric bound functions for s_i^j.
- if an edge $s_i^2 \xrightarrow{\mathrm{w}} v_0$ is obtained after eliminating $< f_N^2, s_N^2, \ldots, f_i^2 >$, then a formula $-w + (j-2)L \leq s_i^j$ must be used in deriving asymptotic parametric bound functions for s_i^j.
- if an edge $s_a^1 \xrightarrow{\mathrm{w}} s_i^2$, is obtained after eliminating $< f_N^2, s_N^2, \ldots, f_i^2 >$, then a formula $s_i^j - s_a^{j-1} \leq w$ must be used in deriving asymptotic parametric bound functions for s_i^j.

After obtaining asymptotic parametric bound functions for s_i^j, $2 \leq j$, we can also find parametric bound functions for Γ^1 by eliminating nodes from $G^{1,k}(f_N^1)$.

Note that, at each iteration in the above algorithm, no explicit transformation of node indices are performed by using $g_{(-1)}$. This is because our purpose is to check the schedulability and obtain asymptotic parametric bound functions, and this may be done without explicit knowledge of node indices. The key property that this algorithm makes use of is that the basis graph is recursively used and transformed until the schedulability is checked. It is clear that this algorithm produces exactly the same result (**True** or **False**) and graph as the node elimination algorithm applied to $G^{1,k}(f_N^k)$ does.

The following theorem provides an upper bound on the number of loop iterations in Algorithm 4.3 that have to be performed before the schedulability is checked.

Theorem 4.1 *If Algorithm 4.3 doesn't terminate within $n^2 - n + 2$ loop iterations, then $sched^{1,\infty}$ is not schedulable.*

Proof: Given in appendix.

Therefore, we obtain the final algorithm for checking $sched^{1,\infty}$ and deriving asymptotic parametric bound functions if $\Gamma^{1,\infty}$ is schedulable. The overview of off-line component is shown in Figure 4.6.

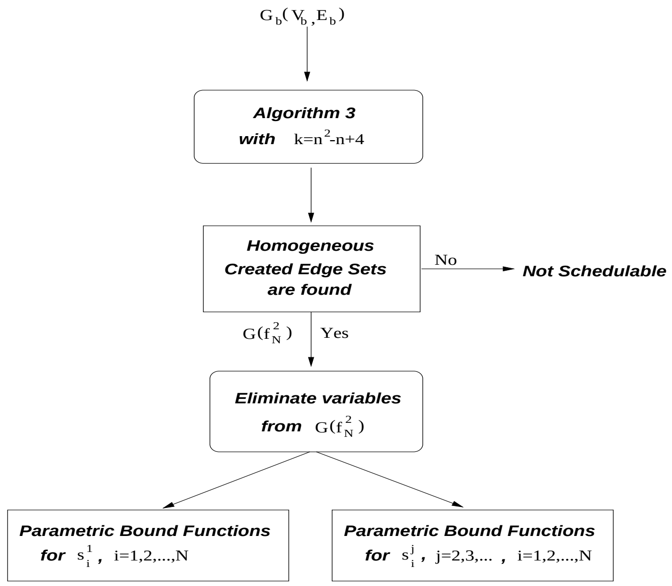

Figure 4.6: Overview of off-line component

From Theorem 4.1 the total complexity of the off-line algorithm is $O(n^2N^3)$, since each loop iteration of Algorithm 4.3 may take at most $O(N^3)$ computation time [23]. If only jitter constraints are allowed from periodic tasks, then the off-line algorithm will be finished within $O(n^4N)$ time where n is the number of periodic tasks that have jitter constraints, since each loop iteration in this case takes at most $O(n^2N)$ time. This is because $\mid PrecNode(\Phi^{1,k}(s_i^j)) \cup P \mid \leq\ n+2$ holds, and because from Proposition 4.3 we know that at most $O(n)$ number of edges exist in $G^{1,k}(s_i^j)$

that are connected to or from s_i^j. This implies that the elimination of s_i^j from the graph $G^{1,k}(s_i^j)$ will require at most $O(n^2)$ time, and eliminating nodes of one job set requires $O(n^2 N)$ time. Also, the on-line component in this case requires at most $O(n)$ execution time.

4.1.5 Off-line Component with Restricted Standard Constraints

For a certain class of standard constraints, called *restricted standard constraints*, it will be shown that the off-line component can be carried out in $O(N^3 + n^5)$ time instead of $O(n^2 N^3)$ time.

Definition 4.17 (Restricted Standard Constraints) *For two jobs, τ_a^j and τ_b^l, where $(j = l - 1) \vee (j = l \wedge a < b)$, the following constraints are defined as* restricted *standard constraints:*

$$\begin{array}{rcl} s_a^j - s_b^l & \leq & c_1 \\ s_a^j + e_a^j - s_b^l & \leq & c_2 \end{array} \qquad \begin{array}{rcl} s_b^l - s_a^j & \leq & c_3 \\ s_b^l + e_b^l - s_a^j & \leq & c_4 \end{array} \tag{4.2}$$

Also, as in the definition for standard constraints, release time and deadline constraints can also be classified as restricted standard constraints. We also include as restricted standard any constraint that can be rewritten in one of the above forms.

For this class of constraints the following lemma makes it possible to pre-process the basis graph and to obtain a smaller graph that can be used in the off-line algorithm instead of the basis graph. This graph is called a *compact* basis graph.

Lemma 4.5 ([42]) *If $\Gamma^{1,k}$ is constructed with restricted standard constraints, it is schedulable if and only if it is schedulable for the maximum execution times of the jobs.*

Let the following be a predicate representing a schedulability for a job set Π.

$$Sched \equiv \exists s_1 :: \forall e_1 \in [l_1, u_1] :: \ldots \exists s_i :: \forall e_i \in [l_i, u_i] :: \exists s_N :: \forall e_N \in [l_N, u_N] :: \mathcal{C}$$

From Lemma 4.5 this predicate is equivalent to the following predicate where $\mathcal{C}$ only consists of restricted standard constraints.

$$\exists s_1 :: \ldots :: \exists s_i :: \ldots \exists s_{N-1} :: \exists s_N :: \mathcal{C}[e_j/u_j \ : \ 1 \leq j \leq N]$$

where e_j/u_j denotes a substitution of u_j for a variable e_j. In other words, *Sched* can be checked by first replacing every universally quantified variable e_j with u_j for $1 \leq j \leq N$, and then by eliminating existentially quantified variables s_N, ..., s_1.

However, eliminating the existentially quantified variables, s_N, s_{N-1}, ..., s_{i+1}, in any order will produce the same constraint graph $G(s_i)$. This is because there exists no exclusively-ORed edges between nodes in $\{v_0, s_{i+1}, s_{i+2}, \ldots, s_N\}$ after substituting the maximum execution times for the variables e_j, $1 \leq j \leq N$, and because any minimum weight acyclic restricted paths through the nodes of $\{s_{i+1}, \ldots, s_N\}$ are preserved in the remaining constraint graph after eliminating the variables s_j, $i + 1 \leq j \leq N$, regardless of the elimination order.

This property is used to find a *compact* basis graph from $sched^{1,2}(f_N^2)$ as follows:

Algorithm 4.4 (Compact Basis Graph) *Algorithm to obtain a compact basis graph.*

- *Input:* $sched^{1,2}(f_N^2)$
- *Output: Compact Basis Graph* $G_{cb}(V_{cb}, E_{cb})$

1. *Let* $G^{'}(V^{'}, E^{'})$ *denote a graph from a predicate that is found by substituting* u_j *for each universally quantified variable* e_j *in* $sched^{1,2}(f_N^2)$.
2. *Let* $\Phi^{'}(s_N^1)$ *denote a crossing edge set of* s_N^1 *found from* $G^{'}(V^{'}, E^{'})$.
3. *Let* $G^{''}(V^{''}, E^{''})$ *denote a graph found after eliminating the following nodes from* $G^{'}(V^{'}, E^{'})$.
$$\{s_1^2, s_2^2, \ldots, s_N^2\} - g_{(1)}(PrecNode(\Phi^{'}(s_N^1)))$$
4. *Let* $G_{cb}(V_{cb}, E_{cb})$ *be a subgraph of* $G^{''}(V^{''}, E^{''})$:
   (a) $V_{cb} = V_{cb,1} \cup V_{cb,2} \cup \{v_0\}$ *where:*
$$V_{cb,1} = \{s_1^1, s_2^1 \ldots, s_N^1\} \cap PrecNode(\Phi^{'}(s_N^1))$$
$$V_{cb,2} = g_{(1)}(V_{cb,1})$$
   (b) *All edges in* $G^{''}(V^{''}, E^{''})$ *connecting two nodes of* V_{cb} *defines* E_{cb}.

We can apply Algorithm 4.3 to this compact basis graph instead of the basis graph. This limits the complexity of obtaining homogeneous created edge sets to $O(N^3 + n^5)$ instead of $O(n^2N^3)$. Once we find homogeneous created edge sets on $V_{cb,1}$, asymptotic parametric bound functions can be found by first unrolling the final graph from the algorithm to obtain $G^{1,\infty}(f_N^2)$ and then by eliminating from this graph the nodes in the sequence $< f_N^2, s_N^2, \ldots, f_1^2, s_1^2 >$. During this elimination process, as in Section 4.1.4 the weights of edges connecting v_0 have to be modified appropriately to reflect scheduling window index as well as the node indices of the graph.

4.2 Example

The asymptotic parametric bound functions are found for the job set, $\Gamma^{1,\infty}$, in Example 3.1. Figure 4.7 shows the parametric bound functions found from $\Gamma^{1,4}$, and Figure 4.8 shows asymptotic parametric bound functions for $sched^{1,\infty}$.

It is clear from this figure that the following hold:

$$\mathcal{F}_{s_1^2}^{min,4} \sim \mathcal{F}_{s_1^3}^{min,4}$$
$$\mathcal{F}_{s_1^2}^{max,4} \sim \mathcal{F}_{s_1^3}^{max,4}$$
$$\mathcal{F}_{s_2^2}^{min,4} \sim \mathcal{F}_{s_2^3}^{min,4}$$
$$\mathcal{F}_{s_2^2}^{max,4} \sim \mathcal{F}_{s_2^3}^{max,4}$$

Note that $n = \mid PrecNode(\Phi^{1,4}(f_2^1)) \mid = 3$, and $n^2 - n + 2 = 8$ is the iteration bound given in Theorem 4.1. But, Algorithm 4.3 found homogeneous created edge sets after 3 loop iterations. This shows that the upper bound on the number of loop iterations given in Theorem 4.1 is not tight in general.

0	$\leq$	s_1^1	$\leq$	2
$\max(8, s_1^1 + e_1^1)$	$\leq$	s_2^1	$\leq$	$\min(10, s_1^1 + e_1^1 + 5)$
$\max(20, s_2^1 + e_2^1, s_1^1 + e_1^1 + 10)$	$\leq$	s_1^2	$\leq$	$\min(22, s_1^1 + e_1^1 + 17, s_2^1 + e_2^1 + 4)$
$\max(28, s_1^2 + e_1^2, s_2^1 + e_2^1 + 10)$	$\leq$	s_2^2	$\leq$	$\min(30, s_2^1 + e_2^1 + 12, s_1^2 + e_1^2 + 5)$
$\max(40, s_2^2 + e_2^2, s_1^2 + e_1^2 + 10)$	$\leq$	s_1^3	$\leq$	$\min(42, s_1^2 + e_1^2 + 17, s_2^2 + e_2^2 + 4)$
$\max(48, s_1^3 + e_1^3, s_2^2 + e_2^2 + 10)$	$\leq$	s_2^3	$\leq$	$\min(50, s_2^2 + e_2^2 + 12, s_1^3 + e_1^3 + 5)$
$\max(60, s_2^3 + e_2^3, s_1^3 + e_1^3 + 10)$	$\leq$	s_1^4	$\leq$	$\min(62, s_1^3 + e_1^3 + 17, s_2^3 + e_2^3 + 4)$
$\max(s_1^4 + e_1^4, s_2^3 + e_2^3 + 10)$	$\leq$	s_2^4	$\leq$	$\min(70, s_2^3 + e_2^3 + 12, s_1^4 + e_1^4 + 5)$

Figure 4.7: Parametric bound functions found from $sched^{1,4}$

$$
\begin{aligned}
\mathcal{F}_{s_1^1}^{min} &= 0 \\
\mathcal{F}_{s_1^1}^{max} &= 2 \\
\mathcal{F}_{s_2^1}^{min} &= \max(8, s_1^1 + e_1^1) \\
\mathcal{F}_{s_2^1}^{max} &= \min(10, s_1^1 + e_1^1 + 5) \\
\mathcal{F}_{s_1^j}^{min} &= \max(20 + (j-2)20, s_2^{j-1} + e_2^{j-1}, s_1^{j-1} + e_1^{j-1} + 10) \\
\mathcal{F}_{s_1^j}^{max} &= \min(22 + (j-2)20, s_1^{j-1} + e_1^{j-1} + 17, s_2^{j-1} + e_2^{j-1} + 4) \\
\mathcal{F}_{s_2^j}^{min} &= \max(28 + (j-2)20, s_1^j + e_1^j, s_2^{j-1} + e_2^{j-1} + 10) \\
\mathcal{F}_{s_2^j}^{max} &= \min(30 + (j-2)20, s_2^{j-1} + e_2^{j-1} + 12, s_1^j + e_1^j + 5)
\end{aligned}
$$

Figure 4.8: Asymptotic parametric bound functions for $sched^{1,\infty}$

4.3 Summary

In this chapter, we presented a solution approach for the problem defined in Chapter 3. A new technique, called dynamic cyclic dispatching, is developed based on the dynamic time-based scheduling scheme introduced in Chapter 1.

A schedule(ordering) of N jobs is assumed to be given on a scheduling window, and it is required that this schedule be repeated at run time. The relative constraints may be cyclically defined across the boundaries of the scheduling windows as well as between jobs in one scheduling window.

Unlike static approaches which assign fixed start times to jobs in a scheduling window, our approach not only allows us to flexibly manage the slack times with the schedulability of a job set not affected, but also yields an guaranteed schedulability in the sense that, if other dispatching mechanism can dispatch the job sequences satisfying all given constraints, then our mechanism can also schedule them.

A pseudo-polynomial time off-line algorithm is presented to check the schedulability of a cyclically constrained job set and to obtain parametric lower and upper bound functions for each job start time. The off-line algorithm requires at most $O(n^2 N^3)$ time where n is the number of jobs in a scheduling window that have relative constraints with jobs in the next scheduling window. Then, the parametric bound functions for each start time can be evaluated by an on-line algorithm within $O(N)$ time. In addition, with restricted standard constraints it is shown that the off-line component requires at most $O(N^3 + n^5)$ execution time.

Chapter 5

Design of a Dynamic Temporal Controller

5.1 Introduction

In this chapter, we consider the issue of how a dynamic temporal controller can be constructed. In dynamic temporal control, the regular sampling interval assumption is relaxed, and computational costs are incorporated into the cost function. At run-time new controls are computed and exercised at chosen time instants such that the cost function is minimized. The feasibility of this new scheme is demonstrated by obtaining dynamic temporal control laws for linear time-invariant control systems.

In Section 5.2, we formulate the dynamic temporal control problem and introduce computation cost into performance index function. The solution approach for linear time-invariant systems is discussed in Section 5.3. In Section 5.4, implementation issues are addressed. We provide an example of controlling rigid body satellite in Section 5.5 . In this example, a dynamic temporal controller is designed. Results show that the dynamic temporal control approach performs better than the traditional sampled data control approach with the same number of control exercises. Section 5.6 discusses the issues arising from the application of dynamic temporal controls to the design of real-time control systems. Finally, Section 5.7, we present a summary.

5.2 Problem Formulation

In dynamic temporal control, the control changing time instants are chosen such that a cost function is minimized which incorporates computational costs as well as state, input costs. We consider a steady state control problem on a finite time line $[0, T_f]$. To formulate the dynamic temporal control problem for a discrete, linear time-invariant system, we first discretize the time interval $[0, T_f]$ into M subintervals of length $\Delta = T_f/M$. Let $D_M = \{0, \Delta, 2\Delta, \ldots (M-1)\Delta\}$ denote M time instants that are regularly spaced. Here, control exercising time instants are restricted within D_M for the purpose of simplicity. The linear time-invariant controlled process is described by the difference equation:

$$x(k+1) \quad = \quad Ax(k) + Bu(k) \tag{5.1}$$

where k is the time index. One unit of time represents the subinterval Δ, whereas $x \in \mathcal{R}^n$ and $u \in \mathcal{R}^l$ are the state and input vectors, respectively.

It is well known that there exists a steady state optimal control law [20, 39]

$$u^o(i) = f_i[x(i)] \qquad i = 0, 1, ..., M-1 \tag{5.2}$$

that minimizes the quadratic performance index function (Cost)

$$J_M = \sum_{k=0}^{M-1} [x^T(k)Qx(k) + u^T(k)Ru(k)] + x^T(M)Qx(M) \tag{5.3}$$

where $Q \in \mathcal{R}^{n\times n}$ is positive semi-definite and $R \in \mathcal{R}^{l\times l}$ is positive definite.

As we can see, traditional controller exercises control at every time instant in D. However, in temporal control, we are no longer constrained to exercise control at every time instant in D. In *dynamic temporal control* we require that the control be exercised with the following steps: At time t_i, $t_i \in D_M$ and $1 \leq i$,

1. Compute a current state $x(t_i)$
2. Compute $\delta(t_i)$
3. Compute and apply $u(t_i)$ to the system
4. Repeat the process at $t_{i+1} = t_i + \delta(t_i)$

Note that t_i, $1 \leq i$, denote control changing time instants, and $\delta(t_i)$ denotes the time interval between i-th control exercise and $(i+1)$-th control exercise.

For the purpose of simplicity, dual mode dynamic temporal control is considered. That is, $\delta(t_i)$ may take one of the following two values:

- $a\Delta$
- $b\Delta$

a and b are positive integers($a < b$) such that b is an integer multiple of a. Also, it is assumed that b divides M without any remainder. $b\Delta$ is called a *base sampling period* and $a\Delta$ is called a *rapid sampling period*. Let $M = \beta b$ where β is a positive integer.

In addition to the above assumption, we further assume that at all time instants in $\{0, b\Delta, 2b\Delta, \ldots, (\beta-1)b\Delta\}$ new controls are computed. Let each time interval $[(i-1)b\Delta, ib\Delta]$ of size $b\Delta$ be called a *frame* for $1 \leq i \leq \beta$. The sampling period decision function δ is evaluated at only time instants that are start times of frames, and once $\delta(ib\Delta)$ is decided it will be enforced during the next time frame $[ib\Delta, (i+1)b\Delta]$. In other words, if $\delta(ib\Delta) = a\Delta$ the control computations will be done at $ib\Delta$, $(ib+a)\Delta$, $\ldots$, $(ib+b-a)\Delta$. And, if $\delta(ib\Delta) = b\Delta$ the control computations will be done only at $ib\Delta$ in $[ib\Delta, (i+1)b\Delta)$. Under these assumptions the steps performed by a dynamic temporal controller can be summarized as follows: At time $ib\Delta$, $0 \leq i \leq \beta - 1$,

1. Compute a current state $x(ib\Delta)$
2. Compute $\delta(ib\Delta) = g_i(x(ib\Delta))$
   (a) If $\delta(ib\Delta) = a\Delta$
      - At $t_j = (ib+ja)\Delta$ for $0 \leq j \leq (b/a - 1)$, compute and apply $u(t_j) = h_{i,j}(x(t_j))$

   (b) If $\delta(ib\Delta) = b\Delta$
      - compute and apply $u(ib\Delta) = h_i(x(ib\Delta))$

3. Repeat the process at $(i+1)b\Delta$ if $i < \beta - 1$.

This new formulation of dynamic temporal control makes it possible to find a good approximation approach to optimal control laws as can be seen in later sections of this chapter.

We want to find a feedback control law, g_i, h_i, and $h_{i,j}$ for $i = 0, 1, 2, ..., \beta - 1$ and $j = 0, 1, \ldots, (b/a - 1)$, that minimizes a new performance index function

$$J_M^{'} \quad = \quad J_M + \mu \cdot \gamma \tag{5.4}$$

Here, μ is the computation cost of exercising control with a rapid sampling period instead of a base sampling period in one frame, and γ denotes the number of frames in $[0, M\Delta]$ done with a rapid sampling period. Hence, exercising controls with a rapid sampling period increases the cost term $\mu \cdot \gamma$. So, if exercising control with a rapid sampling period doesn't reduce the term J_M by more than this increase, exercising control with a base sampling period is likely to be a better choice. This is a key idea of the solution approach given in the next section.

This new cost function is different from J_M in two aspects. First, the concept of computational cost is introduced in $J_M^{'}$ as $\mu \cdot \gamma$ term to regulate the number of frames with rapid sampling periods. If we do not take this computation cost into consideration γ is likely to become β. If computation cost is high (i.e., μ has a large value) then γ is likely to be small in order to minimize the total cost function. Second, in dynamic temporal control, not only do we seek control law $u(x(t))$, but also the control exercising time instants and the number of control changes. In the next section, we present in detail specific techniques for finding a dynamic temporal control law with performances close to optimal solutions.

5.3 Temporal Control with Fixed Sampling Times

Let $\mathcal{T} = \{t_0, t_1, t_2, \ldots, t_{\nu-1}\}$ denote a set of control changing time instants where $t_0 = 0$, $t_1 = n_1\Delta$, ..., $t_{\nu-1} = n_{\nu-1}\Delta$. That is n_0, n_1, ..., $n_{\nu-1}$ are the indices for control changing time instants. In this section, an optimal control law is derived when $\mathcal{T}$ is given which minimizes the cost function J_M. In the next section, the results developed in this section will be used in devising good heuristics for deciding δ values minimizing $J_M^{'}$.

Assume that $\mathcal{T}$ is given. Then a new control input calculated at t_i will be applied to the actuator for the next time interval from t_i to t_{i+1}. Our objective here is to determine the optimal control law

$$u^o(n_i) = f_i[x(n_i)] \;\; i = 0, 1, ..., \nu - 1 \tag{5.5}$$

that minimizes the quadratic performance index function (Cost) J_M which is defined in (5.4).

The principle of optimality, developed by Richard Bellman[7, 8] is the approach used here. That is, if a closed loop control $u^o(n_i) = f_i[x(n_i)]$ is optimal over the interval $t_0 \le t \le t_\nu$, then it is also optimal over any sub-interval $t_m \le t \le t_\nu$, where $0 \le m \le \nu$. As it can be seen from Figure 5.1, the total cost J_M can be decomposed into F_is for $0 \le i \le \nu$ where

$$\begin{aligned} F_i \;\; &= \;\; x^T(n_i)Qx(n_i) + x^T(n_i+1)Qx(n_i+1) \\ &+ \;\; x^T(n_i+2)Qx(n_i+2) + ... + x^T(n_{i+1}-1)Qx(n_{i+1}-1) \\ &+ \;\; (n_{i+1}-n_i)u^T(n_i)Ru(n_i) \end{aligned} \tag{5.6}$$

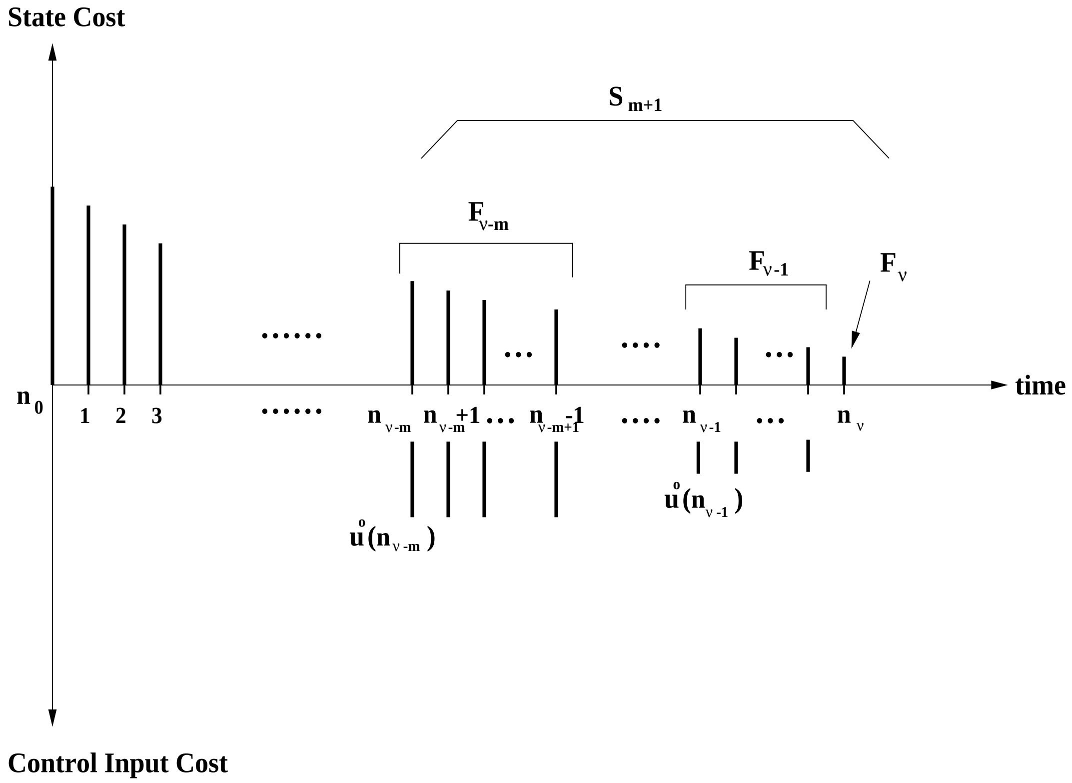

Figure 5.1: Decomposition of J_M into F_i.

That is, from (5.1),

$$\begin{aligned}
F_i &= x^T(n_i)Qx(n_i) + (Ax(n_i) + Bu(n_i))^T Q(Ax(n_i) + Bu(n_i)) \\
&+ (A^2x(n_i) + ABu(n_i) + Bu(n_i))^T Q(A^2x(n_i) + ABu(n_i) + Bu(n_i)) \\
&+ ... + (A^{n_{i+1}-n_i-1}x(n_i) + A^{n_{i+1}-n_i-2}Bu(n_i) + ... + ABu(n_i) + Bu(n_i))^T Q \\
&\quad (A^{n_{i+1}-n_i-1}x(n_i) + A^{n_{i+1}-n_i-2}Bu(n_i) + ... + ABu(n_i) + Bu(n_i)) \\
&+ (n_{i+1} - n_i)u^T(n_i)Ru(n_i)
\end{aligned} \tag{5.7}$$

This can be rewritten as

$$\begin{aligned}
F_i &= x^T(n_i)Qx(n_i) + \sum_{j=1}^{n_{i+1}-n_i-1} [A_jx(n_i) + B_ju(n_i)]^T Q[A_jx(n_i) + B_ju(n_i)] \\
&+ (n_{i+1} - n_i)u^T(n_i)Ru(n_i)
\end{aligned} \tag{5.8}$$

where $A_j = A^j$ and $B_j = \sum_{k=0}^{j-1} A^k B$.

Then J_M can be expressed as

$$J_M = F_0 + F_1 + F_2 + ... + F_\nu. \tag{5.9}$$

Let S_m be the cost from $i = \nu - m + 1$ to $i = \nu$:

$$S_m = F_{\nu-m+1} + F_{\nu-m+2} + ... + F_{\nu-1} + F_\nu, \;\; 1 \leq m \leq \nu + 1. \tag{5.10}$$

These cost terms are well illustrated in the above Figure 5.1.

Therefore, by applying the principle of optimality, we can first minimize $S_1 = F_\nu$, then choose $F_{\nu-1}$ to minimize $S_2 = F_{\nu-1} + F_\nu = S_1^o + F_{\nu-1}$ where S_1^o is the optimal cost occurred at t_ν. We can continue choosing $F_{\nu-2}$ to minimize $S_3 = F_{\nu-2} + F_{\nu-1} + F_\nu = F_{\nu-2} + S_2^o$ and so on until $S_{\nu+1} = J_M$ is minimized. Note that $S_1 = F_\nu = x^T(n_\nu)Qx(n_\nu)$ is determined only from $x(n_\nu)$ which is independent of any other control inputs.

5.3.1 Inductive Construction of an Optimal Control Law with $\mathcal{T}$ Given

We inductively derive an optimal controller which changes its control at ν time instants t_0, t_1, ..., $t_{\nu-1}$. As we showed in the previous section, the inductive procedure goes backwards in time from S_1^o to $S_{\nu+1}^o$. Since $S_1 = F_\nu = x^T(n_\nu)Qx(n_\nu) + u^T(n_\nu)Ru(n_\nu)$ and $x(n_\nu)$ is independent of $u(n_\nu)$, we can let $u^o(n_\nu) = u^o(M) = 0$ and $S_1^o = x^T(n_\nu)Qx(n_\nu)$ where Q is symmetric and positive semi-definite.

Induction Basis: $S_1^o = x^T(n_\nu)Qx(n_\nu)$ where Q is symmetric.

Inductive Assumption: Suppose that
$S_m^o = x^T(n_{\nu-m+1})P(\nu - m + 1)x(n_{\nu-m+1})$
holds for some m
where $1 \le m \le \nu$ and $P(\nu - m + 1)$ is symmetric.

We can write S_m^o as

$$\begin{aligned} S_m^o &= [A_{(n_{\nu-m+1}-n_{\nu-m})}x(n_{\nu-m}) + B_{(n_{\nu-m+1}-n_{\nu-m})}u(n_{\nu-m})]^T \\ &\quad \cdot P(\nu - m + 1) \cdot \\ &\quad [A_{(n_{\nu-m+1}-n_{\nu-m})}x(n_{\nu-m}) + B_{(n_{\nu-m+1}-n_{\nu-m})}u(n_{\nu-m})] \end{aligned} \tag{5.11}$$

From the definition of S_m and (5.8),

$$\begin{aligned} S_{m+1} &= S_m^o + F_{\nu-m} \\ &= S_m^o + x^T(n_{\nu-m})Qx(n_{\nu-m}) \\ &+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [A_jx(n_{\nu-m}) + B_ju(n_{\nu-m})]^T Q[A_jx(n_{\nu-m}) + B_ju(n_{\nu-m})] \\ &+ (n_{\nu-m+1} - n_{\nu-m})u^T(n_{\nu-m})Ru(n_{\nu-m}) \end{aligned} \tag{5.12}$$

And the above equation becomes

$$\begin{aligned} S_{m+1} &= [A_{n_{\nu-m+1}-n_{\nu-m}}x(n_{\nu-m}) + B_{n_{\nu-m+1}-n_{\nu-m}}u(n_{\nu-m})]^T P(\nu - m + 1) \\ &\quad [A_{n_{\nu-m+1}-n_{\nu-m}}x(n_{\nu-m}) + B_{n_{\nu-m+1}-n_{\nu-m}}u(n_{\nu-m})] \\ &+ x^T(n_{\nu-m})Qx(n_{\nu-m}) \\ &+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [A_jx(n_{\nu-m}) + B_ju(n_{\nu-m})]^T Q[A_jx(n_{\nu-m}) + B_ju(n_{\nu-m})] \\ &+ (n_{\nu-m+1} - n_{\nu-m})u^T(n_{\nu-m})Ru(n_{\nu-m}) \end{aligned} \tag{5.13}$$

If we differentiate S_{m+1} with respect to $u(n_{\nu-m})$, then

$$\begin{aligned}
\frac{\partial S_{m+1}}{\partial u(n_{\nu-m})} &= B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) A_{n_{\nu-m+1}-n_{\nu-m}} x(n_{\nu-m}) \qquad (5.14)\\
&+ (A^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}})^T x(n_{\nu-m})\\
&+ 2B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}} u(n_{\nu-m})\\
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [2B_j^T Q A_j x(n_{\nu-m}) + 2B_j^T Q B_j u(n_{\nu-m})]\\
&+ 2(n_{\nu-m+1}-n_{\nu-m}) R u(n_{\nu-m})\\
= \quad & 2\{B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) A_{n_{\nu-m+1}-n_{\nu-m}} \qquad (5.15)\\
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q A_j\} x(n_{\nu-m})\\
&+ 2\{B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}}\\
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q B_j + (n_{\nu-m+1}-n_{\nu-m}) R\} u(n_{\nu-m})
\end{aligned}$$

Note that $P(\nu-m+1)$ is symmetric and the following three rules are applied to differentiate S_{m+1} above.

$$\begin{aligned}
\frac{\partial}{\partial x}(x^T Q x) &= 2Qx\\
\frac{\partial}{\partial x}(x^T Q y) &= Qy\\
\frac{\partial}{\partial y}(x^T Q y) &= Q^T x
\end{aligned}$$

Let $\frac{\partial S_{m+1}}{\partial u(n_{\nu-m})} = 0$, from Lemma 5.1 and Lemma 5.2 given later we can obtain $u^o(n_{\nu-m})$ which minimizes S_{m+1} and thus obtain S^o_{m+1}.

$$\begin{aligned}
u^o(n_{\nu-m}) &= -\{B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}} \qquad (5.16)\\
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q B_j + (n_{\nu-m+1}-n_{\nu-m}) R\}^{-1}\\
&\{B^T_{n_{\nu-m+1}-n_{\nu-m}} P(\nu-m+1) A_{n_{\nu-m+1}-n_{\nu-m}}\\
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q A_j\} x(n_{\nu-m})\\
&= -K(\nu-m) x(n_{\nu-m})
\end{aligned}$$

where $K(\nu-m)$ is defined in (5.16).

Therefore, we can write

$$\begin{aligned}
A_{n_{\nu-m+1}-n_{\nu-m}} x(n_{\nu-m}) \quad + \quad & B_{n_{\nu-m+1}-n_{\nu-m}} u^o(n_{\nu-m}) = \qquad (5.17)\\
& [A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}} K(\nu-m)] x(n_{\nu-m})
\end{aligned}$$

If we use (5.16) and (5.17), we have

$$\begin{aligned}
S^o_{m+1} &= \{[A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)]x(n_{\nu-m})\}^T \\
&\quad P(\nu-m+1)\{[A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)]x(n_{\nu-m})\} \\
&+ x^T(n_{\nu-m})Qx(n_{\nu-m}) \\
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} \{[A_j - B_jK(\nu-m)]x(n_{\nu-m})\}^T Q \\
&\quad \{[A_j - B_jK(\nu-m)]x(n_{\nu-m})\} \\
&+ (n_{\nu-m+1}-n_{\nu-m})[K(\nu-m)x(n_{\nu-m})]^T R[K(\nu-m)x(n_{\nu-m})]
\end{aligned} \tag{5.18}$$

This equation can be rewritten as

$$\begin{aligned}
S^o_{m+1} &= x^T(n_{\nu-m})\{[A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)]^T \\
&\quad P(\nu-m+1)[A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)] \\
&+ Q \\
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [A_j - B_jK(\nu-m)]^T Q[A_j - B_jK(\nu-m)] \\
&+ (n_{\nu-m+1}-n_{\nu-m})K^T(\nu-m)RK(\nu-m)\}x(n_{\nu-m}). \\
&= x^T(n_{\nu-m})P(\nu-m)x(n_{\nu-m})
\end{aligned} \tag{5.19}$$

where $P(\nu-m)$ is obtained from $K(\nu-m)$ and $P(\nu-m+1)$ as in (5.19). Also note that knowing $P(\nu-m+1)$ is enough to compute $K(\nu-m)$ because other terms of (5.16) are known a priori.

Therefore, we find a symmetric matrix $P(\nu-m)$ satisfying $S^o_{m+1} = x^T(n_{\nu-m})P(\nu-m)x(n_{\nu-m})$. From (5.16) and (5.19), we have the following recursive equations for obtaining $P(\nu-m)$ from $P(\nu-m+1)$ where $m = 1, 2, ..., \nu$.

$$\begin{aligned}
K(\nu-m) &= \{B^T_{n_{\nu-m+1}-n_{\nu-m}}P(\nu-m+1)B_{n_{\nu-m+1}-n_{\nu-m}} \\
&\quad + \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B^T_jQB_j + (n_{\nu-m+1}-n_{\nu-m})R\}^{-1} \\
&\quad \{B^T_{n_{\nu-m+1}-n_{\nu-m}}P(\nu-m+1)A_{n_{\nu-m+1}-n_{\nu-m}} \\
&\quad\quad + \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B^T_jQA_j\}
\end{aligned} \tag{5.20}$$

$$\begin{aligned}
P(\nu-m) &= [A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)]^T \\
&\quad P(\nu-m+1)[A_{n_{\nu-m+1}-n_{\nu-m}} - B_{n_{\nu-m+1}-n_{\nu-m}}K(\nu-m)] \\
&+ Q
\end{aligned} \tag{5.21}$$

$$
\begin{aligned}
&+ \sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} [A_j - B_j K(\nu-m)]^T Q [A_j - B_j K(\nu-m)] \\
&+ (n_{\nu-m+1} - n_{\nu-m}) K^T(\nu-m) R K(\nu-m)
\end{aligned}
$$

Also, we know that at each time instant $n_{\nu-m}\Delta$

$$
u^o(n_{\nu-m}) = -K(\nu-m)x(n_{\nu-m}) \tag{5.22}
$$

Hence, with $P(\nu) = Q$, we can obtain $K(i)$ and $P(i)$ for $i = \nu-1, \nu-2, ..., 0$ recursively using (5.20) and (5.21). At each time instant $n_i\Delta,\ i = 0, 1, 2, ..., \nu-1$ the new control input value will be obtained using (5.22) by multiplying $K(i)$ by $x(n_i)$ where $x(n_i)$ is the estimate of the system state at $n_i\Delta$. Also, note that the optimal control cost is $J_M^o = S_{\nu+1}^o = x^T(0)P(0)x(0)$ where $P(0)$ is found from the above procedure.

To prove the optimality of this control law we need the following lemmas.

Lemma 5.1 *If Q is positive semi-definite and R is positive definite, then $P(i),\ i = \nu, \nu-1, \nu-2, ..., 0,$ matrices are positive semi-definite. Hence, $P(i)$s are symmetric from the definition of a positive semi-definite matrix.*

Proof Since $P(\nu) = Q$, from assumption $P(\nu)$ is positive semi-definite. Assume that for $k = i+1$, $P(k)$ is positive semi-definite. We use induction to prove that $P(i)$ is semi-definite. Note that Q is positive semi-definite and R is positive definite. From (5.21) we have

$$
\begin{aligned}
P(i) &= [A_{n_{i+1}-n_i} - B_{n_{i+1}-n_i} K(i)]^T P(i+1) \\
&\quad [A_{n_{i+1}-n_i} - B_{n_{i+1}-n_i} K(i)] \\
&+ Q \\
&+ \sum_{j=1}^{n_{i+1}-n_i-1} [A_j - B_j K(i)]^T Q [A_j - B_j K(i)] \\
&+ (n_{i+1} - n_i) K^T(i) R K(i)
\end{aligned} \tag{5.23}
$$

Since $P(i+1)$ and Q are positive semi-definite, R is positive definite, and $(n_{i+1} - n_i) > 0$, it is easy to verify that for $\forall y \in R^m\ :\ y^T P(i) y \geq 0$. This means that $P(i)$ is positive semi-definite. This inductive procedure proves the lemma.

Lemma 5.2 *Given $\mathcal{T}$, the inverse matrix in (5.20) always exists.*

Proof Let $V = B_{n_{\nu-m+1}-n_{\nu-m}}^T P(\nu-m+1) B_{n_{\nu-m+1}-n_{\nu-m}}$ $+\sum_{j=1}^{n_{\nu-m+1}-n_{\nu-m}-1} B_j^T Q B_j + (n_{\nu-m+1} - n_{\nu-m})R$. From Lemma 5.1, $P(\nu-m+1)$ is positive semi-definite. Therefore, $\forall y \in R^m\ :\ y^T V y > 0$ because Q is positive semi-definite, R is positive definite and $n_{\nu-m+1} - n_{\nu-m} > 0$. This implies that V is positive definite. Hence the inverse matrix exists.

Theorem 5.1 *Given $\mathcal{T}$, $K(i)$ $(i = 0, 1, 2, ..., \nu - 1)$ obtained from the above procedure are the optimal feedback gains which minimize the cost function J_M (and J'_M) on $[0, M\Delta]$.*

Proof Note that given $\mathcal{T}$, J_M is a convex function of $u(n_i), i = 0, 1, ..., \nu - 1$. Thus the above feedback control law is optimal.

Suppose that $\mathcal{T}_1$ and $\mathcal{T}_2$ denote two sets of control changing time instants.

Lemma 5.3 *If $\mathcal{T}_1 \subseteq \mathcal{T}_2$, then $J^o_{M,1} \geq J^o_{M,2}$ where $J^o_{M,1}$ and $J^o_{M,2}$ are the optimal costs of controls which change controls at time instants in $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively.*

Proof Suppose that $J^o_{M,1} < J^o_{M,2}$, then, in controlling the system with $\mathcal{T}_2$, if we do not change controls at time instants in $\mathcal{T}_2 - \mathcal{T}_1$ and change controls at time instants in $\mathcal{T}_1$ to the same control inputs that were exercised to get $J^o_{M,1}$ with $\mathcal{T}_1$, we obtain $\hat{J}_{M,2}$ which is equal to $J^o_{M,1}$. This contradicts the fact that $J^o_{M,2}$ is the minimum cost obtainable with D_q since we have found $\hat{J}_{M,2}$ which is equal to $J^o_{M,1}$ and therefore less than $J^o_{M,2}$. Hence, $J^o_{M,1} \geq J^o_{M,2}$.

This lemma implies that if we do not take computation cost, μ, into consideration, then the more control exercising points, the better the controller is (less cost). With the computation cost being included in the cost function, the statement above is no longer true. Therefore we need to search for an optimal $\mathcal{T}$ which minimizes the cost function J'_M. The following sections provide a detailed discussion on searching for such an optimal solution. Note that if we let $\mathcal{T} = D_M$ then the optimal temporal control law is the same as the traditional linear feedback optimal control law.

5.3.2 Dynamic Temporal Control

In this section, we design a dynamic temporal controller by introducing a heuristic for $\delta(ib\Delta)$ function. The heuristic tries to estimate how much performance gain(reduction of J_M term in J'_M) and how much performance loss (increase of $\mu\gamma$ term) will incur if a rapid sampling period is used in the next frame. If the performance gain is greater than or equal to a given threshold θ, then $\delta(i) = a\Delta$, otherwise $\delta(i) = b\Delta$.

By making use of the results developed in the previous section, we can obtain an optimal control law for $\mathcal{T}_i^1 = \{ib\Delta, (i+1)b\Delta, \ldots, (\beta-1)b\Delta\}$ on a time interval $[ib\Delta, \beta b\Delta]$ where $0 \leq i \leq \beta - 1$. Let $\mathcal{K}_1(i)$ and $\mathcal{P}_1(i)$ denote two matrices found from $\mathcal{T}_i^1$ by applying the algorithm given in the previous section.

Consider another control changing time instants set $\mathcal{T}_i^2 = \{ib\Delta, (ib+a)\Delta, \ldots, (ib+b-a)\Delta, (i+1)b\Delta, \ldots, (\beta-1)b\Delta\}$ where $0 \leq i \leq \beta - 1$. Also, let $\mathcal{K}_2(i)$ and $\mathcal{P}_2(i)$ denote two matrices found from $\mathcal{T}_i^2$ by applying the algorithm given in the previous section. Also, let $\mathcal{K}_2(i,j), 0 \leq j \leq (b/a-1)$, denote a gain matrix obtained for time instant $(ib + ja)\Delta$.

Two control changing time sets, $\mathcal{T}_i^1$ and $\mathcal{T}_i^2$, are depicted in Figure 5.2.

From Lemma 5.3 we know that $x^T(ib\Delta)\ \mathcal{P}_1(i)\ x(ib\Delta)$ is less than or equal to $x^T(ib\Delta)\ \mathcal{P}_2(i)\ x(ib\Delta)$. Furthermore, $x^T(ib\Delta)\ \mathcal{P}_1(i)\ x(ib\Delta)$ is less than or equal to $x^T(ib\Delta)\ \mathcal{P}_a(i)\ x(ib\Delta)$ where

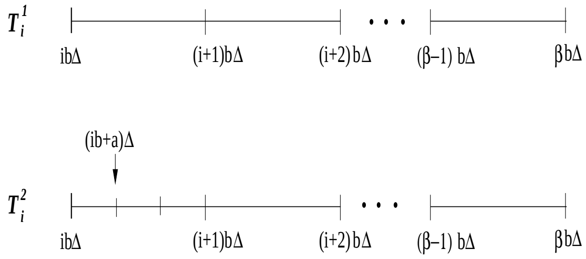

Figure 5.2: Two control changing time sets $\mathcal{T}_i^1$ and $\mathcal{T}_i^2$.

$\mathcal{P}_a(i)$ is a matrix found from any arbitrary control changing time instant set on $[ib\Delta, \beta b\Delta]$ conforming to the assumptions given in the problem formulation section, i.e., the same sampling period is enforced during one frame.

In addition, the cost $x^T(ib\Delta)\ \mathcal{P}_2(i)\ x(ib\Delta)$ is less than or equal to $x^T(ib\Delta)\ \mathcal{P}_b(i)\ x(ib\Delta)$ where $\mathcal{P}_b(i)$ is a matrix found from any arbitrary control changing time instant set on $[ib\Delta, \beta b\Delta]$ that contains time instants $ib\Delta, (ib+a)\Delta, \ldots, (ib+b-a)\Delta$, i.e., a rapid sampling period is used in the first frame $[ib\Delta, (i+1)b\Delta]$.

From these facts, it can be said that a cost $x^T(ib\Delta)\ \mathcal{P}_1(i)\ x(ib\Delta)$ is a lower bound of the costs found from any control changing time instant sets on $[ib\Delta, \beta b\Delta]$ that conform to the assumptions, and a cost $x^T(ib\Delta)\ \mathcal{P}_2(i)\ x(ib\Delta)$ is a lower bound of the costs found from any control changing time instant sets that enforce rapid sampling period in the first frame $[ib\Delta, (i+1)b\Delta)$.

In our solution approach, the above costs are used at time $ib\Delta$ to estimate the performance gain of using a rapid sampling period in the next frame $[ib\Delta, (i+1)b\Delta]$. This is a heuristic approach, and the effectiveness of this approach is validated through an example in a later section.

We present a heuristic dynamic temporal control law which performs the following steps at each frame start time:

1. Compute a current state $x(ib\Delta)$
2. If $x^T(ib\Delta)(\mathcal{P}_1(i) - \mathcal{P}_2(i))x(ib\Delta) < \theta$, let $\delta(i) = b\Delta$.
   Otherwise, let $\delta(i) = a\Delta$.
   (a) If $\delta(i) = a\Delta$,
      - At each time instant $t_j = ib\Delta + ja\Delta$, $0 \le j \le (b/a - 1)$,
        apply $u(t_j) = -\mathcal{K}_2(i,j)x(t_j)$
   (b) If $\delta(i) = b\Delta$,
      - $u(ib\Delta) = -\mathcal{K}_1(i)x(ib\Delta)$
3. Repeat the process at $(i+1)(b\Delta)$

The following theorem proves that the dynamic temporal control using the above control law guarantees the cost term J_M of $J_M^{'}$ to be less than or equal to $x^T(0)\ \mathcal{P}_1(0)\ x(0)$ which is a cost for $\mathcal{T}_0^1$ with only a base sampling period enforced on the entire interval $[0, T_f]$.

Theorem 5.2 *If the above dynamic temporal control law is used, the cost J_M of $J_M^{'}$ is less than or equal to $x^T(0)\mathcal{P}_1(0)x(0)$ where $\mathcal{P}_1(0)$ is obtained from $\mathcal{T}_0^1$.*

Proof Suppose that $C^d(x_0)$ denotes a set of time instants at which new controls are exercised according to the above dynamic temporal control law for a given initial state x_0. Let $I^d(x_0) = \{i \mid 1 \leq i \leq \beta\}$ denote a set of frame indices at which a rapid sampling period is used. Also, let $i_1 \in I^d(x_0)$ denote a smallest index in $I^d(x_0)$, and $i_2 \in I^d(x_0)$ denote a second smallest index, and so on. Consider two control changing time sets, $\mathcal{T}_0^1$ and $\mathcal{T}_0^{'}$, where in $\mathcal{T}_0^{'}$ only i_1-th frame uses a rapid sampling period. Also, suppose that for these two control changing time sets, $\mathcal{K}_1(l)$ is used if l-th frame uses a base sampling period, and $\mathcal{K}_2(l,j)$ is used if l-th frame uses a rapid sampling period. Under these assumptions, it is clear that the control cost (without computation cost) for $\mathcal{T}_0^1$ is greater than or equal to that for $\mathcal{T}_0^{'}$, when the same initial state x_0 is used.

Consider two control changing time sets, $\mathcal{T}_0^{'}$ and $\mathcal{T}_0^{''}$, where in $\mathcal{T}_0^{''}$ i_1-th and i_2-th frames use a rapid sampling period. Also, suppose that for these two control changing time sets, $\mathcal{K}_1(l)$ is used if l-th frame uses a base sampling period, and $\mathcal{K}_2(l,j)$ is used if l-th frame uses a rapid sampling period. Under these assumptions, it is clear that the control cost (without computation cost) for $\mathcal{T}_0^{'}$ is greater than or equal to that for $\mathcal{T}_0^{''}$, when the same initial state x_0 is used.

If we transitively apply this process, we can conclude that, for the same initial state x_0, the control cost (without computation cost) for $\mathcal{T}_0^1$ is greater than or equal to that obtained by applying the dynamic temporal control law. This proves the theorem.

5.4 Implementation

To implement dynamic temporal control, we need to calculate and store $\mathcal{K}_1(i)$ and $\mathcal{K}_2(i,j)$ matrices, and use them when controlling the system. The number of matrices that need to be stored is $O(\beta + (b/a)\beta)$, which is $O((b/a)\beta)$. Note that in traditional optimal linear control a similar matrix is obtained and used at every time instant in D_M to generate control input value.

In dynamic temporal control, there is a CPU time overhead for calculating $x^T(ib\Delta)\ (\mathcal{P}_1(i) - \mathcal{P}_2(i))\ x(ib\Delta)$ at the start of each frame. This calculation can be done within $O(n^2)$ time. This calculation has to be done once each frame. More discussion is presented in a discussion section on this overhead.

In order to implement temporal control we require an operating system that supports scheduling control computations at specific time instants, and allows dynamic selection of sampling periods. The Maruti system developed at the University of Maryland is a suitable host for the implementation of dynamic temporal control [43, 35, 34]. In Maruti, all executions are scheduled in time and the time of execution can be modified dynamically, if so desired. This is in contrast with traditional cyclic executives often used in real-time systems, which have a fixed, cyclic operation and which are well suited only for the sampled data control systems operating in a static environment. It is the availability of the system such as Maruti that allows us to consider the notion of dynamic temporal control, in which time becomes an emergent property of the system.

5.5 Example

To illustrate the advantages of a dynamic temporal control scheme let us consider a simple example of rigid body satellite control problem [51]. The system state equations are as follows:

$$
\begin{aligned}
x(k+1) &= \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 0.00125 \end{bmatrix} u(k) \\
y(k) &= \begin{bmatrix} 1 & 1 \end{bmatrix} x(k)
\end{aligned}
$$

where k represents the time index and one unit of time is the discretized subinterval of length $\Delta = 0.05$. The linear quadratic performance index J_M in (5.4) is used here with the following parameters.

$$
\begin{aligned}
Q &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
R &= 0.0001 \\
M &= 40 \\
\Delta &= 0.05 \\
a &= 1 \\
b &= 4
\end{aligned} \tag{5.24}
$$

The objective of the control is to drive the satellite to the zero position and the desired goal state is $x_f = [0,\ \ 0]^T$.

We applied the dynamic temporal control law with an initial state space $\{(x_1, x_2) \mid \ 0.2 \leq x_1, x_2 \leq 0.8\}$ with the following parameter:

$$
\theta = 0.01 \tag{5.25}
$$

The performance of the dynamic temporal controller is compared to that of traditional optimal control with a sampling period 0.05. In Figure 5.3 the cost differences between dynamic temporal controller and a traditional optimal controller are depicted for each initial state (x_1, x_2). Note that the maximum cost difference is less than 0.03. In Figure 5.4 the number of control computation performed by a dynamic temporal controller is shown for each initial state. Note that the maximum number of control computation is less than 20, and for many of initial states they are less than 18.

To estimate how much cost reduction is achieved through dynamic temporal control, we compare its performance with that of traditional optimal controller with 0.1 sampling period, i.e., sampling is done at 20 regular spaced time instants. In Figure 5.5 the cost differences between optimal controller with 0.05 sampling period and an optimal controller with 0.1 sampling period are depicted for each initial state (x_1, x_2). Note that the maximum cost difference is almost 0.5. The cost differences shown in Figure 5.3 and Figure 5.5 are compared together in Figure 5.6. Note that with almost all initial states the dynamic temporal controller outperforms traditional optimal controller with sampling period 0.1, even though the number of control computations done by a dynamic temporal controller is smaller than that for optimal controller.

If we normalize the costs from dynamic temporal controller and from traditional controller with sampling period 0.1, by dividing by the cost from traditional controller with sampling period 0.05, we obtain graphs shown in Figure 5.7.

The Figure 5.7 shows two graphs, one for normalized costs from dynamic temporal controller and the other for normalized costs from traditional controller with a sampling period 0.1. Note that

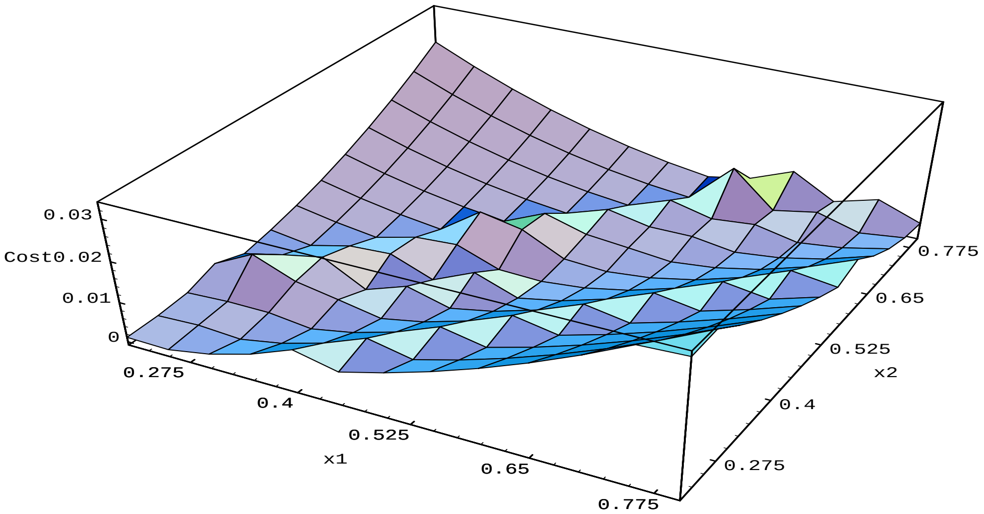

Figure 5.3: Cost differences between dynamic temporal controller and traditional controller with 0.05 sampling period. The maximum cost difference is less than 0.03.

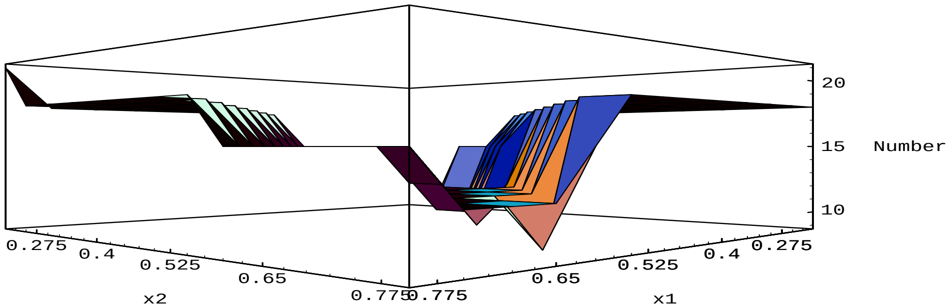

Figure 5.4: Number of control computation performed by a dynamic temporal controller is shown for each initial state. Note that the maximum number of control computation is less than 20, and for many of initial states they are less than 18.

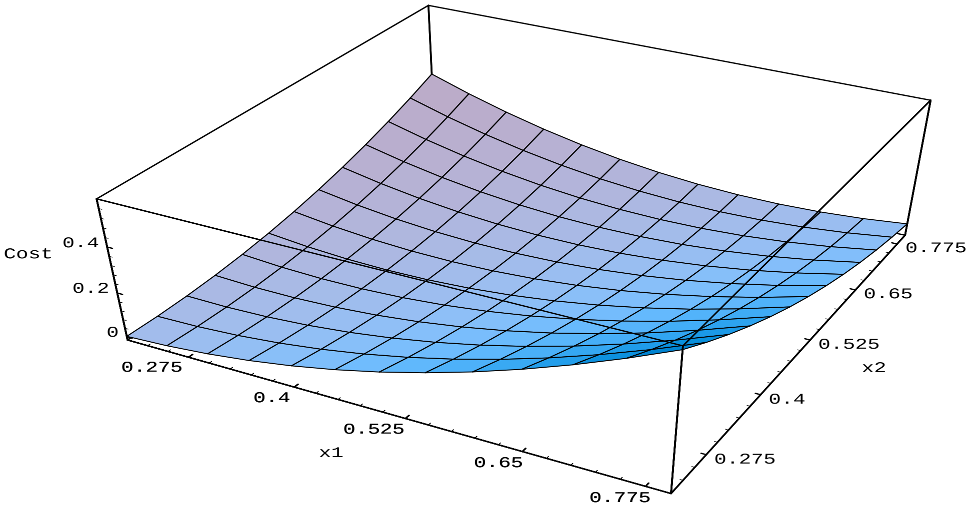

Figure 5.5: Cost differences between optimal controller with 0.05 sampling period and an optimal controller with 0.1 sampling period are depicted for each initial state. The maximum cost difference is almost 0.5.

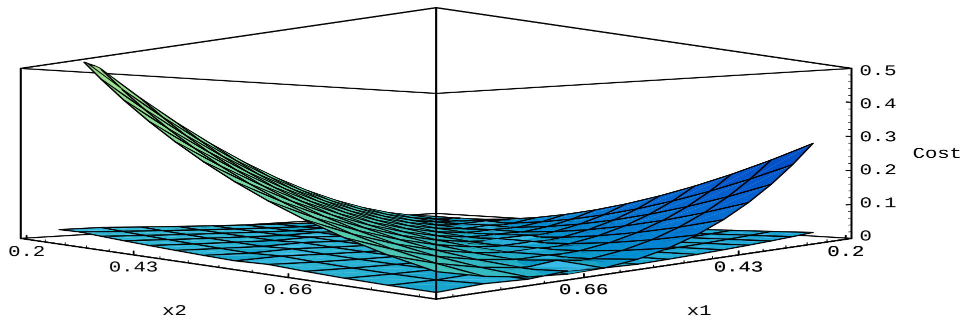

Figure 5.6: Cost differences shown in Figure 5.3 and Figure 5.5 are compared together. Note that for almost all initial states the dynamic temporal controller outperforms traditional controller with equal sampling period 0.1, even though the number of control computations done by a dynamic temporal controller is smaller than that for traditional controller.

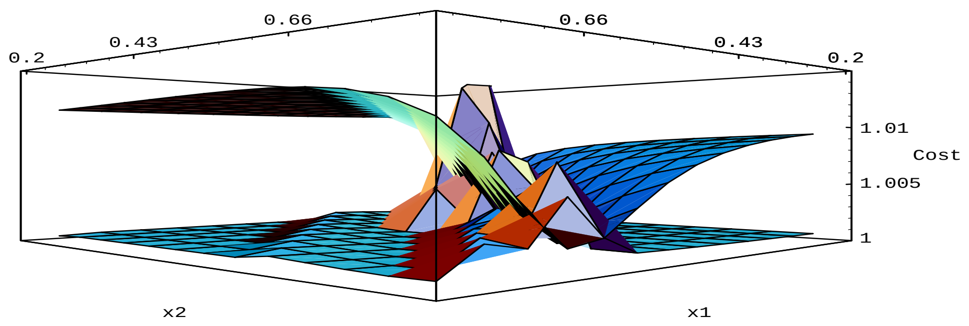

Figure 5.7: Normalize costs from dynamic temporal controller and from traditional controller with sampling period 0.1. Costs are normalized by dividing by the cost from traditional controller with sampling period 0.05.

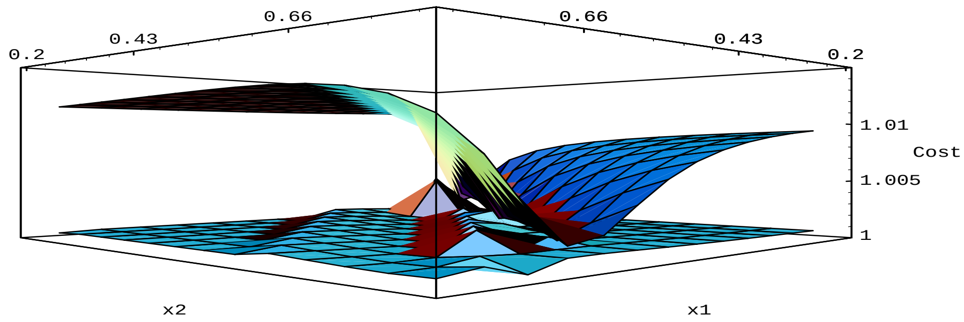

Figure 5.8: Normalized costs from two controllers with adjusted threshold values. One from dynamic temporal controller and the other from traditional controller with equal sampling period 0.1.

for some initial states the optimal controller outperforms dynamic temporal controller. However, this is from using uniform threshold value θ for the entire initial state space. As a result of using one threshold value, the number of control computations over initial state space shows non-uniformity as can be seen in Figure 5.4. By adjusting threshold values for some initial state, we can obtain more uniform graphs. This is seen from Figure 5.8 which is found after using different(smaller) threshold values for the initial states that results in higher normalized costs in Figure 5.7.

The differences between normalized costs shown in Figure 5.8 is not so big, less than 0.01. However, the advantage of dynamic temporal scheme is more clearly seen from the following experiment. Usually, in concurrent real-time systems, the actual control update time instants for one periodic control task varies in consecutive periods. This is from the variations of task execution times and also from the resource contention between different tasks. The delay of control update from the ideal control updating time instant is called *computational delay*. Computational delay has an adverse effect on control algorithm's performance. Figure 5.9 shows the differences of

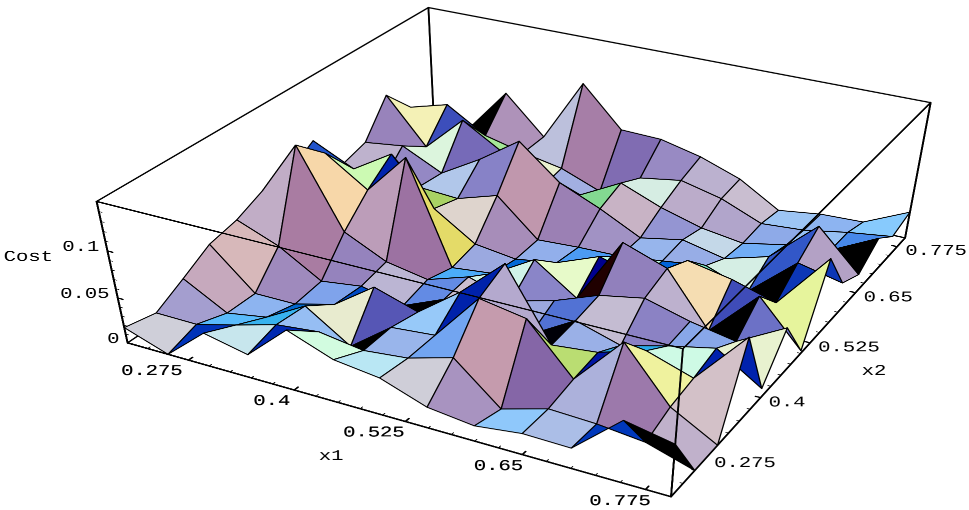

Figure 5.9: Differences of worst case normalized costs between a dynamic temporal controller with $\theta = 0.01$ and a traditional controller with a sampling period 0.1. The computational delays are randomly generated with a normal distribution. For each initial state, the control trajectories are found 100 times, and the maximum cost among them is recorded.

worst case normalized costs between a dynamic temporal controller with $\theta = 0.01$ and a traditional controller with a sampling period 0.1. The computational delays are randomly generated with a normal distribution in $[0, \Delta]$, and they are injected into the system trajectory. For each initial state, the control trajectories are found 100 times, and the maximum cost among them is recorded. The graph shows that using a dynamic temporal controller reduces normalized costs.

Finally, Figure 5.10 shows the differences of average normalized costs between a dynamic temporal controller with $\theta = 0.01$ and a traditional controller with a sampling period 0.1. Again, the computational delays are randomly generated with a normal distribution. For each initial state, the control trajectories are found 100 times, and the average cost is recorded. This figure shows that the differences are not as large as in Figure 5.9 since they are from average costs.

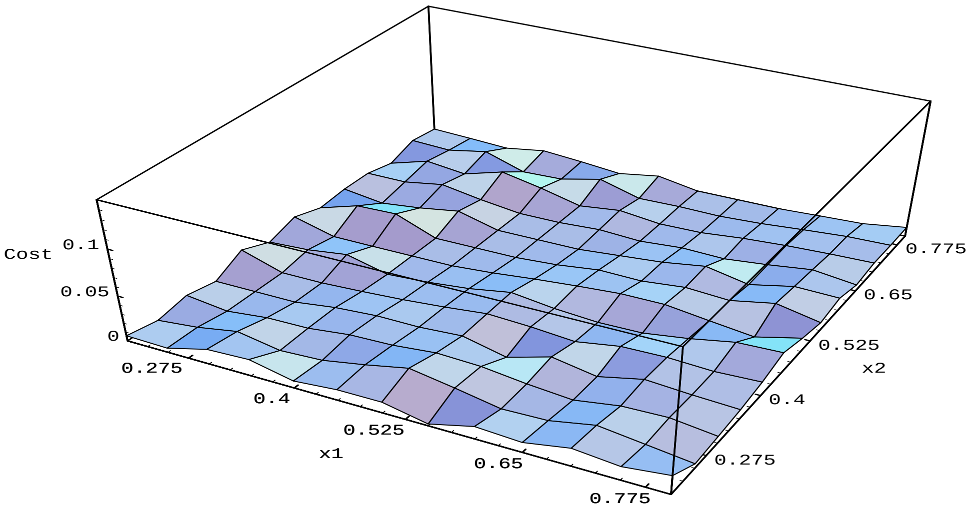

Figure 5.10: Differences of average normalized costs between a dynamic temporal controller with $\theta = 0.01$ and a traditional controller with a sampling period 0.1. The computational delays are randomly generated with a normal distribution. For each initial state, the control trajectories are found 100 times, and the average cost is recorded.

5.6 Discussion

In the previous section, we showed by using an example that the number of control computations can be dramatically reduced by using dynamic temporal control law, while not sacrificing the quality of control. Employing the dynamic temporal control methodology in concurrent real-time embedded systems will have a significant impact on the way computational resources are utilized by control tasks. A minimal amount of control computations can be obtained for a given regulator by which we can achieve almost the same control performance compared to that of traditional controller with equal sampling period. This significantly reduces the CPU times for each controlling task and thus increases the number of real-time control functions which can be accommodated concurrently in one embedded system. Particularly, in a hierarchical control system if dynamic temporal controllers can be employed for lower level controllers the higher level controllers will have a great degree of flexibility in managing resource usages by adjusting computational requirements of each lower level controller. For example, in emergency situations the higher level controller may force the lower level controller to run as infrequently as they possibly can (thus freeing computational resources for handling the emergency). In contrast, during normal operations the temporal control tasks may run as necessary, and the additional computation time can be used for higher level functions such as monitoring and planning, etc.

As is mentioned in Section 5.4, there is an associated CPU overhead with dynamic temporal controller. At start of each frame the sampling period decision has to be done, which requires $O(n^2)$ execution time. However, this computation is required once every frame, and we can get benefits by reducing the number of context switches in concurrent real-time systems.

More work needs to be done on the effects of computational delays and variations on control systems performance when dynamic temporal controls are used.

5.7 Summary

In this chapter, we proposed a *dynamic temporal control* technique based on a new cost function which takes into account computational cost as well as state and input cost. In this scheme new control input values are defined at time instants which are not necessarily regularly spaced. For the linear control problem we showed that almost the same quality of control can be achieved while much less computations are used than in a traditional controller.

The proposed formulation of dynamic temporal control is likely to have a significant impact on the way concurrent embedded real-time systems are designed. In hierarchical control environment, this approach is likely to result in designs which are significantly more efficient and flexible than traditional control schemes. As it uses less computational resources, the lower level temporal controllers will make the resources available to the higher level controllers without compromising the quality of control.

Chapter 6

Scheduling Aperiodic and Sporadic Tasks

6.1 Introduction

In this chapter we develop an approach to addressing the problem of *incremental* scheduling of dynamic tasks in a hard real-time system.

Traditionally, the scheduling problem considered for real-time systems is that of generating a schedule for n tasks. In practice, however, a system may have to accept additional tasks during its operation. Here, we study the problem of *incremental scheduling* in *dynamic* time-based environment. We assume that we are given a set of n tasks, $\mathcal{T}$(and all their task instances), along with a schedule for their execution. We consider adding a task to the schedule. To add a new task, we have to first analyze the acceptability of it. If this task can not be scheduled without violating constraints of any of the tasks in $\mathcal{T}$ then this task is not accepted. If this can be scheduled, we not only accept the task, but also add it to the schedule.

In Section 6.2 the incremental scheduling problem is formally defined within a time-based scheduling scheme. The results on incremental scheduling of aperiodic and sporadic tasks are presented in Section 6.3. Finally, a summary follows in Section 6.4.

6.2 Problem Description

The main problem addressed in this chapter is how to incrementally accept and schedule tasks while not sacrificing the schedulability of the tasks already accepted.

A task in a real-time system may invoke its corresponding *task instances* by informing the system of the release time, deadline, and execution time of the task instance. Tasks in real-time systems may be classified into *single instance* task and *multiple instance* task. Single instance task, which is also called *aperiodic* task, invokes its task instance only once, and multiple instance task invokes its instance repeatedly. Multiple instance tasks are further divided into *periodic* tasks and *sporadic* tasks. A periodic task invokes its instances at regular time intervals(period), whereas a sporadic task invokes its instances at any time instant with a defined minimum inter-arrival time between two consecutive invocations.

Any arriving task belongs to one of these classes. A periodic task P is characterized by an invocation of a sequence of task instances. The following characteristics are assumed to be known at the arrival time, $\mathcal{A}^p$, of the periodic task, P.

- task invocation time $\mathcal{I}^p$ from which the task starts to invoke its instances.
- task termination time $\mathcal{X}^p$ when the task is terminated.
- period p
- invocation time of the j-th task instance is defined to be $\mathcal{I}_j^p = \mathcal{I}^p + (j-1)p$
- relative deadline d^p which implies that the absolute deadline of j-th task instance is $\mathcal{I}_j^p + d^p$.
- worst case execution time c^p

A hard aperiodic task A invokes its task instance only once. A has the following set of parameters:

- arrival time of the request, $\mathcal{A}^a$
- ready time $\mathcal{R}^a$ from which the task instance can start its execution.
- relative deadline d^a which implies that the absolute deadline is $D^a = \mathcal{R}^a + d^a$
- worst case execution time c^a

A sporadic task S is characterized by an invocation of its task instances with a minimum inter-arrival time. The following characteristics are assumed to be known at the arrival time, $\mathcal{A}^s$, of the sporadic task, S.

- task invocation time $\mathcal{I}^s$ from which the task instances can be invoked.
- task termination time $\mathcal{X}^s$ when the task is terminated.
- minimum inter-arrival time δ
- invocation time of the j-th task instance, $\mathcal{I}_j^s$, can be any time instant satisfying $\mathcal{I}_j^s \geq \mathcal{I}_{j-1}^s + \delta$
- relative deadline d^s ($\leq \delta$) which implies that the absolute deadline of the j-th task instance is $\mathcal{I}_j^s + d^s$.
- worst case execution time c^s

In addition to these, the system may be called upon to handle non-realtime tasks which don't have deadlines; Instead, they require as fast completion time as possible(best effort).

For a set of task instances to be scheduled, a traditional time-based scheduling scheme first finds a complete schedule for them in a given scheduling window. This schedule contains a static start time, s_i, for each task instance, which is decided based on the worst case execution time c_i and reflects all task dependencies. However, to enhance the scheduler with the ability to schedule dynamically arriving tasks, it may change s_i at runtime, while conforming to all constraints, such as release time r_i, deadline d_i, precedence relations, relative constraints, etc. Clearly, this additional information has to be kept for each task instance with the schedule. If a new task arrives, based on the current schedule it needs to be decided whether this new task can be accepted by the system, and if it can be accepted, a new schedule has to be constructed to incorporate this new task.

In a hard real-time environment, tasks may be executed in preemptive or non-preemptive manner. When a task is executed non-preemptively it begins execution at time s_i and is assured CPU access for the time, c_i, without any interruption or preemption. In preemptive execution, the task execution may be preempted at some defined time instant, and resumed at a later time instant. Note that the task preemption and resumption times may be dynamically decided.

We extend the static time-based scheduling scheme into a dynamic time-based scheduling scheme that enables any dynamically arriving aperiodic, periodic, or sporadic task to be incrementally scheduled. In a traditional static time-based scheduling scheme, every resource requirement is met by assigning explicit start times to the task instances. But, in this dynamic time-based scheduling scheme, the start times no longer have to be statically determined. Instead, the schedule includes a mechanism for determining the time when a task instance will be started or resumed based on the information available prior to its start time.

6.3 Dynamic Time-based Scheduling Schemes

Two variations of dynamic time-based scheduling scheme are presented here. In Section 6.3.1, a mechanism is presented to incrementally schedule aperiodic tasks over a schedule for static tasks found at pre-runtime. In Section 6.3.2, a mechanism is presented to incrementally schedule sporadic(periodic) tasks. In both sections, it is assumed that a valid schedule of static tasks is initially given with start times of the task instances. We develop acceptance tests for dynamically arriving aperiodic(or sporadic) tasks under the assumption that the total ordering among the static tasks is maintained, and EDF scheduling policy is assumed to be used for resolving the CPU contentions between static and dynamic tasks. Between static tasks, the time-based scheduling scheme is used in a sense that a total ordering among them is maintained at run-time, and between static(dynamic) and dynamic tasks, EDF scheduling algorithm is used.

6.3.1 Aperiodic Task Scheduling

In this section, a mechanism is presented to schedule arriving aperiodic tasks. The key idea of this mechanism is to make use of the fact that the task executions may be dynamically shifted to the left or to the right in a time line as long as the timing constraints of the tasks can be satisfied. All task instances in this section are assumed to be preemptable.

Task Model

We assume that an initial schedule of task instances is given in a scheduling window $[0, L]$ and this schedule is used by dispatcher at run-time. Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_N\}$ be a set of task instances in the initial schedule. It is assumed that τ_i is scheduled before τ_{i+1} in the schedule. Each task instance τ_i has the following parameters in the schedule:

- release time R_i
- absolute deadline D_i ($D_i \leq L$ for all $1 \leq i \leq N$)
- worst case execution time C_i

- runtime variable e_i denoting the processing time already spent for τ_i up to a current time instant
- runtime variable ω_i denoting the latest start(or resume) time of τ_i, which is a function of the current time t and the value of e_i
- earliest start time $est(i)$
- latest start time $lst(i)$

A hard aperiodic task A is defined the same way as in Section 6.2 except that the ready time is assumed to be equal to its arrival time, i.e, $\mathcal{A}^a = R^a$. Also, the task instances in Γ are assumed to be *preemptable* by an aperiodic task and any aperiodic task is assumed to be *preemptable* by a task instance in Γ.

The values of $est(i)$, $lst(i)$, $i = 1, 2, \ldots, N$, are found as follows:

$$\begin{aligned} est(1) &= R_1 \\ est(i) &= \max(R_i, est(i-1) + C_i) \quad \text{for } i = 2, 3, \ldots, N \\ lst(N) &= D_N - C_N \\ lst(i) &= \min(D_i, lst(i+1)) - C_i \quad \text{for } i = N-1, N-2, \ldots, 1 \end{aligned}$$

If $D_i \le lst(i+1)$, then $lst(i)$ value will be decided from D_i. And if $D_i > lst(i+1)$, then $lst(i)$ will be decided from $lst(i+1)$. Fig 6.1 shows an example of these relationships.

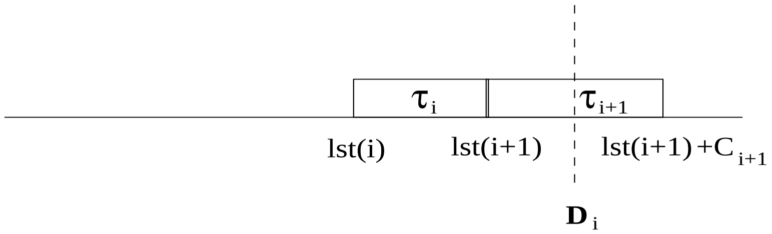

Figure 6.1: Deriving $\omega_i(0)$ recursively

Also, Fig 6.2 shows an example set of task instances with their $est(i)$ and $lst(i)$.

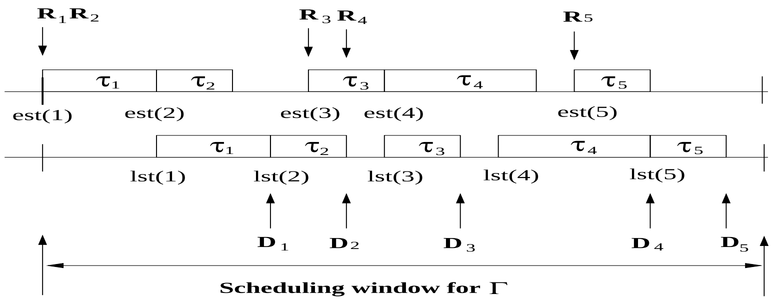

Figure 6.2: $est(i)$ and $lst(i)$ for an example task set

Note that the run-time variable e_i is initialized to 0 and ω_i to $lst(i)$.

Γ and a set of arriving aperiodic tasks A_1, ..., A_i are said to be *feasible* if and only if there exists a schedule which satisfies all the timing constraints on Γ and aperiodic tasks. The *optimality* of a scheduling algorithm is defined as:

Definition 6.1 (Optimality) *A scheduling algorithm is* optimal *if and only if the following is satisfied:*

- *It can schedule Γ and arriving aperiodic tasks whenever there exists a feasible schedule.*

Scheduling of Non-realtime Tasks

We can efficiently schedule any non-realtime tasks in a sense that maximum processor time can be found and used to service non-realtime tasks at any time instant by delaying as much as possible the executions of task instances. The non-realtime tasks are assumed to be processed by using FIFO [1] scheduling policy.

At a current time instant t_1, let τ_j denote a task instance in Γ which is just finished or partially executed. Also, let t_0 denote the last time instant when the dispatcher took control before t_1, and let t_2 denote the run-time variable denoting the future time instant when the dispatcher can take control. The dispatcher takes control whenever a non-realtime task or a task instance in Γ is finished, or whenever $t_1 = t_2$ holds. Then, at a current time instant t_1 when a dispatcher takes the control:

```
If τ_j is executed in [t_0, t_1]
  then
       let e_j = e_j + t_1 − t_0
       let ω_j = ω_j + t_1 − t_0
If τ_j is finished
  then
       let j = j + 1
let t_2 = ω_j
If t_1 < ω_j
  then
       if there exists a non-realtime task pending,
          then give the processor to the first non-realtime task in the queue
          else  if R_j ≤ t_1,
                   then give the processor to τ_j
                   else  let the processor be idle
  else
       give the processor to τ_j
```

If no non-realtime tasks are pending, the next(or partially executed) task τ_j is executed if it is possible, i.e., the release time of it is reached. Whenever there exists a non-realtime task waiting in the queue, and the latest start(or resume) time, ω_j, is not reached for τ_j the non-realtime task will be executed(after preempting τ_j if it is already started) until it finishes or ω_j is reached. If it

[1] FIFO stands for First In First Out.

continues its execution until ω_j, the non-realtime task is preempted and τ_j will resume its execution or start its execution. In other words, the non-realtime tasks have higher priorities until the latest start(or resume) time of τ_j is reached.

Example case is shown in Fig 6.3.

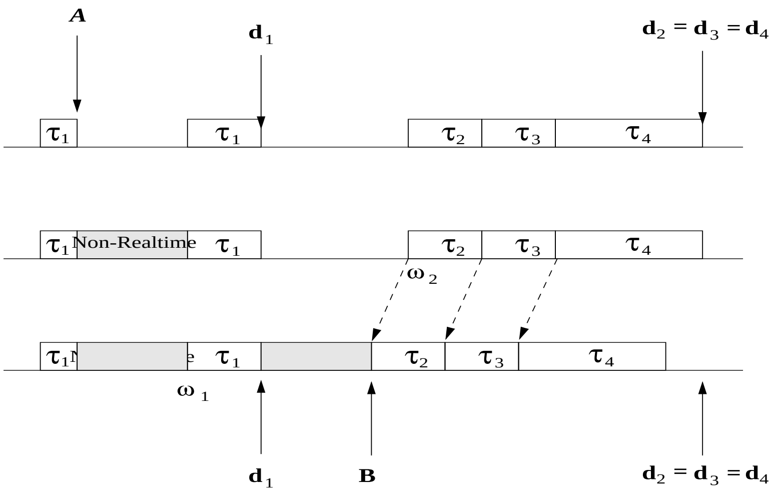

Figure 6.3: Joint scheduling of a non-realtime and Γ

Acceptance Test for A Hard Aperiodic Task

In some real-time systems, there may exist aperiodic tasks that may arrive to the system at any time instants. At their arrival times, tests should be performed to decide if they can be accepted to the system or not. Once an aperiodic task is accepted and started it must be completed before its hard deadline. If it is rejected, then a higher level entity in the application may decide the following steps to the rejection message. For example, the higher level task may decide to re-invoke the aperiodic task until it is finally accepted.

In this section, an acceptance test is developed that should be performed at the arrival times of hard aperiodic tasks. It is assumed that the context switch overheads are small and they are not taken into account in our work.

The relative deadline of an aperiodic task A is assumed to be less than or equal to the scheduling window size L. The approach taken in this section treats arriving aperiodic task instances in FIFO order. This assumption will be removed in the next section.

The acceptance test algorithm follows. Assume that τ_i is the next or partially executed task when the hard aperiodic task, A, arrived at time R^a.

At the arrival time, R^a, of an aperiodic task, A:

$TotalCapacity = \omega_i - R^a$
$k = i + 1$
While ($TotalCapacity < c^a$ and $lst(k) \leq R^a + d^a$)
    begin
    $TotalCapacity = TotalCapacity + lst(k) - lst(k-1) - C_k$

$k = k + 1$
If $(TotalCapacity \geq c^a)$
then Return(Success)
end
$TotalCapacity = TotalCapacity + \max(0, R^a + d^a - lst(k-1) - C_{k-1})$
If $(TotalCapacity \geq c^a)$
then Return(Success)
else Return(Fail)

At the arrival time of an aperiodic task, R^a, the acceptance test can be done in $O(M)$ time within this framework where M denotes the total number of task instance $\tau_j(i \leq j)$ which satisfies $R^a \leq lst(j) \leq R^a + d^a$. In this case, the total amount of available processor time for A in $[R^a, R^a + d^a]$ can be found by the following formula:

$$\begin{aligned} \Omega(R^a, R^a + d^a) &= \omega_i - R^a \\ &+ \sum_{k=i}^{j'-1} (lst(k+1) - lst(k) - C_k) \\ &+ \max(0, R^a + d^a - lst(j') - C_{j'}) \end{aligned} \tag{6.1}$$

where $j'(i \leq j')$ is the last index satisfying $\omega_{j'} \leq R^a + d^a$.

Example case is depicted in Fig 6.4 where $j' = 5$.

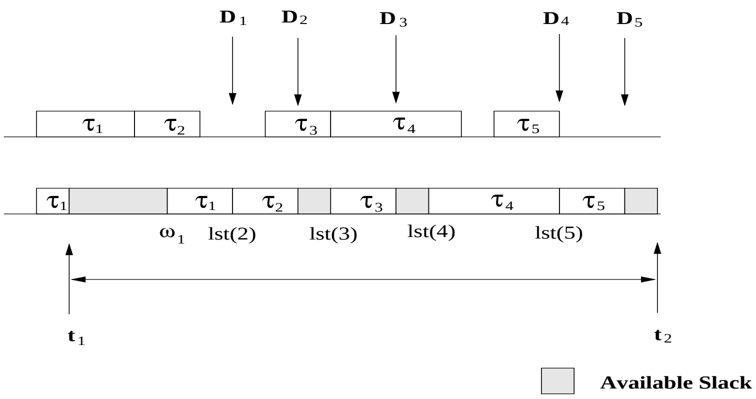

Figure 6.4: Obtaining maximum slack within a scheduling window of a hard aperiodic task A.

Acceptance Test for A Set of Hard Aperiodic Tasks

In this section, we address the problem of scheduling aperiodic tasks when several such tasks may arrive at any time instants. In this generalized scheduling model, we need to decide which scheduling policy is to be used for resolving the resource conflicts between the task instances in Γ and the aperiodic tasks, as well as the conflicts among the aperiodic tasks. For example, we can assign higher priorities to aperiodic tasks than the task instances in Γ as long as the latest start times of them are not reached, and use an earliest deadline first scheduling algorithm among the aperiodic tasks. However, this algorithm is not optimal as you can see from Fig 6.5. In this figure, the example task set is shown which is not schedulable according to the above approach. But, there exists a feasible schedule for this task set as is shown at the bottom of this figure. In the following subsections, we develop an optimal scheduling algorithm.

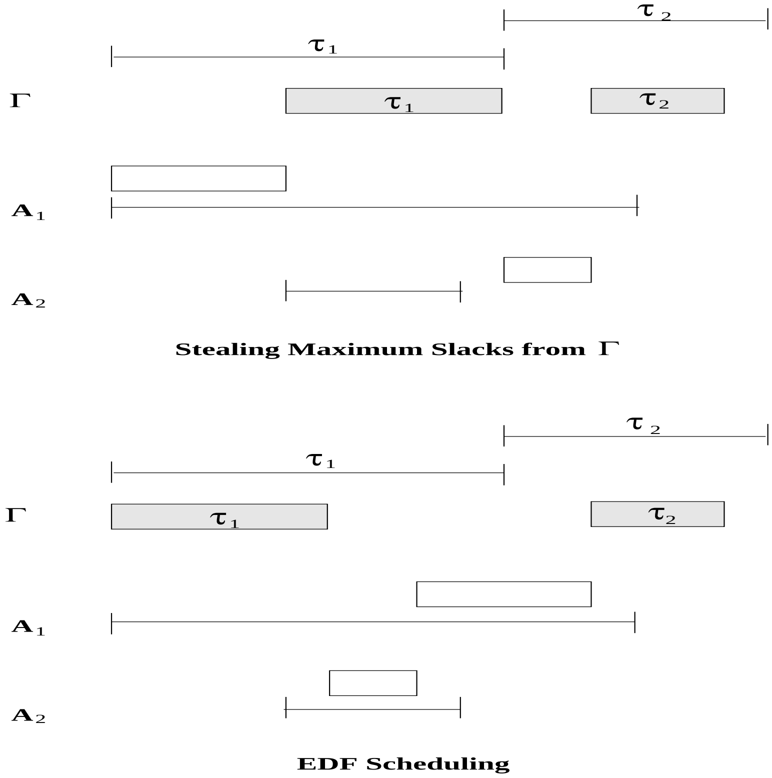

Figure 6.5: Example Schedules

Deriving Virtual Deadlines and Virtual Release Times

As a first step, we derive a virtual deadline and a virtual release time for each task instance τ_i in Γ. This process is necessary to enforce the total order on Γ when we employ EDF scheduling policy to resolve the resource conflicts in an unified manner for all the task instances.

A virtual deadline of τ_i is defined by the following recursive equation where D_i^o is the original deadline of τ_i:

$$
\begin{aligned}
D_N &= D_N^o \\
D_i &= min(D_{i+1} - C_{i+1}, D_i^o) \text{ for } i = N-1, N-2, \ldots, 1
\end{aligned}
$$

If a virtual deadline is missed by some task τ_i, then either the deadline of that task itself is missed or at least one of the following tasks misses its deadline. It is clear that the virtual deadline is always less than or equal to the original one and the virtual deadline D_i is always less than D_{i+1} by a difference of at least C_{i+1}, i.e. $D_i \leq D_{i+1} - C_{i+1}$.

Also, a virtual release time of τ_i is defined by the following recursive equation where R_i^o is the original release time of τ_i. Fig 6.6 explains the virtual release time and deadlines of the example tasks. Virtual release time is necessary to impose a total order on Γ when an EDF scheduling algorithm is used to schedule the tasks.

$$\begin{aligned} R_1 &= R_1^o \\ R_i &= max(R_{i-1}, R_i^o) \text{ for } i = 2, 3, \ldots, N \end{aligned}$$

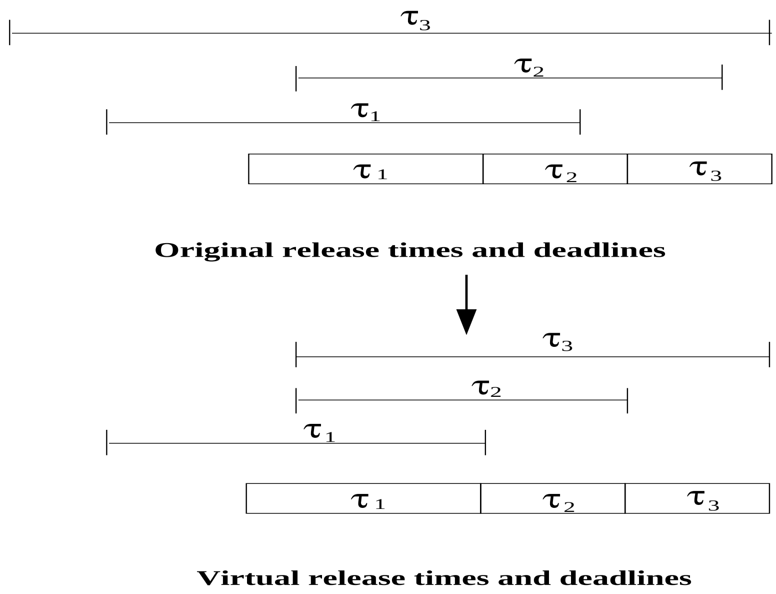

Figure 6.6: Deriving virtual deadlines and release times

This reduction of scheduling window of each task to $[R_i, D_i]$ from $[R_i^o, D_i^o]$ by the introduction of the virtual deadline is the result of imposing total order on Γ.

The following proposition establishes the equivalence between the original task set and the transformed task set with virtual deadline and release times in terms of the schedulability when an EDF is used to schedule Γ and an additional set of aperiodic tasks. Here, it is assumed that the total order of the task instances in Γ should be kept.

Proposition 6.1 *Γ and a set of additional aperiodic tasks are schedulable by EDF if and only if Γ with virtual deadlines and release times is schedulable with the additional aperiodic tasks by EDF.*

Proof Proof can be derived from the theorem in [15].

Optimal Scheduling Algorithm

In this section, the optimal scheduling algorithm is presented and its optimality is proved. We assume that the task instances in Γ have virtual deadlines and virtual release times instead of the original ones. The optimal scheduling algorithm assigns a higher priority to a task instance with a closer deadline in an unified manner.

At any time instant t, let $\mathcal{A}^{old}(t) = \{A_1^{old}, A_2^{old}, \ldots, A_m^{old}\}$ denote a set of *active* aperiodic tasks. Here, active aperiodic task is the aperiodic task that was accepted before t and still needs to be executed. It is obvious that the deadlines of these aperiodic tasks are greater than t. The tasks in $\mathcal{A}^{old}(t)$ are assumed to be sorted in their increasing order of deadlines. In addition, A_t^{new} denotes a newly arrived aperiodic task at time t. The first step of testing the acceptability of A_t^{new} is to insert A_t^{new} into $\mathcal{A}^{old}(t)$, thus producing $\mathcal{A}(t) = \{A_1, A_2, \ldots, A_{m+1}\}$ in which the tasks are sorted according to their deadlines in increasing order. Also, let $e_i^a(t)$ denote the processor time already spent for A_i up to time t. Obviously, $e_i^a(t) = 0$ if $A_i = A_t^{new}$. At this point, we derive the following lemmas and theorem which proves the optimality of the EDF scheduling algorithm proposed above.

The following lemma specifies the necessary condition for $\mathcal{A}(t)$ to be schedulable. Here, let D_i^a $(1 \leq i \leq m+1)$ denote a deadline of the i-th aperiodic task, A_i, in $\mathcal{A}(t)$.

Lemma 6.1 *Let $\mathcal{A}(t)$ denote a set of aperiodic tasks defined above. If there exists a feasible schedule for $\mathcal{A}(t)$, then*

$$\forall\, 1 \leq i \leq m+1 \;::\; \Omega(t, D_i^a) \geq \sum_{j=1}^{i} (c_j^a - e_j^a(t)) \tag{6.2}$$

Proof Suppose (6.2) is not satisfied for some $1 \leq k \leq m+1$, then

$$\Omega(t, D_k^a) < \sum_{j=1}^{k} (c_j^a - e_j^a(t))$$

This means that the processor demand in $[t, D_k^a]$ required by $\mathcal{A}(t)$ exceeds the maximum processor time in $[t, D_k^a]$ available for $\mathcal{A}(t)$. The un-schedulability of $\mathcal{A}(t)$ follows.

Lemma 6.2 *Let $\mathcal{A}(t)$ denote a set of aperiodic tasks defined above. Then $\mathcal{A}(t)$ can be scheduled under the proposed EDF if*

$$\forall\, 1 \leq i \leq m+1 \;::\; \Omega(t, D_i^a) \geq \sum_{j=1}^{i} (c_j^a - e_j^a(t))$$

Proof The proof can be easily derived from the theorems 3.2 and 3.3 in the paper by Chetto *et al.* [13].

Theorem 6.1 *Let $\mathcal{A}(t)$ denote a set of aperiodic tasks defined above. Then the proposed EDF scheduling algorithm is optimal and the schedulability condition is:*

$$\forall\, 1 \leq i \leq m+1 \;::\; \Omega(t, D_i^a) \geq \sum_{j=1}^{i} (c_j^a - e_j^a(t))$$

Proof From Lemma 6.1 and Lemma 6.2, this theorem follows.

Clearly, the condition of the above theorem can be checked within $O(M + m)$ by utilizing the formula (6.1) where M denotes the total number of task instances in Γ whose deadlines are greater than t and less than or equal to D_{m+1}^a, i.e., the task instances in Γ which may be executed within the range $[t, D^a_{m+1}]$. The first step is to insert the newly arrived aperiodic task into the set of active aperiodic tasks so that the aperiodic tasks are ordered in increasing order of their deadlines. Then, the maximum slack times, $\Omega(t, D^a_i)$, are found from $i = 1$ to $i = m + 1$ by making use of $\Omega(t, D^a_{i-1})$ already found.

If multiple aperiodic tasks arrive at t, we have to give priorities to these aperiodic tasks to decide which one has to be accepted and scheduled first. In this case, the above acceptance test is repeated for each aperiodic task from the one with highest priority to the one with lowest importance. The total complexity in this case is $O(K(M + m))$ where K denotes the number of aperiodic tasks arrived at t.

6.3.2 Sporadic Task Scheduling

One of the drawbacks of time-based scheduling scheme is that the sporadic task scheduling becomes very difficult. The algorithm to transform a sporadic task to an equivalent pseudo-periodic task has been proposed by *Al Mok* [38]. From the definition of the sporadic tasks, the events which invoke the sporadic task instances may occur at any time instant with the minimum inter-arrival time, δ. And, once the task is invoked, it has to be finished within its relative deadline from the invocation time, d^s. The first step of the transformation is to decide the relative deadline of the pseudo-periodic task, d^p, which is less than or equal to d^s. And then, the period, prd^p, of the pseudo task is found from the equation $prd^p = \min(d^s - d^p + 1, \delta)$. This is justified from the worst case scenario which can be seen in Figure 6.7.

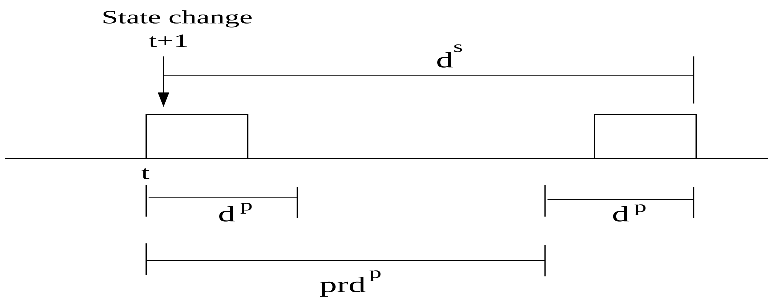

Figure 6.7: Worst Case for Deadline Determination

However, this approach may lead to significant under-utilization of the processor time, especially when d^s is small compared to δ, since a great amount of processor time has to be reserved statically at pre-runtime for servicing dynamic requests from sporadic tasks. This is well explained in Fig 6.8 through a simple example where an equivalent periodic task is to be found from a sporadic task whose worst case execution time is $c^s = 4$, whose relative deadline is $d^s = 8$, and whose minimum inter-arrival time is $\delta = 8$. If we employ *Mok*'s algorithm, the corresponding periodic task has a worst case execution time $c^p = c^s = 4$, a relative deadline $d^p = 4(\leq d^s)$, and a period $prd^p = \min(d^s - d^p + 1, \delta) = 5$. The processor utilization of this new periodic task is $4/5 = 0.8$.

In our proposed scheduling approach, the incremental scheduling of hard periodic tasks and sporadic tasks may be decomposed into two steps. We assume that the initial schedule of task instances is given in a scheduling window $[0, L]$ as in the previous sections. Then, the release times and deadlines of those task instances are transformed into virtual ones as was done in Section 6.3.1. And at runtime, every time new sporadic task arrives, the schedulability check is performed to see if

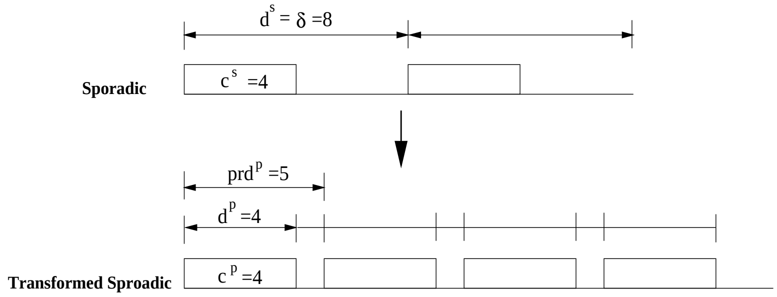

Figure 6.8: Under-utilization of the transformed sporadic task

the already accepted tasks and this new sporadic tasks can be scheduled using the EDF scheduling algorithm. And at runtime, the hard task instances from the schedule and the sporadic tasks are scheduled according to EDF. This can be viewed as merging two task instance streams, one from hard tasks and the other from sporadic tasks.

Extended Task Model

As in Section 6.3.1, an initial schedule of task instances is assumed to be given in an scheduling window $[0, L]$ and denoted as Γ. Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_N\}$ be a set of task instances where τ_i appears earlier than τ_{i+1}. Each τ_i has a following set of parameters in the schedule.

- virtual release time R_i
- virtual deadline $D_i (\leq L)$
- worst case execution time C_i

deadlines and virtual release times are obtained as in Section 6.3.1 from the original ones.

Let $\mathcal{S} = \{S_1, S_2, \ldots, S_{m_s}\}$ be a set of sporadic tasks which have to be scheduled with Γ. For each sporadic task S_i, the minimum inter-arrival time δ_i, the maximum execution time c_i^s, and the relative deadline d_i^s $(\leq \delta_i)$ are assumed to be given. It is also assumed that the S_is are ordered in increasing order of their relative deadlines, d_i^s, i.e., $d_i^s \leq d_{i+1}^s$. The objective of this section is to develop an optimal scheduling algorithm and its schedulability test for Γ and $\mathcal{S}$ together.

Some additional terms are defined in the following:

- Extended scheduling window for Γ and $\mathcal{S}$, $[0, LCM]$, where LCM is the least common multiple of L and the minimum inter-arrival times of the tasks in $\mathcal{S}$.
- N' denotes the total number of hard task instances scheduled in $[0, LCM]$. $N' = N(LCM/L)$ where $[0, L]$ is the original scheduling window.
- Extended schedule in an extended scheduling window $[0, kLCM]$ is found by repeating k times the schedule Γ and denoted as $k\Gamma$.

We need to check the schedule in an extended window $[0, 2LCM]$ to verify a schedulability of Γ and $\mathcal{S}$ according to the following scheduling model.

Scheduling Model

The CPU contention among tasks in Γ is resolved naturally from the total order among the tasks. This can be done by using an earliest deadline first scheduling algorithm and by using the virtual deadlines introduced earlier since $R_i \leq R_{i+1}$ and $D_i < D_{i+1}$. But, the mechanisms to resolve the resource contention between tasks from $\mathcal{S}$ and those from Γ should be provided to enable them to be scheduled at run-time. We assume that those contentions are also resolved through the same scheduling algorithm(EDF), leading to an uniform scheduling policy for $\mathcal{S}$ and Γ.

We denote a subset, $\{\tau_a, \tau_{a+1}, \ldots, \tau_b\}$, of Γ in $[0, LCM]$ as Υ if:

- $1 \leq a \leq b \leq N$
- $est(j+1) = est(j) + C_j$ for $j = a+1, a+2, \ldots, b-1$
- $est(a) > est(a-1) + C_{a-1}$ if $1 < a$
- $est(b+1) > est(b) + C_b$ if $b+1 \leq N$

In this case, we divide the set of task instances in $[0, LCM]$ into disjoint subsets, Υ_1, Υ_2, ..., Υ_υ, satisfying the above conditions. Let $est(\Upsilon_i)$ denote the earliest start time of the first task instance in Υ_i and let $eft(\Upsilon_i)$ denote the earliest finish time of Υ_i. Figure 6.9 shows an example case.

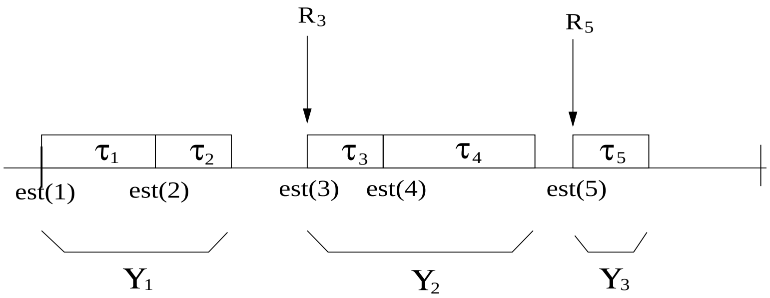

Figure 6.9: Υ found for an example task set

In addition, we define $\Omega'(t_1, t_2)$ $(0 \leq t_1 < LCM \ \wedge \ t_1 < t_2 < 2LCM)$ as the maximum slack time obtainable in $[t_1, t_2]$ under the assumption that from time 0 up to time instant t_1 task instances only from Γ have been executed with their maximum execution times, i.e., tasks have started at their earliest start times and spent their worst case execution times. Then, $\Omega'(t_1, t_2)$ can be obtained as follows. First step is to find task instance τ_i satisfying:

$$est(i-1) + C_{i-1} \leq t_1 \ \wedge \ t_1 \leq est(i) + C_i$$

If $t_1 \leq est(1) + C_1$, then let $i = 1$. Then,

$$\begin{aligned} \Omega'(t_1, t_2) &= lst(i) - t_1 + \max(0, t_1 - est(i)) \\ &+ \sum_{k=i}^{j'-1} (lst(k+1) - lst(k) - C_k) + \max(0, t_2 - lst(j') - C_{j'}) \end{aligned} \quad (6.3)$$

where $j'(i \leq j')$ is the last index satisfying $lst(j') \leq t_2$. This process is similar to the one used in the acceptance test of aperiodic task in Section 6.3.1. An example case is depicted in Figure 6.10.

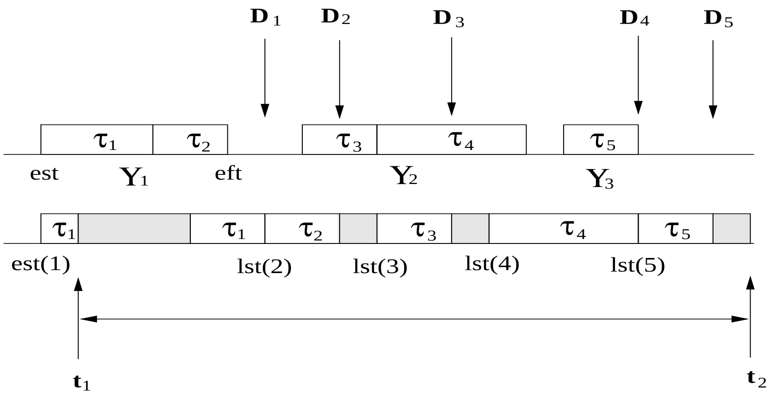

Figure 6.10: $\Omega'(t_1, t_2)$ for an example task set

Schedulability Test

The following proposition specifies the necessary condition for Γ and $\mathcal{S}$ to have a feasible schedule.

Proposition 6.2 *If there exists a feasible schedule for* Γ *and* $\mathcal{S}$*, then*

$$\forall i \in [1, v] \ :: \ \forall t \in [est(\Upsilon_i), est(\Upsilon_i) + LCM] \\ :: \ \Omega'(est(\Upsilon_i), t) \geq \sum_{k=1}^{m_s} c_k^s \cdot \lfloor \frac{(t - est(\Upsilon_i) + \delta_k - d_k^s)}{\delta_k} \rfloor \tag{6.4}$$

Proof: This is proved in the appendix.

The following theorem specifies the sufficient and necessary schedulability condition of the task set Γ and $\mathcal{S}$. The extended schedule in $[0, 2LCM]$ is assumed to be given.

Theorem 6.2 Γ *and* $\mathcal{S}$ *are schedulable according to EDF if and only if*

$$\forall i \in [1, v] \ :: \ \forall t \in [est(\Upsilon_i), est(\Upsilon_i) + LCM] \\ :: \ \Omega'(est(\Upsilon_i), t) \geq \sum_{k=1}^{m_s} c_k^s \cdot \lfloor \frac{(t - est(\Upsilon_i) + \delta_k - d_k^s)}{\delta_k} \rfloor \tag{6.5}$$

Proof: By proposition 6.2 and proposition B.5.

From the above proposition and a theorem, we can know that EDF is optimal for scheduling Γ and $\mathcal{S}$. Finally, we obtain an equivalent condition to (6.5) of the theorem 6.2, which enables us to reduce the complexity of the schedulability check. This corollary specifies that only the time instant which is equal to a deadline of some task instance in $\mathcal{S}$ needs to be examined at or after $est(\Upsilon_i)$ in checking the condition (6.5) of the theorem 6.2.

Corollary 6.1 *The following two conditions are equivalent to each other:*

$$(1) \quad \forall i \in [1, v] \ :: \ \forall t \in [est(\Upsilon_i), est(\Upsilon_i) + LCM] \\ :: \ \Omega'(est(\Upsilon_i), t) \geq \sum_{k=1}^{m_s} c_k^s \cdot \lfloor \frac{(t - est(\Upsilon_i) + \delta_k - d_k^s)}{\delta_k} \rfloor$$

$$(2)\quad \forall i \in [1, v] \;::\; \forall d_j \in [est(\Upsilon_i), est(\Upsilon_i) + LCM]$$
$$::\; \Omega'(est(\Upsilon_i), d_j) \geq \sum_{k=1}^{m_s} c_k^s \cdot \lfloor \frac{(d_j - est(\Upsilon_i) + \delta_k - d_k^s)}{\delta_k} \rfloor$$

where d_j is the deadline of some task instance in $\mathcal{S}$.

Therefore, the total complexity of the schedulability check algorithm is reduced to $O(M')$ where $M' = v(N' + \sum_{i=1}^{m_s}(LCM/\delta_i)) + \sum_{i=1}^{m_s}(LCM/\delta_i)\log(\sum_{i=1}^{m_s}(LCM/\delta_i))$. The first step is to obtain the deadlines(d_j) of the task instances from $\mathcal{S}$ in the window $[0, LCM]$ and sort them in increasing order. Then, for each $est(\Upsilon_i)$ $(1 \leq i \leq v)$, the second condition of the above corollary is checked in $O(N' + \sum_{i=1}^{m_s}(LCM/\delta_i))$ for the deadlines obtained in the first step. This process is similar to the one used in Section 6.3.1.

6.4 Summary

In this chapter, we addressed the issue of incremental scheduling on the basis of time-based scheduling scheme. The acceptance tests are developed for dynamically arriving aperiodic tasks, and for dynamically arriving sporadic tasks, respectively. A mixed scheduling policy was used such that the total ordering among static tasks is maintained. By making use of this property, we can extend the approach when there exist complex timing constraints between static tasks such as standard relative constraints.

Chapter 7

Conclusion

A new dynamic time-based scheduling scheme has been developed in this dissertation, and it is applied as a solution approach to several problems. In the new scheme, task attributes in the schedule may be represented as functions parameterized with information available at task dispatching time. By doing so, more freedom is available for a task dispatcher, and flexible resource management becomes possible at system operation time.

In Chapter 3 and 4, we addressed the problem of scheduling tasks in the presence of relative timing constraints in addition to release time and deadline constraints. Applying dynamic time-based scheduling scheme as a solution approach to this problem enables us not only to check the schedulability of a given cyclically constrained job set, but also to flexibly manage slack times at system operation time.

In Chapter 5, we addressed the problem of designing a dynamic temporal controller for linear time-invariant control systems. In dynamic temporal control technique, the fixed sampling period assumption is relaxed and sampling periods are adaptively decided based on current physical system state. It is shown that this new technique allows us to greatly reduce the computational resource requirement while maintaining the quality of control. When multiplexing multiple concurrent control tasks, especially when a transient overload has occurred, this new scheme provides a sound basis for increasing the system performance by efficiently distributing computational powers to tasks. This technique may be implemented by applying the dynamic time-based scheduling scheme, for example, by parameterizing task execution mode.

Finally, in Chapter 6, incremental scheduling problem is addressed on the basis of time-based scheduling scheme. That is, the total ordering among static tasks is maintained during system operation time, while dynamic tasks are executed in slack times available from static tasks. Only release time and deadline constraints are assumed to exist, and EDF is assumed to be used in resolving resource contention between dynamic(static) and dynamic tasks.

It is shown in this dissertation that dynamic time-based scheduling scheme may be effectively used as solution approaches to the problems in dynamic real-time systems.

7.1 Future Research

In this dissertation, a new dynamic time-based scheduling scheme is presented and its applicability has been shown through examples. In the presence of relative timing constraints, each entry in the dynamic calendar is parameterized with start or finish times of previous task instances. However, this restriction may be removed and an entry in the dynamic calendar may be an arbitrary function parameterized with any information available to the system. With this generalization, other extensions may be possible, especially in the presence of inter-task dependencies, or fault-tolerance requirements. Clearly, such functions lead to a highly state dependent dynamic schedules.

For example, the dynamic time-based scheduling scheme may be applied to cope with transient overloads that occur in many real-time systems [4]. In fixed priority-based systems, some work has been done on this issue [44]. However, as far as we know, no systematic work has been done on this, especially on time-based scheduling scheme. Dynamic time-based scheduling scheme seems to be an appropriate framework for this problem.

In Chapter 3 and 4, it is assumed that task order remains fixed throughout the system operation time. When a new task is to be added to a schedule, the original order may no longer be the best or most appropriate. In the presence of relative timing constraints, a new task order generated at run-time should be validated such that every timing constraints will be satisfied. This may require $O(n^2N^3)$ time in the worst case if our algorithm is applied. But, if a few task instances in the near future are allowed to change their orders, it may be possible to develop an algorithm with less complexity by utilizing that fact.

In Chapter 3 and 4, it is also assumed that a total ordering among tasks is found at pre-runtime by an off-line scheduler. Previous work by Cheng *et al.* [11] and Mok *et al.* [37] use a heuristic approach called *smallest latest start time first* to schedule task instances with relative constraints. However, their heuristics don't fully reflect the relative timing constraints. Improved heuristic functions may be developed if the constraint graph structure is utilized.

We considered the scheduling of tasks in uni-processor systems where tasks may have relative timing constraints. However, if we want to extend the dynamic dispatching approach to distributed systems, where tasks located in different nodes may have relative constraints, several issues have to be addressed further such as what kind of information have to be sent out to other nodes, and how parametric functions can be found.

In Chapter 4, a new controller design method is presented while its implementation issues were not addressed. As was mentioned in Chapter 4, dynamic time-based mechanism may be utilized to implement the scheme by creating a variable for each task instance designating its execution mode, i.e., whether that specific instance will be invoked or not. More work needs to be done on how the parametric functions can be found in this case.

In Chapter 6, the solution approach is found under the assumption that every task is preemptable. An extension of the work needs to be made for non-preemptive tasks.

Appendix A

A.1 Proofs for Chapter 4

Proof of Lemma 4.1: It is obvious that there exists an one-to-one correspondence between an edge pair set in $G(f_a)$ from which a new edge will be created after f_a is eliminated, and a constraint in $Sched(e_a)$ to be changed after eliminating e_a. Also, it is clear that a new constraint created in $Sched(s_a)$ will correspond to a new edge created in $G(s_a)$. Therefore, $Elim(G(f_a), f_a)$ is equal to $G(s_a)$.

Proof of Lemma 4.2: The proof for this lemma is similar to that of Lemma 4.1, and is omitted.

Proof of Proposition 4.2: Let π be a negative weight restricted cycle in $G(f_N)$ satisfying:

- no restricted cycle appears as a proper sub-cycle of π.

If there exists a negative weight restricted cycle in $G(f_N)$, then π also exists in $G(f_N)$. Also, let y be a node in π that appears first in a sequence $< v_0, s_1, f_1, \ldots, s_N, f_N >$. Then, π can be denoted as

$$< y \xrightarrow{w_1} v_1 \xrightarrow{w_2} v_2 \ldots v_i \xrightarrow{w_{i+1}} y >$$

where $\sum_{j=1}^{i+1} w_j < 0$. By eliminating nodes that lie after y in the node sequence $< v_0, s_1, f_1, \ldots, s_N, f_N >$, we will obtain a negative weight edge $y \xrightarrow{w'} y$ where $w' < 0$. This is clear from the path preserving property of node elimination algorithms. Then, from the equivalence relationship between constraint graphs and predicates, a contradiction is obtained during the elimination of the variables from $Sched$. Therefore, $Sched$ is equal to **False**. ■

Proof of Lemma 4.3: Claim 1: If $y \xrightarrow{w} z \in G(f_a)$ holds where $y \neq z$, then there exists an acyclic[1] restricted path $y \overset{w'}{\leadsto} z$ in $G(f_N)$ where $w' \leq w$ and all its intermediate nodes belong to $\{s_{a+1}, f_{a+1}, \ldots, s_N, f_N\}$.

If $v = f_N$, then the claim holds. Suppose that there exists an edge $y \xrightarrow{w} z$ in $G(f_a)$ where $1 \leq a \leq N-1$.

[1]For a case when $y = z$, it can be similarly shown that a restricted path without any intermediate restricted cycle(i.e., excluding y and z) is obtained, even though the resulting restricted path is not acyclic.

Assume that there exists an acyclic restricted path in $G(f_b)$ with a weight sum w, $a \le b \le N-1$,

$$< y \xrightarrow{\mathrm{w_{b,1}}} v_1 \xrightarrow{\mathrm{w_{b,2}}} v_2 \ldots \xrightarrow{\mathrm{w_{b,i}}} v_i \xrightarrow{\mathrm{w_{b,i+1}}} z > \tag{A.1}$$

where $i \ge 0$, and $v_j \in \{s_{a+1}, f_{a+1}, \ldots, s_b, f_b\}$ for $1 \le j \le i$. If all edges constituting this path exist in $G(f_{b+1})$ with same weights, then there exists an acyclic restricted path in $G(f_{b+1})$ with a weight sum w where all its intermediate nodes belong to $\{s_{a+1}, f_{a+1}, \ldots, s_{b+1}, f_{b+1}\}$. So, assume that at least one of these edges is created in $G(f_b)$ just after eliminating f_{b+1} and s_{b+1} from $G(f_{b+1})$. Let $\mathcal{J} = \{j_1, j_2, \ldots, j_k\}$, where $1 \le k \le i+1$ and $1 \le j_l \le i+1$ for $1 \le l \le k$, denote an index set of edges in the above path which are newly created in $G(f_b)$. The indices in $\mathcal{J}$ is assumed to be increasing. Each edge $v_{j_l-1} \xrightarrow{\mathrm{w_{b,j_l}}} v_{j_l}$, for $1 \le l \le k$, is created[2] just after f_{b+1} and s_{b+1} are eliminated from $G(f_{b+1})$.

Fact 1: In $G(f_{b+1})$ the weight of an edge $s_{b+1} \rightarrow f_{b+1}$ is equal to l_{b+1}, and the weight of $f_{b+1} \rightarrow s_{b+1}$ is equal to $-u_{b+1}$.

If the fact is not true, then a contradiction should have been derived, which is against the assumption.

From the node elimination algorithm we know that the edge $v_{j_l-1} \xrightarrow{\mathrm{w_{b,j_l}}} v_{j_l}$ is created from one of the following restricted paths in $G(f_{b+1})$ whose weight sum is w_{b,j_l}:

1. $< v_{j_l-1} \xrightarrow{\mathrm{w^1_{b+1,j_l}}} s_{b+1} \xrightarrow{\mathrm{w^2_{b+1,j_l}}} v_{j_l} >$

2. $< v_{j_l-1} \xrightarrow{\mathrm{w^1_{b+1,j_l}}} f_{b+1} \xrightarrow{\mathrm{-u_{b+1}}} s_{b+1} \xrightarrow{\mathrm{w^2_{b+1,j_l}}} v_{j_l} >$

3. $< v_{j_l-1} \xrightarrow{\mathrm{w^1_{b+1,j_l}}} s_{b+1} \xrightarrow{\mathrm{l_{b+1}}} f_{b+1} \xrightarrow{\mathrm{w^2_{b+1,j_l}}} v_{j_l} >$

4. $< v_{j_l-1} \xrightarrow{\mathrm{w^1_{b+1,j_l}}} f_{b+1} \xrightarrow{\mathrm{-u_{b+1}}} s_{b+1} \xrightarrow{\mathrm{l_{b+1}}} f_{b+1} \xrightarrow{\mathrm{w^2_{b+1,j_l}}} v_{j_l} >$

We can extend the path in (A.1) into a path in $G(f_{b+1})$ by replacing each edge in (A.1) with an index j_l by one of the above paths via s_{b+1} and f_{b+1}.

If $k = 1$, i.e., only one edge is created after eliminating f_{b+1} and s_{b+1} from $G(f_{b+1})$, then it is obvious that the extended path is also a restricted path with a weight w in $G(f_{b+1})$. So, assume that $k \ge 2$. In this case, there exists a cycle in the extended path.

First, consider two edges, $v_{j_1-1} \xrightarrow{\mathrm{w_{b,j_1}}} v_{j_1}$ and $v_{j_2-1} \xrightarrow{\mathrm{w_{b,j_2}}} v_{j_2}$. For all 16 possible combinations of the above 4 paths from which these two edges will be created, a restricted cycle is obtained after extending these two edges in (A.1). For example, if both of these two edges are created from the paths of the form 4, then the extended path will be of the following form:

$$\begin{aligned} < \ y \ &\rightarrow v_1 \rightarrow v_2 \ldots v_{j_1-1} \rightarrow \\ &< f_{b+1} \rightarrow s_{b+1} \rightarrow f_{b+1} \rightarrow v_{j_1} \ldots v_{j_2-1} \rightarrow f_{b+1} > \\ &\rightarrow s_{b+1} \rightarrow f_{b+1} \rightarrow v_{j_2} \ldots v_i \rightarrow z > \end{aligned}$$

The inner path, $< f_{b+1} \rightarrow s_{b+1} \rightarrow f_{b+1} \rightarrow v_{j_1} \ldots v_{j_2-1} \rightarrow f_{b+1} >$, is a restricted cycle, since the sub-path $< v_{j_1} \ldots v_{j_2-1} >$ is a restricted path and neither s_{b+1} nor f_{b+1} appears in this sub-path.

[2]For the purpose of convenience v_0 denotes a node y, and v_{i+1} denotes a node z.

Then, from Proposition 4.2 the weight sum of this restricted cycle is non-negative. If it is negative, then a **False** should have been derived during eliminating the nodes in $\{f_N, s_N, \ldots, f_{a+1}, s_{a+1}\}$, which is a contradiction to the assumption. Therefore, if we reduce this restricted cycle into a single node f_{b+1}, then we obtain the following restricted path whose weight sum is less than or equal to w:

$$< y \to v_1 \to v_2 \ldots v_{j_1-1} \to f_{b+1} \to s_{b+1} \to f_{b+1} \to v_{j_2} \ldots v_i \to z >$$

As a result, two edges, $v_{j_1-1} \xrightarrow{\mathrm{w_{b,j_1}}} v_{j_1}$ and $v_{j_2-1} \xrightarrow{\mathrm{w_{b,j_2}}} v_{j_2}$, are merged into one sub-path

$$v_{j_1-1} \to f_{b+1} \to s_{b+1} \to f_{b+1} \to v_{j_2}$$

Similarly, for other combinations for two edges, $v_{j_1-1} \xrightarrow{\mathrm{w_{b,j_1}}} v_{j_1}$ and $v_{j_2-1} \xrightarrow{\mathrm{w_{b,j_2}}} v_{j_2}$, the similar results can be obtained.

If we continue this merging process for an edge, $v_{j_3-1} \xrightarrow{\mathrm{w_{b,j_3}}} v_{j_3}$, and for the sub-path $< v_{j_1-1} \to f_{b+1} \to s_{b+1} \to f_{b+1} \to v_{j_2} >$ found above, we will obtain a merged acyclic sub-path from v_{j_1-1} to v_{j_3} through f_{b+1} or s_{b+1}.

Therefore, after $k-1$ iterations of the above process, we will obtain an acyclic restricted path in $G(f_{b+1})$ whose intermediate nodes belong to $\{s_{a+1}, f_{a+1}, \ldots, s_{b+1}, f_{b+1}\}$ and whose weight sum is less than or equal to w.

Therefore, by inductively applying the above argument, we know that there exists an acyclic restricted path in $G(f_N)$ whose intermediate nodes belong to $\{s_{a+1}, f_{a+1}, \ldots, s_N, f_N\}$ and whose weight sum is less than or equal to w.

Claim 2: If there exists an acyclic [3] restricted path $y \overset{\mathrm{w}}{\leadsto} z$ in $G(f_N)$ whose intermediate nodes belong to $\{s_{a+1}, f_{a+1}, \ldots, s_N, f_N\}$, then $y \xrightarrow{\mathrm{w'}} z \in G(f_a)$ holds where $w' \le w$.

The proof for this claim is similar to that for Proposition 4.2, and is omitted.

From claim 1 and 2 the lemma is proved. ■

Proof of Corollary 4.1: Suppose that an edge $y \to z$ exists in $G(v)$. If $v = f_a$ for some a, then from Lemma 4.3 it is obvious that there exists a path $y \leadsto z$ in $G(f_N)$ whose intermediate nodes belong to $\{v', \ldots, s_N, f_N\}$. So, assume that $v = s_a$ for some a in $[1, N]$.

If there exists an edge from y to z in $G(f_a)$, then the condition 2 holds. Hence, further assume that an edge $y \to z$ is created just after eliminating f_a from $G(f_a)$. From the node elimination algorithm, the edge is created from either of the following paths:

1. $y \to f_a \to s_a$
2. $s_a \to f_a \to z$

From Lemma 4.3 we know that there exist two acyclic restricted paths whose intermediate nodes belong to $\{s_{a+1}, f_{a+1}, \ldots, s_N, f_N\}$. By merging these paths, we obtain a path from y to z whose intermediate nodes belong to $\{f_a, s_{a+1}, f_{a+1}, \ldots, s_N, f_N\}$. ■

[3] For a case when $y = z$, it can be similarly shown that the claim holds for a restricted path without any intermediate restricted cycle(i.e., excluding y and z).

Proof of Proposition 4.3: If there exists an edge connecting s_i^j and v in $G^{1,k}(f_N^k)$, then it is obvious that v belongs to a node set P. So, assume that there exists no such edge in $G^{1,k}(f_N^k)$.

Two cases must be considered.

Case 1: $v \xrightarrow{w} s_i^j \in G^{1,k}(s_i^j)$

From Corollary 4.1 there exists a path from v to s_i^j in $G^{1,k}(f_N^k)$ whose intermediate nodes belong to $\{f_i^j, \ldots, s_N, f_N\}$. Note that this path has at least one intermediate node. From the definition of a crossing edge set $\Phi^{1,k}(s_i^j)$, it is clear that $v \in PrecNode(\Phi^{1,k}(s_i^j))$.

Case 2: $s_i^j \xrightarrow{w} v \in G^{1,k}(s_i^j)$

Similarly, the proposition can be proved in this case. ∎

Proof of Proposition 4.4: Suppose that v belong to $Node(\Psi^{1,k}(s_i^j))$. Then, there exist an edge $v' \in \{v_0, s_1^1, f_1^1, \ldots, s_i^j\}$ such that an edge $v \rightarrow v'$ exists in $G(s_i^j)$. Then from Corollary 4.1, we know that there exists a path $v \rightsquigarrow v'$ in $G^{1,k}(f_N^k)$ where all intermediate nodes in the path belong to $\{f_i^j, \ldots, s_N^k, f_N^k\}$. From the definition of $\Psi^{1,k}(s_i^j)$ there exist two edges $v \rightarrow v_1$ and $v_2 \rightarrow v'$ in $v \rightsquigarrow v'$ where v_1 and v_2 belong to $\{f_i^j, \ldots, s_N^k, f_N^k\}$. Note that v_1 may be equal to v_2. This means that v is an element of $PrecNode(\Phi^{1,k}(s_i^j))$. Thus, $Node(\Psi^{1,k}(s_i^j)) \subseteq PrecNode(\Phi^{1,k}(s_i^j))$ is proved. The second assertion, $Node(\Psi^{1,k}(f_i^j)) \subseteq PrecNode(\Phi^{1,k}(f_i^j))$, can be proved in a similar way. Also, from these we know that a maximum number of edges in $\Psi^{1,k}(f_N^j)$, $1 \leq j \leq k-1$, is less than or equal to $n(n-1)$, since n is the number of nodes in $PrecNode(\Phi^{1,2}(f_N^1))$. ∎

Proof of Proposition 4.5: Claim 1: If there exists an edge from v_1 to v_2 in $\Psi^{1,k}(f_N^{j-1})$, then there also exists an edge from $g_{(1)}(v_1)$ to $g_{(1)}(v_2)$ in $\Psi^{1,k}(f_N^j)$.

First suppose that $v_1 \rightarrow v_2 \in \Psi^{1,k}(f_N^{j-1})$ where $1 \leq j-1 \leq k-3$. Then, from the definition of a created edge set, there exists a path from v_1 to v_2 that has at least one intermediate node and whose intermediate nodes belong to $\{s_1^j, f_1^j, \ldots, s_N^k, f_N^k\}$. By applying a technique similar to the one used in the claim 1 of the proof for Lemma 4.3, we can reduce this path into an acyclic restricted path from v_1 to v_2 that has at least one intermediate node. Let this reduced path be denoted as $< v_1 \rightarrow x_1 \rightarrow x_2 \ldots \rightarrow x_l \rightarrow v_2 >$, $l \geq 1$, where every intermediate node x_h $(1 \leq h \leq l)$ belongs to $< s_1^j, f_1^j, \ldots, s_N^k, f_N^k >$. If all nodes x_h, $1 \leq h \leq l$, belong to $\{s_1^j, f_1^j, \ldots, s_N^{k-1}, f_N^{k-1}\}$, then it is clear from the cyclic nature of constraint graphs that there exists an acyclic restricted path from $g_{(1)}(v_1)$ to $g_{(1)}(v_2)$ in $G^{1,k}(f_N^k)$ whose intermediate nodes belong to $\{s_1^{j+1}, f_1^{j+1}, \ldots, s_N^k, f_N^k\}$.

Hence, assume that there exists at least one x_m, $1 \leq m \leq l$, that belongs to $\{s_1^k, f_1^k, \ldots, s_N^k, f_N^k\}$. Note that $x_1, x_l \in \{s_1^j, f_1^j, \ldots, s_N^j, f_N^j\}$. There are two possible cases to be considered:

1. x_1 is located later than x_l in the node sequence $< s_1^j, f_1^j, \ldots, s_N^j, f_N^j >$.

   - In this case there exists an acyclic restricted path $< v_1 \rightarrow x_1 \rightsquigarrow x_l \rightarrow v_2 >$ whose intermediate nodes belong to $\{s_1^j, f_1^j, \ldots, s_N^j, f_N^j\}$. This is because every node in constraint graphs has an edge to its previous node in the node sequence $< s_1^1, f_1^1, \ldots, s_N^k, f_N^k >$. In other words, $g_{(1)} < v_1 \rightarrow x_1 \rightsquigarrow x_l \rightarrow v_2 >$ is an acyclic restricted path from $g_{(1)}(v_1)$ to $g_{(1)}(v_2)$ in $G^{1,k}(f_N^k)$ whose intermediate nodes belong to $\{s_1^{j+1}, f_1^{j+1}, \ldots, s_N^k f_N^k\}$.[4]

[4] $g_{(a)} < y_1 \rightarrow y_2 \ldots \rightarrow y_i >$ is defined to be $< g_{(a)}(y_1) \rightarrow g_{(a)}(y_2) \ldots \rightarrow g_{(a)}(y_i) >$.

Hence, from Lemma 4.3 there exists an edge $g_{(1)}(v_1) \rightarrow g_{(1)}(v_2)$ in $G^{1,k}(f_N^j)$. Because there exists a path from $g_{(1)}(v_1)$ to $g_{(1)}(v_2)$ satisfying the condition given in definition of a created edge set, this edge belongs to $\Psi^{1,k}(f_N^j)$

2. x_1 is located before x_l.

   - Let the reduced path be denoted as $< v_1 \rightarrow x_1 \rightsquigarrow x_i \rightsquigarrow x_m \rightarrow x_l \rightarrow v_2 >$ where x_i is a first node appearing in this path that lies after x_1 in the node sequence $< s_1^j, f_1^j, \ldots, s_N^k, f_N^k >$. Note that $x_i \in \{s_1^j, f_1^j, \ldots, s_N^{j+1}, f_N^{j+1}\}$. Again, since $j+1 \leq k-1$ and every node has a path to its predecessor in the node sequence, there exists an acyclic restricted path $< v_1 \rightarrow x_1 \rightsquigarrow x_i \rightsquigarrow x_l \rightarrow v_2 >$ that doesn't have a node for a job in Γ^k. Hence, there exists an acyclic restricted path $g_{(1)} < v_1 \rightarrow x_1 \rightsquigarrow x_i \rightsquigarrow x_l \rightarrow v_2 >$ whose intermediate nodes belong to $\{s_1^{j+1}, f_1^{j+1}, \ldots, s_N^k, f_N^k\}$. This means that $g_{(1)}(v_1) \rightarrow g_{(1)}(v_2) \in G^{1,k}(f_N^j)$. Also, because the above path satisfies the definition for a created edge set, this edge belongs to $\Psi^{1,k}(f_N^j)$

<u>Claim 2:</u> If there exists an edge from v_3 to v_4 in $\Psi^{1,k}(f_N^j)$, then there also exists an edge from $g_{(-1)}(v_3)$ to $g_{(-1)}(v_4)$ in $\Psi^{1,k}(f_N^{j-1})$.

Suppose that there exists an edge from v_3 to v_4 in $\Psi^{1,k}(f_N^j)$. Then, from the definition of a created edge set, there exists a path from v_3 to v_4 that has at least one intermediate node and whose intermediate nodes belong to $\{s_1^{j+1}, f_1^{j+1}, \ldots, s_N^k, f_N^k\}$. By applying the technique in the claim 1 of the proof for Lemma 4.3, we can reduce this path into an acyclic restricted path from v_3 to v_4 that has at least one intermediate node. Let this path be denoted as $< v_3 \rightsquigarrow v' \rightsquigarrow v_4 >$ where v' belongs to $\{s_1^{j+1}, f_N^{j+1}, \ldots, s_N^k, f_N^k\}$. In this case, the path $g_{(-1)} < v_3 \rightsquigarrow v' \rightsquigarrow v_4 >$ is also an acyclic restricted path in $G(f_N^k)$ whose intermediate nodes belong to $\{s_1^j, f_N^j, \ldots, s_N^{k-1}, f_N^{k-1}\}$. Then, from Lemma 4.3 there exists an edge $g_{(-1)}(v_3) \rightarrow g_{(-1)}(v_4)$ in $G^{1,k}(f_N^{j-1})$. Also, because the path $g_{(-1)} < v_3 \rightsquigarrow v' \rightsquigarrow v_4 >$ satisfies the condition in the definition of a created edge set, this edge belongs to $\Psi^{1,k}(f_N^{j-1})$, too.

From Claim 1 and 2, we conclude that $\Psi^{1,k}(f_N^{j_1})$ is semi-homogeneous to $\Psi^{1,k}(f_N^{j_2})$ for $1 \leq j_1 \leq j_2 \leq k-2$. ■

Proof of Proposition 4.6: From Lemma 4.3 there exists a minimum weight acyclic restricted path $\pi_1 =< v_1 \overset{w}{\rightsquigarrow} v_2 >$ whose intermediate nodes belong to $\{s_1^{j+1}, f_1^{j+1}, \ldots, s_N^k, f_N^k\}$, and a minimum weight acyclic restricted path $\pi_2 =< g_{(-1)}(v_1) \overset{w'}{\rightsquigarrow} g_{(-1)}(v_2) >$ whose intermediate nodes belong to $\{s_1^j, f_1^j, \ldots, s_N^k, f_N^k\}$. Three cases must be examined:

<u>Case 1:</u> $v_1 \neq v_0$ and $v_2 \neq v_0$

In this case it is clear that w' is less than or equal to w, since the set of acyclic restricted paths from v_1 to v_2 in $G^{1,k}(f_N^k)$ whose intermediate nodes belong to $\{s_1^{j+1}, f_1^{j+1}, \ldots, s_N^k, f_N^k\}$ is a subset of a set of acyclic restricted paths from v_1 to v_2 in $G^{1,k}(f_N^k)$ whose intermediate nodes belong to $\{s_1^j, f_1^j, \ldots, s_N^k, f_N^k\}$.

<u>Case 2:</u> $v_1 = v_0$

The path $g_{(-1)}\pi_1$ is also an acyclic restricted path. The weight of a path $g_{(-1)}\pi_1$ is equal to $w - L$, since every edge weight in this new path is the same as that of corresponding edge in π_1

except for the first edge $v_0 \to g_{(-1)}(x_1)$ of $g_{(-1)}\pi_1$ where x_1 denotes the first node appearing after v_0 in π_1. The weight of this edge is L less than that of $v_0 \to x_1$ which is the first edge of π_1. This implies $w' \le w - L$ from Lemma 4.3.

Case 3: $v_2 = v_0$

The path $g_{(-1)}\pi_1$ is also an acyclic restricted path. The weight of a path $g_{(-1)}\pi_1$ is equal to $w + L$, since every edge weight in this new path is the same as that of corresponding edge in π_1 except for the last edge $g_{(-1)}(x_l) \to v_0$ of $g_{(-1)}\pi_1$. The weight of this edge is L more than the weight of $x_l \to v_0$ which is the last edge of π_1. This implies $w' \le w + L$ from Lemma 4.3. ■

Proof of Proposition 4.7: Note that two created edge sets, $\Psi^{1,k}(f_i^{j-1})$ and $\Psi^{1,k}(f_i^j)$, can be shown to be semi-homogeneous by employing similar proof to that for Proposition 4.5 where $2 \le j \le k-2$

The following claim is proved where i is any integer satisfying $1 \le i \le N$.

Claim 1: $\Psi^{1,k}(f_i^{j-1}) \sim \Psi^{1,k}(f_i^j)$

First, suppose that $v_1 \xrightarrow{w_1} v_2 \in \Psi^{1,k}(f_i^{j-1})$, where $v_1 \ne v_0, v_2 \ne v_0$. Consider a graph $G^{1,k}(f_N^{j-1})$. From Lemma 4.3, we can find a minimum weight acyclic restricted path within this graph, $\pi_1 = < v_1 \overset{w_1}{\leadsto} v_2 >$ whose intermediate nodes belong to $\{s_{i+1}^{j-1}, f_{i+1}^{j-1}, \ldots, s_N^{j-1}, f_N^{j-1}\}$. From the assumption of homogeneity between $\Psi^{1,k}(f_N^j)$ and $\Psi^{1,k}(f_N^{j-1})$, every edge $x_1 \to x_2$ in $G^{1,k}(f_N^{j-1})$, where $x_1, x_2 \in \{s_{i+1}^{j-1}, \ldots, s_N^{j-1}, f_N^{j-1}\}$, has the same weight as an edge $g_{(1)}(x_1) \to g_{(1)}(x_2)$ in $G^{1,k}(f_N^j)$. This one-to-one correspondence between created edge sets implies that an acyclic restricted path $g_{(1)}\pi_1$ has the same weight w_1 as that of π_1, and $g_{(1)}\pi_1$ is a minimum weight acyclic restricted path among the acyclic restricted paths in $G^{1,k}(f_N^j)$ whose intermediate nodes belong to $\{s_{i+1}^j, \ldots, s_N^j, f_N^j\}$. Hence, $g_{(1)}(v_1) \xrightarrow{w_1} g_{(1)}(v_2) \in G^{1,k}(f_i^j)$ holds from Lemma 4.3. Because $\Psi^{1,k}(f_i^{j-1})$ and $\Psi^{1,k}(f_i^j)$ are semi-homogeneous, this edge also belongs to $\Psi^{1,k}(f_i^j)$.

Second, suppose that $v_3 \xrightarrow{w_2} v_4 \in \Psi^{1,k}(f_i^j)$, where $v_3 \ne v_0, v_4 \ne v_0$. Consider a graph $G^{1,k}(f_N^j)$. From Lemma 4.3, we can find a minimum weight acyclic restricted path within this graph, $\pi_2 = < v_3 \overset{w_2}{\leadsto} v_4 >$ whose intermediate nodes belong to $\{s_{i+1}^j, f_{i+1}^j, \ldots, s_N^j, f_N^j\}$. Again, from the one-to-one correspondence between created edge sets, a path $g_{(-1)}\pi_2$ has the same weight w_2 as that of π_2, and the path is also a minimum weight acyclic restricted path among the acyclic restricted paths in $G^{1,k}(f_N^{j-1})$ whose intermediate nodes belong to $\{s_{i+1}^{j-1}, \ldots, s_N^{j-1}, f_N^{j-1}\}$. Hence, $g_{(-1)}(v_3) \xrightarrow{w_2} g_{(-1)}(v_4) \in G^{1,k}(f_i^{j-1})$ holds from Lemma 4.3. Because $\Psi^{1,k}(f_i^{j-1})$ and $\Psi^{1,k}(f_i^j)$ are semi-homogeneous, this edge also belongs to $\Psi^{1,k}(f_i^{j-1})$.

Therefore, the following is proved where $v_1 \ne v_0$ and $v_2 \ne v_0$:

$$(v_1 \xrightarrow{w} v_2 \in \Psi^{1,k}(f_i^{j-1})) \iff (g_{(1)}(v_1) \xrightarrow{w} g_{(1)}(v_2) \in \Psi^{1,k}(f_i^j))$$

For cases where one of v_1 or v_2 is equal to v_0, the condition 3 or 4 in the definition of homogeneous edge sets may be proved in a similar way to the above one by using the definition of homogeneity between created edge sets and Lemma 4.3.

Therefore, the Claim 1 is proved. Then, from the transitivity of homogeneous relations, it is clear that the following holds:

$$\forall l : 2 \le l \le j-1 \ :: \ \forall i : 1 \le i \le N \ :: \ \Psi^{1,k}(f_i^l) \sim \Psi^{1,k}(f_i^j)$$

Claim 2: $\forall l : 2 \leq l \leq j-1 \ :: \ \forall i : 1 \leq i \leq N \ :: \ \Psi^{1,k}(s_i^l) \sim \Psi^{1,k}(s_i^j)$

For fixed l and i, we know that $\Psi^{1,k}(f_i^l) \sim \Psi^{1,k}(f_i^j)$ holds from claim 1. From this homogeneity, it is clear that $\Psi^{1,k}(s_i^l) \sim \Psi^{1,k}(s_i^j)$ holds from node elimination algorithms. That is, $\Psi^{1,k}(f_i^l)$ is obtained after eliminating f_i^l from $G(f_i^l)$, and $\Psi^{1,k}(f_i^j)$ is obtained after eliminating f_i^j from $G(f_i^j)$. ■

Proof of Theorem 4.1: Let $G_b(V_b, E_b)$ denote a basis graph obtained from an initial constraint graph for a cyclically constrained job set.

Claim: If the Algorithm 4.3 applied to $G_b(V_b, E_b)$ doesn't terminate within $n^2 - n + 2$ loop iterations, then there exists a negative weight cycle in $G^{1,k}(f_N^k)$ for $k \gg n^2$.

Suppose that the algorithm doesn't terminate within $n^2 - n + 2$ loop iterations. From Proposition 4.5, we know that $\Psi^{1,k}(f_N^{k-2})$, $\Psi^{1,k}(f_N^{k-3})$, ..., $\Psi^{1,k}(f_N^1)$ are semi-homogeneous. Thus, $G_{out}^i(V_{b,1} \cup \{v_0\}, E_{out}^i)$, $2 \leq i \leq n^2 - n + 2$, are semi-homogeneous, too. This means that after each loop iteration for $i \geq 3$ in the algorithm, there exists at least one edge in $G_{out}^i(V_{b,1} \cup \{v_0\}, E_{out}^i)$, $3 \leq i \leq n^2 - n + 2$, whose weight has been reduced from the corresponding one in $G_{out}^{i-1}(V_{b,1} \cup \{v_0\}, E_{out}^{i-1})$. If not, then the algorithm should have been completed within $n^2 - n + 2$ loop iterations at step $3-(d)$, because homogeneous created edge sets are already found, which is against the assumption. For the purpose of clarity, each node $v_i (\in V_b)$ used in this proof will be denoted as v_i^j to represent that v_i belongs to a node set V_b in a graph $G_{in}^j(V_b, E_{in}^j)$, or to a node set $V_{b,1} \cup \{v_0\}$ of a graph $G_{out}^j(V_{b,1} \cup \{v_0\}, E_{out}^j)$.

Let $v_1^{n^2-n+2} \longrightarrow v_2^{n^2-n+2}$, $v_1, v_2 \in V_{b,1} \cup \{v_0\} (v_1 \neq v_2)$, denote one such edge in $G_{out}^{n^2-n+2}(V_{b,1} \cup \{v_0\}, E_{out}^{n^2-n+2})$ whose weight is less than that of the corresponding edge in $G_{out}^{n^2-n+1}(V_{b,1} \cup \{v_0\}, E_{out}^{n^2-n+1})$. Equivalently, from the cyclic operation performed at step $3-(f)$ in Algorithm 4.3 we can say that $v_1^{n^2-n+2} \longrightarrow v_2^{n^2-n+2}$ is an edge in $G_{out}^{n^2-n+2}(V_{b,1} \cup \{v_0\}, E_{in}^{n^2-n+2})$ whose weight is less than or equal to

- $w - 1$, if the edge doesn't connect v_0.
- $w - L - 1$, if the edge is from v_0.
- $w + L - 1$, if the edge is to v_0.

where w is a weight of an edge $(g_{(1)}(v_1))^{n^2-n+2} \longrightarrow (g_{(1)}(v_2))^{n^2-n+2}$ of $G_{in}^{n^2-n+2}(V_b, E_{in}^{n^2-n+2})$.

Let p_1 denote a minimum weight acyclic restricted path from $v_1^{n^2-n+2}$ to $v_2^{n^2-n+2}$ with a weight w_{12} in $G_{in}^{n^2-n+2}(V_b, E_{in}^{n^2-n+2})$ whose intermediate nodes belong to $V_{b,2}$. Note that no intermediate node, if there exists any, is equal to v_0. p_1 exists from Lemma 4.3. Then, after $(n^2 - n + 2)$-th loop iteration, the weight of $v_1^{n^2-n+2} \rightarrow v_2^{n^2-n+2}$ will be changed to w_{12} in $G_{out}^{n^2-n+2}(V_{b,1} \cup \{v_0\}, E_{out}^{n^2-n+2})$.

sub-claim 1: In $G_{in}^{n^2-n+2}(V_b, E_{in}^{n^2-n+2})$, p_1 has at least one edge connecting two different nodes that belong to $g_{(1)}(V_{b,1}) \cup \{v_0\}$.

Suppose the claim is not true. Then, p_1 is also a minimum weight acyclic restricted path from $v_1^{n^2-n+1}$ to $v_2^{n^2-n+1}$ with a weight w_{12} in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$, since only the weights of edges connecting two different nodes of $g_{(1)}(V_{b,1}) \cup \{v_0\}$ may be reduced after each loop iteration of the algorithm. This contradicts to the definition of the path p_1.

Then, the following is proved. Here, it is assumed that v_3, v_4 $(v_3 \neq v_4)$ belong to $g_{(1)}(V_{b,1}) \cup \{v_0\}$, and thus $g_{(-1)}(v_3)$, $g_{(-1)}(v_4)$ belong to $V_{b,1} \cup \{v_0\}$.

sub-claim 2: There exists at least one edge in p_1, $v_3^{n^2-n+2} \xrightarrow{w_{34}} v_4^{n^2-n+2}$, $v_3, v_4 \in g_{(1)}(V_{b,1}) \cup \{v_0\}$, satisfying

$$w_{34} < w'_{34}$$

where w'_{34} is a weight of an edge $v_3^{n^2-n+1} \longrightarrow v_4^{n^2-n+1}$ in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$.

Suppose that the claim is not true. Then, all edges lying in p_1 that connect two nodes of $g_{(1)}(V_{b,1}) \cup \{v_0\}$ don't satisfy the above condition. In other words, all edge weights of p_1 in $G_{in}^{n^2-n+2}(V_b, E_{in}^{n^2-n+2})$ are not reduced compared to the edge weights of p_1 in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$. This means that p_1 is also a minimum weight acyclic restricted path with a weight w_{12} in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$, which is clear from Proposition 4.6. From Lemma 4.3 this implies that the weight of $v_1^{n^2-n+1} \rightarrow v_2^{n^2-n+1}$ in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$ is equal to w_{12}. This contradicts to the definition of the path p_1. Therefore, sub-claim 2 is proved.

Hence, we know that in path p_1 there exists an edge $v_3 \xrightarrow{w_{34}} v_4$ whose weight is less than that of the corresponding edge $v_3^{n^2-n+1} \xrightarrow{w'_{34}} v_4^{n^2-n+1}$ in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$.

From the cyclic operation performed at step $3-(f)$ in Algorithm 4.3 and from Lemma 4.3, we know that there exists a minimum weight acyclic restricted path from $g_{(-1)}(v_3)$ to $g_{(-1)}(v_4)$ in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$ whose intermediate nodes belong to $V_{b,2}$ and which is equal to one of the following forms:

1. If $v_3 \neq v_0$ and $v_4 \neq v_0$,

$$(g_{(-1)}(v_3))^{n^2-n+1} \overset{w_{34}}{\rightsquigarrow} (g_{(-1)}(v_4))^{n^2-n+1}$$

2. If $v_3 = v_0$ and $v_4 \neq v_0$,

$$v_0^{n^2-n+1} \overset{w_{34}-L}{\rightsquigarrow} (g_{(-1)}(v_4))^{n^2-n+1}$$

3. If $v_3 \neq v_0$ and $v_4 = v_0$,

$$(g_{(-1)}(v_3))^{n^2-n+1} \overset{w_{34}+L}{\rightsquigarrow} v_0^{n^2-n+1}$$

Note that, if any edge weight in the above minimum weight acyclic restricted path is reduced, the weight of an edge $v_3 \rightarrow v_4$ in $G_{in}^{n^2-n+2}(V_b, E_{in}^{n^2-n+2})$ will also be reduced by at least the same amount after (n^2-n+1)-th loop iteration of Algorithm 4.3.

Hence, p_1 can be denoted as:

$$< v_1^{n^2-n+2} \rightsquigarrow v_3^{n^2-n+2} \xrightarrow{w_{34}} v_4^{n^2-n+2} \rightsquigarrow v_2^{n^2-n+2} >$$

where the edge $v_3^{n^2-n+2} \xrightarrow{w_{34}} v_4^{n^2-n+2}$ can be replaced by one of the above minimum weight paths. Then p_1 can be denoted as:

$$< v_1^{n^2-n+2} \rightsquigarrow < (g_{(-1)}(v_3))^{n^2-n+1} \rightsquigarrow (g_{(-1)}(v_4))^{n^2-n+1} > \rightsquigarrow v_2^{n^2-n+2} >$$

where the inner path, $< (g_{(-1)}(v_3))^{n^2-n+1} \rightsquigarrow (g_{(-1)}(v_4))^{n^2-n+1} >$, will be reduced to an edge $v_3^{n^2-n+2} \xrightarrow{w_{34}} v_4^{n^2-n+2}$ after (n^2-n+1)-th loop iteration if Algorithm 4.3 is applied to the above extended path.

Note that applying Algorithm 4.3 to this new path will produce an edge $v_1^{n^2-n+2} \xrightarrow{w_{12}} v_2^{n^2-n+2}$, and if some edge weight is reduced, w_{12} will be reduced, too.

From the above result and from $w_{34} < w'_{34}$, we know that the edge weight of $g_{(-1)}(v_3) \rightarrow g_{(-1)}(v_4)$ in a graph $G_{out}^{n^2-n+1}(V_{b,1} \cup \{v_0\}, E_{out}^{n^2-n+1})$ is:

- $w'_{34} - 1$ or less, if $v_3 \neq v_0$ and $v_4 \neq v_0$.
- $w'_{34} - L - 1$ or less, if $v_3 = v_0$.
- $w'_{34} + L - 1$ or less, if $v_4 = v_0$.

where w'_{34} is an edge weight of $v_3 \rightarrow v_4$ in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$.

This enables us to repeatedly apply the same procedure to a new minimum weight acyclic restricted path $g_{(-1)}(v_3) \rightsquigarrow g_{(-1)}(v_4)$ in $G_{in}^{n^2-n+1}(V_b, E_{in}^{n^2-n+1})$. Therefore, we obtain the following extension of path p_1:

$$\begin{aligned} < v_1^{n^2-n+2} \rightsquigarrow < (g_{(-1)}(v_3))^{n^2-n+1} &\rightsquigarrow\ < (g_{(-1)}(v_5))^{n^2-n} \rightsquigarrow (g_{(-1)}(v_6))^{n^2-n} > \\ &\rightsquigarrow\ (g_{(-1)}(v_4))^{n^2-n+1} > \rightsquigarrow v_2^{n^2-n+2} > \end{aligned}$$

where the intermediate nodes of $< (g_{(-1)}(v_5))^{n^2-n} \rightsquigarrow (g_{(-1)}(v_6))^{n^2-n} >$ in the above path belong to $V_{b,2}$ of $G_{in}^{n^2-n}(V_b, E_{in}^{n^2-n})$.

And, this extension may be continued until the following is obtained:

$$\begin{aligned} < v_1^{n^2-n+2} &\rightsquigarrow\ < (g_{(-1)}(v_3))^{n^2-n+1} \rightsquigarrow < (g_{(-1)}(v_5))^{n^2-n} \rightsquigarrow \ldots \\ &\rightsquigarrow\ < (g_{(-1)}(v_{2(n^2-n+1)-1}))^2 \rightsquigarrow (g_{(-1)}(v_{2(n^2-n+1)}))^2 > \rightsquigarrow \\ \ldots &\rightsquigarrow\ (g_{(-1)}(v_6))^{n^2-n} > \rightsquigarrow (g_{(-1)}(v_4))^{n^2-n+1} > \rightsquigarrow v_2^{n^2-n+2} > \end{aligned}$$

Consider the following set of node pairs in $V_{b,1} \cup \{v_0\}$ of $G_{in}^j(V_b, E_{in}^j)$, $2 \leq j \leq n^2 - n + 2$, that have been included in the extension of path p_1 at each iteration of the process.

$$\begin{aligned} \{(v_1^{n^2-n+2}, v_2^{n^2-n+2}), \quad &((g_{(-1)}(v_3))^{n^2-n+1}, (g_{(-1)}(v_4))^{n^2-n+1}), \ldots, \\ &((g_{(-1)}(v_{2(n^2-n+1)-1}))^2, (g_{(-1)}(v_{2(n^2-n+1)}))^2)\} \end{aligned}$$

Note that this set has $n^2 - n + 1$ node pairs. Because there exist n nodes in $V_{b,1} \cup \{v_0\}$, there may exist only $n^2 - n$ distinct node pairs. Hence, there should exist at least one node pair that appears twice in the above node pair set. Let $(v_{i_1}^j, v_{i_2}^j), (v_{i_3}^l, v_{i_4}^l)$, $l < j$, denote two such node pairs where $i_1 = i_3 \wedge i_2 = i_4$. Therefore, in the extension process of p_1 performed above, we should have encountered the following path:

$$< v_{i_1}^j \rightsquigarrow < v_{i_1}^l \rightarrow v_{i_2}^l > \rightsquigarrow v_{i_2}^j > \tag{A.2}$$

Because the extension process choose an edge $v_{i_1}^j \rightarrow v_{i_2}^j$ in $G_{in}^j(V_b, E_{in}^j)$ whose weight is less than $v_{i_1}^{j-1} \rightarrow v_{i_2}^{j-1}$ at the $(n^2 - n + 2 - j + 1)$-th iteration of Algorithm 4.3, we know that the weight of an edge $v_{i_1}^l \rightarrow v_{i_2}^l$ is greater than the weight of the edge $v_{i_1}^j \rightarrow v_{i_2}^j$ since $j > l$.

This implies that there exists a path that reduces the the edge weight of $v_{i_1}^j \rightarrow v_{i_2}^j$ from that of $v_{i_1}^l \rightarrow v_{i_2}^l$ after j-th loop iteration in Algorithm 4.3. Then, from Proposition 4.6 we know that, after $l + k(j - l)$ loop iteration in Algorithm 4.3 where $k \geq 1$, the edge weight of $v_{i_1} \rightarrow v_{i_2}$ in the resulting graph will be reduced from the corresponding edge weight in the graph found after $l + (k-1)(j-l)$ loop iteration. This means that the edge weight of $v_{i_1} \rightarrow v_{i_2}$ will be infinitely decreased. But, since every job has a release time and a deadline constraints, this repeated process will eventually create a negative weight cycle during the variable elimination process applied to a constraint graph for $sched^{1,\infty}$.

This contradicts to the assumption, and proves Claim 1 and the theorem. ■

Appendix B

B.1 Proofs for Chapter 6

The proof of theorem 6.2 is presented here.

Proposition B.1 *If Γ and $\mathcal{S}$ are schedulable then the following condition is satisfied:*

$$\forall i \in [1, v] \ :: \ \forall t \in [est(\Upsilon_i), est(\Upsilon_i) + LCM]$$
$$:: \ \Omega'(est(\Upsilon_i), t) \geq \sum_{k=1}^{m_s} c_k^s \cdot \lfloor \frac{(t - est(\Upsilon_i) + \delta_k - d_k^s)}{\delta_k} \rfloor$$

Proof: Suppose that $\mathcal{S}$ and Γ are schedulable and the above condition doesn't hold. Let t_v be the first time instant at which the condition is not satisfied. That is, the following is satisfied for some $i_v \in [1, v]$:

$$\Omega'(est(\Upsilon_{i_v}), t_v) < \sum_{k=1}^{m_s} c_k^s \cdot \lfloor \frac{(t_v - est(\Upsilon_{i_v}) + \delta_k - d_k^s)}{\delta_k} \rfloor$$

However, from this we can conclude that the task set is not schedulable when all the sporadic tasks start to be invoked at time $est(\Upsilon_{i_v})$ with their minimum inter-arrival times. This is because the processor demand by $\mathcal{S}$ in $[est(\Upsilon_{i_v}), t_v]$ exceeds the processor time in $[est(\Upsilon_{i_v}), t_v]$ available for tasks in $\mathcal{S}$. Therefore, if Γ and $\mathcal{S}$ are schedulable, the condition is satisfied.

We define a *busy period* for the given task, α, which belongs to Γ or $\mathcal{S}$ and denote it as $BP_\alpha = [a, f_\alpha]$ where f_α is the actual finish time of the task α at run-time. Let D_α denote a deadline of α. Then, let β be the *last* task satisfying the following conditions:

(1) $\beta \in \Gamma$ or $\beta \in \mathcal{S}$
(2) β starts its execution before f_α.
(3) β starts its execution at its release time r_β.
(4) no idle period exists between r_β and f_α.
(5) no task whose deadline is greater than D_α is executed between r_β and f_α.

Then, the following proposition claims that the task β exists for any given task α.

Proposition B.2 *If EDF is used at run-time to schedule Γ and $\mathcal{S}$, for any given task α ($\in \Gamma$ or $\in \mathcal{S}$), the task β($\in \Gamma$ or $\in \mathcal{S}$) exists.*

Proof: It is clear that at the end of the last idle period before f_α, the conditions (1), (2), (3), and (4), hold for some task β_0 whose release time is equal to the end of that idle period. If there is no idle period before α, then let β_0 denote the first task which starts its execution at time 0. Let β_1 denote the *last* task which starts its execution between the end of the idle period and f_α, and which satisfies all of the conditions from (1) to (4). In this case, β_1 is the last task in $[r_{\beta_0}, f_\alpha]$ which starts its execution at its release time. In other words, every task executed between r_{β_1} and $f_\alpha]$ has started its execution some time after its release time except β_1.

Suppose that the condition (5) is not satisfied in $[r_{\beta_1}, f_\alpha]$ and let γ denote a task whose start time is between r_{β_1} and f_α, and which has a deadline D_γ greater than D_α. But, because D_α is less than D_γ and EDF is used to schedule the tasks, a contradiction has occurred. γ should never have been executed between r_γ and f_α since the task α has a higher priority than γ. Therefore, task instance β_1 satisfies the condition from (1) to (5).

Then, the start time a of the busy period for α is defined to be r_{β_1} which is found in the above proof procedure. Example busy period is depicted in Fig B.1.

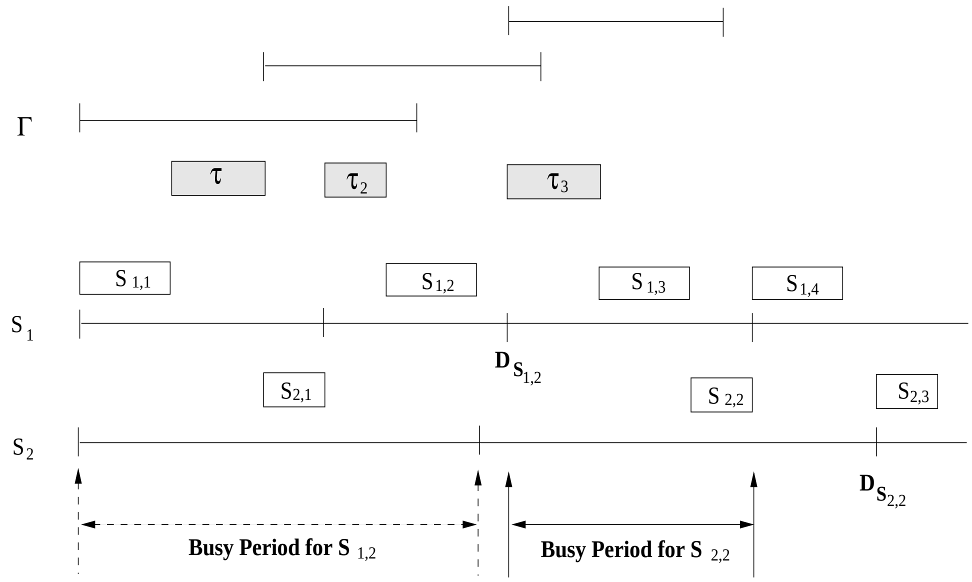

Figure B.1: Busy period

Here, the *earliest finish time* of τ_i is defined as $est(i) + C_i$.

Proposition B.3 *The following is satisfied for every* $i \in [2, v+1]$*:*

$$\forall t_1 \in [eft(\Upsilon_{i-1}), est(\Upsilon_i)] \; :: \; \forall l > 0 \; :: \; \Omega'(t_1, t_1 + l) \geq \Omega'(est(\Upsilon_i), est(\Upsilon_i) + l) \qquad \text{(B.1)}$$

Proof: If the time interval $[est(\Upsilon_i), est(\Upsilon_i) + l]$ is shifted to the left by the amount of $est(\Upsilon_i) - t_1$ which results in a new time interval $[t_1, t_1 + l]$, the slack time is increased by the amount of $est(\Upsilon_i) - t_1$ and decreased with the amount less than or equal to $est(\Upsilon_i) - t_1$. This is depicted in Figure B.2.

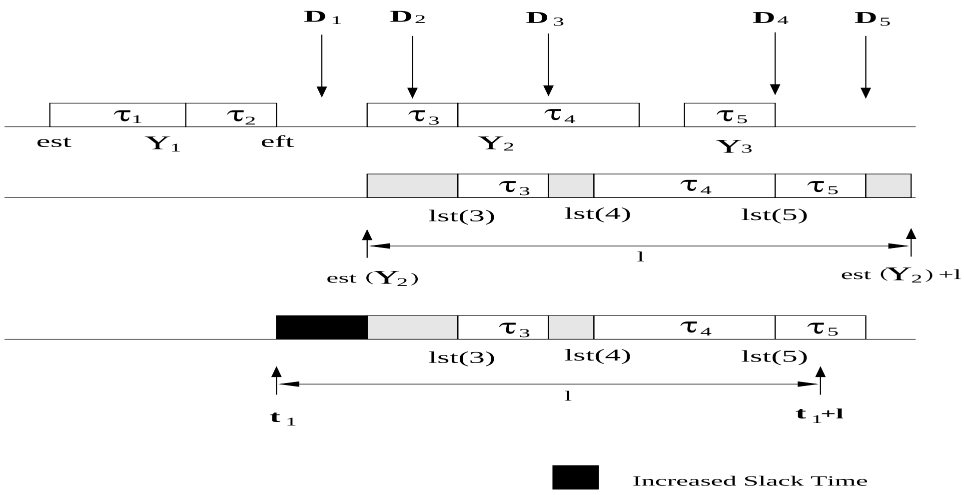

Figure B.2: Ω' is increased or remains the same in the shifted interval

Proposition B.4 *The following is satisfied for every $i \in [1, v]$:*

$$\forall t_1 \in (est(\Upsilon_i), eft(\Upsilon_i)) \ :: \ \forall l > 0 \ :: \ \Omega'(t_1, t_1 + l) \geq \Omega'(est(\Upsilon_i), est(\Upsilon_i) + l) \tag{B.2}$$

Proof: If the time interval $[est(\Upsilon_i), est(\Upsilon_i) + l]$ is shifted to the right by the amount of $t_1 - est(\Upsilon_i)$ which results in a new time interval $[t_1, t_1 + l]$, the maximum slack time, Ω' is increased or at least remains the same as can be seen from Figure B.3. This proves the proposition.

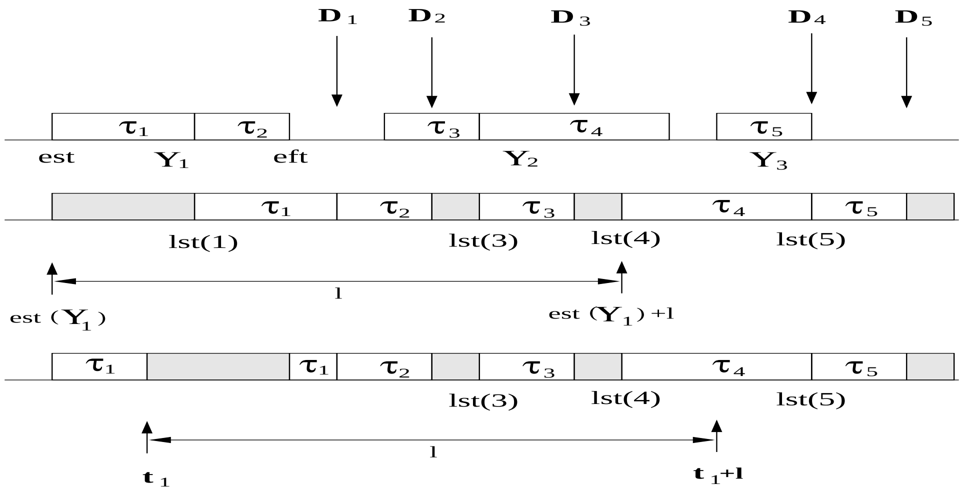

Figure B.3: Ω' is increased or remains the same in the shifted interval

Proposition B.5 *If Γ and $\mathcal{S}$ satisfy the condition of proposition 6.2, then they are schedulable by EDF scheduling algorithm.*

Proof:

Suppose that the condition is satisfied for $\mathcal{S}$ and Γ and some task can't be finished within its deadline. Let's call that task α ($\alpha \in \Gamma$ or $\alpha \in \mathcal{S}$) and the deadline of that task D_α. And, let $BP_\alpha = [t_i, f_\alpha]$ denote a busy period for α. In this case, the actual finish time of α, f_α, is greater than D_α.

Then there are two cases to be considered.

Case 1: $D_\alpha - t_i > LCM$.

Note that the maximum processor demand in $[t_i, t_i + LCM]$ by task instances from $\mathcal{S}$ is less than or equal to $\Omega^{'}(t_i, t_i + LCM)$ from the condition 6.4. In this case, at $t_i + LCM$ a new task instance starts its execution whose release time is equal to $t_i + LCM$. Then, it is obvious that the start time of the busy period, t_i, should be greater than or equal to $t_i + LCM$, which is a contradiction.

Case 2: $D_\alpha - t_i \leq LCM$.

Let τ_ι be the first task in $[t_i, D_\alpha]$ which belongs to Γ. First, suppose that this exists. Then, let Υ_j denote the task group containing τ_{iota}. From the definition of a busy period we know that the release time of τ_ι, r_ι, is greater than or equal to t_i. Then from proposition B.3 and B.4,

$$\forall l > 0 \ :: \ \Omega^{'}(t_i, t_i + l) \geq \Omega^{'}(est(\Upsilon_j), est(\Upsilon_j) + l)$$

This means that if the tasks in $\mathcal{S}$ starts to invoke their task instances from t_i with their minimum inter-arrival times, then they are schedulable with Γ. This implies that the task instances invoked at or after t_i are schedulable since the worst case scenario is that every $S_i \in \mathcal{S}$ starts to be invoked at t_i with δ_i inter-arrival time, which is proven to be schedulable. This contradicts to the assumption that α misses its deadline at D_α.

Second, suppose that τ_ι doesn't exist. In this case all the task instances executed in the interval $[t_i, D_\alpha] \subset [t_i, f_\alpha]$ are from $\mathcal{S}$. It is clear in this case from the condition 6.4 that

$$\forall l > 0 \ :: \ \Omega^{'}(t_i, l) \geq \sum_{k=1}^{m_s} c_k^s \cdot \lfloor (l - t_i + \delta_k - d_k^s)/\delta_k \rfloor$$

From this, we can conclude that every task instance in $[t_i, D_\alpha]$ is schedulable, which contradicts to the assumption that α misses its deadline at D_α.

Bibliography

[1] Ashok K. Agrawala, Seonho Choi, and Leyuan Shi. Designing temporal controls. Technical Report CS-TR-3504, UMIACS-TR-95-81, Department of Computer Science, University of Maryland, July 1995.

[2] N. C. Audlsey, A. Burns, R. I. Davis, and A. J. Wellings. Integrating best effort and fixed priority scheduling. In *Proceedings of the 1994 Workshop on Real-Time Programming*, Lake Constance, Germany, June 1994.

[3] N. C. Audsley. Deadline monotonic scheduling. YCS 146, University of York, Department of Computer Science, October 1990.

[4] T. Baker and A. Shaw. The Cyclic Executive Model and Ada. *Real-Time Systems*, 1(1):7–25, September 1989.

[5] T. P. Baker. A Stack-Based Resource Allocation Policy for RealTime Processes. In *Proceedings, IEEE Real-Time Systems Symposium*, 1990.

[6] A. Belleisle. "Stability of systems with nonlinear feedback through randomly time-varying delays". *IEEE Transactions on Automatic Control*, AC-20:67–75, February 1975.

[7] R. Bellman. *Adaptive Control Process: A Guided Tour.* Princeton,NJ: Princeton University Press, 1961.

[8] R. Bellman. Bellman special issue. *IEEE Transactions on Automatic Control*, AC-26, October 1981.

[9] T. Carpenter, K. Driscoll, K. Hoyme, and J. Carciofini. Arinc 659 scheduling: Problem definition. In *Proceedings, IEEE Real-time Systems Symposium*, San Juan, PR, December 1994.

[10] M. Chen and K. Lin. Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems. *Real-Time Systems*, 2(4):325–346, 1990.

[11] S. Cheng and Ashok K. Agrawala. Scheduling of periodic tasks with relative timing constraints. Technical Report CS-TR-3392, UMIACS-TR-94-135, Department of Computer Science, University of Maryland, December 1994.

[12] S. T. Cheng and Ashok K. Agrawala. Allocation and scheduling of real-time periodic tasks with relative timing constraints. Technical Report CS-TR-3402, UMIACS-TR-95-6, Department of Computer Science, University of Maryland, January 1995.

[13] H. Chetto and M. Chetto. Scheduling Periodic and Sporadic Task in a Real-Time System. *Information Processing Letters*, 30(4):177–184, 1989.

[14] H. Chetto and M. Chetto. Some Results of the Earliest Deadline First Algorithm. *IEEE Transactions on Software Engineering*, SE-15(10):1261–1269, October 1989.

[15] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-Time Tasks under Precedence Constraints. *Real-Time Systems*, 2:181–194, 1990.

[16] G. Dantzig and B. Eaves. Fourier-Motzkin Elimination and its Dual. *Journal of Combinatorial Theory(A)*, 14:288–297, 1973.

[17] R. I. Davis. Approximate slack stealing algorithms for fixed priority pre-emptive systems. Technical Report YCS 217 (1993), Department of Computer Science, University of York, England, November 1993.

[18] R. I. Davis, K. W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Proceedings, IEEE Real-Time Systems Symposium*, pages 222–231. IEEE Computer Society Press, December 1993.

[19] M. Dertouzos. Control Robotics: the Procedural Control of Physical Processes. *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[20] P. Dorato and A. Levis. "Optimal linear regulators: The discrete time case". *IEEE Transactions on Automatic Control*, AC-16:613–620, December 1971.

[21] G. Fohler and C. Koza. Heuristic Scheduling for Distributed Real-Time Systems. MARS 6/89, Technische Universitat Wien, Vienna, Austria, April 1989.

[22] Gerhard Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proceedings, IEEE Real-Time Systems Symposium*. IEEE Computer Society Press, December 1995.

[23] R. Gerber, W. Pugh, and M. Saksena. Parametric Dispatching of Hard Real-Time Tasks. *IEEE Transactions on Computers*, 44(3), Mar. 1995.

[24] T.M. Ghazalie and T.P. Baker. Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9:31–67, 1995.

[25] A. Gosiewski and A. Olbrot. "The effect of feedback delays on the performance of multivariable linear control systems". *IEEE Transactions on Automatic Control*, AC-25(4):729–734, August 1980.

[26] C. Han, C. Hou, and K. Lin. Distance-Constrained Scheduling and Its Applications to Real-Time Systems. *IEEE Transactions on Computers*. To appear.

[27] K. Hirai and Y. Satoh. "Stability of a system with variable time delay". *IEEE Transactions on Automatic Control*, AC-25(3):552–554, June 1980.

[28] X. Homayoun and P. Ramanathan. Dynamic priority scheduling of periodic and aperiodic tasks in hard real-time systems. *Real-Time Systems*, 6(2), March 1994.

[29] Seung H. Hong. Scheduling Algorithm of Data Sampling Times in the Integrated Communication and Control Systems. *IEEE Transactions on Control Systems Technology*, 3(2):225–230, June 1995.

[30] J. P. Lehoczky, L. Sha, and Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proceedings, IEEE Real-Time Systems Symposium*, pages 166–171, Dec. 1989.

[31] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings, IEEE Real-Time Systems Symposium*, pages 261–270, Dec. 1987.

[32] John P. Lehoczky and Sandra Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Proceedings, IEEE Real-Time Systems Symposium*, pages 110–123. IEEE Computer Society Press, December 1992.

[33] J.Y. Leung and J. Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[34] S. T. Levi, Satish K. Tripathi, Scott Carson, and Ashok K. Agrawala. "The MARUTI hard real-time operating system". *ACM Symp. on Op. Syst. Principles, Op. Syst. Review*, 23(3), July 1989.

[35] Shem-Tov Levi and Ashok K. Agrawala. *Real Time System Design*. McGraw Hill, 1990.

[36] C. L. Liu and J. Layland. Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM.*, 20(1):46–61, Jan. 1973.

[37] A. Mok, D. Tsou, and R. Rooij. The msp.rtl real-time scheduler synthesis tool. In *Proceedings, IEEE Real-Time Systems Symposium*, Dec. 1996.

[38] A. K. Mok. *Fundamental Design Problems for the Hard Real-time Environments*. PhD thesis, MIT, May 1983.

[39] C. L. Phillips and H. Troy Nagle. *Digital Control System: Analysis and Design*, chapter 10. Linear Quadratic Optimal Control, pages 356–399. Prentice Hall, 1990.

[40] K. Ramamritham and J. A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67, January 1994.

[41] Z. Rekasius. "Stability of digital control with computer interruptions". *IEEE Transactions on Automatic Control*, AC-31:356–359, April 1986.

[42] M. Saksena. *Parametric Scheduling for Hard Real-Time Systems*. PhD thesis, University of Maryland, College Park, MD 20742, 1994.

[43] Manas Saksena, James da Silva, and Ashok K. Agrawala. *"Design and implementation of Maruti-II"*, chapter 4. Prentice Hall, 1995. In *Advances in Real-Time Systems*, edited by Sang H. Son.

[44] L. Sha, J. P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *Proc. IEEE Real-Time Syst. Symp.*, pages 181–191, Dec. 1986.

[45] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[46] K. G. Shin and H. Kim. "Derivation and application of hard deadlines for real-time control systems". *IEEE Transactions on Systems, Man and Cybernetics*, 22(6):1403–1413, November 1992.

[47] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, June 1989.

[48] Marco Spuri and Giorgio C. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Proceedings, IEEE Real-Time Systems Symposium*, pages 2–11. IEEE Computer Society Press, December 1994.

[49] Sandra R. Thuel and John P. Lehoczky. Algorithm for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Proceedings, IEEE Real-Time Systems Symposium*, pages 22–33. IEEE Computer Society Press, December 1994.

[50] K. Tindell, A. Burns, and A. Wellings. An Extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks. *Real-Time Systems*, 6(2), March 1994.

[51] G.S. Virk. *Digital Computer Control Systems*, chapter 4. McGraw Hill, 1991.

[52] J. Xu and D. L. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, SE-16(3):360–369, March 1990.

[53] J. Xu and D. L. Parnas. On Satisfying Timing Constraints in Hard-Real-Time Systems. In *Proceedings of the ACM SIGSOFT'91 Conference on Software for Critical Systems*, pages 132–146, December 1991.

[54] X. Yuan, M. Saksena, and A. Agrawala. A Decomposition Approach to Real-Time Scheduling. *Real-Time Systems*, 6(1), 1994.

[55] K. Zahr and C. Slivinsky. "Delay in multivariable computer controlled linear systems". *IEEE Transactions on Automatic Control*, pages 442–443, August 1974.

[56] W. Zhao and K. Ramamritham. Simple and Integrated Heuristic Algorithms for Scheduling Tasks with Time and Resource Constraints. *Journal of Systems and Software*, pages 195–205, 1987.