CONTENTION-RESOLVING MODEL PREDICTIVE CONTROL FOR COUPLED CONTROL SYSTEMS WITH SHARED RESOURCES

A Dissertation
Presented to
The Academic Faculty

By

Ningshi Yao

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

August 2020

CONTENTION-RESOLVING MODEL PREDICTIVE CONTROL FOR COUPLED CONTROL SYSTEMS WITH SHARED RESOURCES

I dedicate this dissertation to my parents, Xu Yao and Shenglin Zheng who have been supporting me spiritually throughout my life, and my adviser, Dr. Fumin Zhang who guided me to pursue my dreams and finish my dissertation.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere and greatest appreciation to people who have helped and supported me during my Ph.D study.

Foremost, I want to express my deepest gratitude to my advisor, Dr. Fumin Zhang, for his support, patience and guidance. He provided me with an excellent atmosphere for doing research, patiently corrected my writing, and financially supported my research. I would never have been able to finish my dissertation without his guidance and support.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Coogan, Dr. Zhou, Dr. Wardi, and Dr. Wang for their encouragement, insightful comments, and valuable suggestions.

I would like to thank my fellow lab members and my friends who provided great collaboration and assistance during my study.

Last but not the least, I would like to thank all the funding agencies who supported the research work in this thesis, which are ONR grants N00014-19-1-2556, N00014-19-1-2266 and N00014-16-1-2667; NSF grants OCE-1559475, CNS-1828678, and S&AS-1849228; NRL grants N00173-17-1-G001 and N00173-19-P-1412 ; and NOAA grant NA16NOS0120028 and China Scholarship Council.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Priority-based scheduling strategies are often used to resolve contentions in resource constrained control systems. Such scheduling strategies inevitably introduce time delays into controls, which may degrade the performance or sabotage the stability of control systems. Considering the coupling between priority assignment and control, this thesis presents a novel method to co-design priority assignments and control laws for each control system, which aims to minimize the overall performance degradation caused by contentions. The co-design problem is formulated as a mixed integer optimization problem with a very large search space, rendering difficulty in computing the optimal solution. To solve the problem, we develop a contention-resolving model predictive control method to dynamically assign priorities and compute an optimal control. The priority assignment can be generated using a sample-based approach without excessive demand on computing resources, and all possible priority combinations can be presented by a decision tree. We present sufficient and necessary conditions to test the schedulabilty of the generated priorities assignments when constructing the decision tree, which guarantee that the priority assignments in the decision tree always lead to feasible solutions. The optimal controls can then be computed iteratively following the order of the generated feasible priorities. The optimal priority assignment and control design can be determined by searching the lowest cost path in the decision tree. With the fundamental assumptions required in real-time scheduling, the solution computed by the contention-resolving model predictive control is proved to be globally optimal. The effectiveness of the presented method is verified through simulation in three real-world applications, which are networked control systems, traffic intersection management systems, and human-robot collaboration systems. The performance of our method is compared with the well-known and most commonly used scheduling methods in these applications and demonstrate significant improvements using our method.

CHAPTER 1
INTRODUCTION

In modern industry, shared resources are widely used as the complexity of the systems increases. When multiple systems need access to a shared resource at the same time, a contention occurs. An arbitration mechanism is needed to determine which system can access the resource first. This is a generic problem for the control of complex systems where many control systems are coupled or connected and need to share resources. Examples of such systems include networked control systems (or NCSs), swarming robots, smart grids and traffic intersection management. For NCSs, the communication media (e.g., the network cable or radio frequency) is the shared resource. Control loops that share the same communication media must be scheduled to communicate at proper times to ensure success in transmitting messages to guarantee stability[1, 2]. For the case of load management in a micro power grid, the amount of available electric power generated is a shared resource, and each electric load needs to be scheduled to consume enough power over a time period to accomplish its task [3]. For the case of traffic intersection, the limited space of intersection is a shared resource, and each vehicle needs to be scheduled to avoid collisions [4]. For the case of one human operator supervising a group of robots, limited human cognitive capacity is a shared resource, and each robot should be scheduled to have properly allocated time slots for human intervention to ensure satisfactory performance [5].

A common feature of these applications is that a scheduling policy is needed to resolve contentions. For some applications, many feasible scheduling policies can be used. It is sometimes sufficient to use the one that is easiest to implement or easiest to analyze [6, 7]. However, in many applications, a choice of the scheduling policy may affect performance significantly [8]. For example, well-known scheduling policies, such as rate monotonic scheduling (or RMS) and earliest deadline first (or EDF) algorithms introduced in [9], are widely used in real-time systems. These algorithms are optimal in real-time scheduling in the sense that they can maximize the number of tasks that can be scheduled before deadlines. However, they are not optimized for control purposes. Priority assignments scheduled by EDF and RMS can violate the stability of the whole system [10]. The first-come-first-serve (or FCFS) scheduling mechanism has been used to guarantee fairness; see [11, 12, 13]. However, the FCFS mechanism is conservative, in the sense that it prevents the scheduler from reordering the request of tasks. It may lead to poor scheduling and possible congestion. The drawbacks of these existing scheduling methods motivate the co-design of scheduling and control to improve coordination among control systems and obtain more reliable control performance.

Recent works showed encouraging results by co-designing the scheduling and control in the scenario when multiple control systems need to share a resource, e.g. a shared communication media or limited power resources [14]. One co-design approach is to determine a specific scheduling strategy first and then design the control law to compensate for the time delays or packet dropout induced by the scheduling strategy; see [15, 16, 17, 18, 19, 20]. Another approach is to use optimization-based methods to solve a mixed-integer optimization problem to optimize scheduling decisions along with the control laws. There are relatively fewer studies [21, 22, 23, 24, 25] which take this approach. The co-design problems were formulated as mixed integer quadratic programs (or MIQPs) or mixed integer linear program (or MILP) problems, and were solved by optimization packages such as IBM CPLEX solver. Although these methods can obtain an optimal or a local optimal solution, the major disadvantage is the computation requirement. The optimization problem formulated for co-designing scheduling and control is high dimensional and takes a long time for optimization solvers to find an optimal solution.

Model predictive control (or MPC) offers a natural way to solve the scheduling and control co-design challenge. Instead of considering the whole design time window, MPC performs a prediction-optimization procedure on a finite optimization time horizon [26, 27]. MPCs can incorporate contentions as system constraints and coordinate all the control systems. Many works utilized MPC to design the schedule and control laws for networked control systems [28, 29, 30, 31, 32, 33], energy storage systems [34, 35, 36] and chemical processes [37]. While promising, MPC is largely based on prediction models that are usually nonlinear and non-convex. Therefore, a major challenge in implementing MPC for complex control systems is real-time computational performance.

In this thesis, we propose a contention-resolving model predictive control method to co-design optimal priorities and control in coupled control systems. The contention-resolving MPC can dynamically assign priorities to each control system to minimize the overall performance degradation caused by contentions. Our method differs from existing methods, because we consider priorities as independent decision variables in the objective function of the MPC, not as constraints as was done in previous works [21, 38, 39]. By computing the priorities of each control system, MPCs can achieve better performance. Although the problem can be formulated as a mixed integer optimization problem (or MIP) with a very large search space, doing so would produce difficulty in computing an optimal solution. Therefore, this work proposes a sample-based method to solve this optimization problem without excessive demand on computing resources. The major contributions in this work are as follows:

1. *Sufficient and necessary condition to compute contention time instants.* We utilize the significant moment analysis published in work [19] and establish analytical timing models for preemptive-resume, preemptive-repeat and non-preemptive real-time systems. Based on the timing models, we present sufficient and necessary conditions to determine the time instants when contentions occur and compute the significant moments when a control system actually gains access to the shared resource and when the resource is not occupied. Based on these significant moments, the priority assignment and control law design can be decoupled and we can construct a decision tree to efficiently search all of the possible priority assignments.

2. *Co-design decision tree formulation.* Enabled by the significant moments computed by the timing model, the infinite dimensional priority and control co-design optimization problem can be converted into a path planning problem for a decision tree with only finitely many leaves and branches. Our algorithm assigns priorities only at the significant moments when contentions occur, which are a finite number of time instances on the MPC optimization horizon. The decision tree contains a finite number of branches and each branch corresponds to one possible priority assignment. The optimal control law design is embedded in the computation of branch costs. An optimal solution of the co-design problem must be a path from the root of the decision tree to one of the terminal leaves. There are only finitely many such paths that can be searched. Second, among the finitely many paths, not all need to be searched to find the optimal solution. To the best of our knowledge, the use of a decision tree to decouple the coupled priority assignments and control design had not previously been documented in the literature. In addition, we present a new formula to compute branch costs in the decision tree that is constructed by contention-resolving MPC. The cost function can handle cases where a control system's access to the shared resource is delayed multiple times.

3. *Sufficient and necessary conditions for schedulability (feasibility).* Based on the timing states and significant moments analysis, we developed a finite-time schedulability test to check if there exit some tasks which are not schedulable under a priority assignment along one branch, and an infinite-time schedulability test based on the timing states of the end leaf of a branch to check if this end leaf will definitely lead to unschedulability under all posible priority assignments. We rigorously show that the schedulability consitions are sufficient and necessary.

4. *Co-design algorithm.* We provide a significant modification of the A-star algorithm from [40] to search for the optimal priority assignment. The A-star algorithm is a sampling based algorithm that has been widely used for online path planning in robotics. Different from the works [41, 42], which use a genetic algorithm or an MIP solver to find optimal

schedules, our method searches through a greatly reduced number of possible paths in the decision tree, which can provide scalable methods that eliminate the need for an exhaustive search through the full decision tree.

5. *Practical application case study*. We apply contention-resolving MPC to networked control systems with shared communication media, to scheduling automated vehicles in traffic intersection, and to scheduling human operator to collaborate with multiple robots. We evaluate the performance of the contention-resolving MPC through simulations and compare the results with classical scheduling methods. The optimal priority assignment computed by contention-resolving MPC achieves significant improvement compared to the priority assignment computed by the popular rate monotonic scheduling (or RMS) and earliest deadline first (or EDF), first come first serve (or FCFS), highest speed first (or HSF) and highest trust first (or HTF) scheduling methods.

Compared to the standard MPC framework, contention-resolving MPC produces a computationally tractable approach that lends itself to optimal control and priority assignment co-design. It is a theoretical framework that is general and can be applied to many connected or coupled control systems with shared resources. Our works on contention-resolving MPC have been publish in [10, 43, 44, 45, 46].

The rest of this thesis is organized as follows. Chapter 2 presents the background information and literature review of the existing scheduling and control co-design methods. Chapter 3 introduces the general contention-resolving MPC formulation. Chapter 4 reviews the analytical timing models and presents the sufficient and necessary conditions for infinite-time feasibility test. Chapter 5 presents the path planning problem converted from the priority assignment and MPC design problem. Chapters 6, 7 and 8 present the results of applying contention-resolving MPC to three real-world applications. Chapter 9 summarizes this thesis and discuss the potential future work.

CHAPTER 2
BACKGROUND

This chapter provides the background information of scheduling and control co-design. In particular, the first part gives an overview of real-time control systems and the most famous scheduling methods. The second part discusses the motivation of conducting scheduling and control co-design and the existing co-design methods. The third part presents the background of model predictive control and the limitations of using the traditional model predictive control to solve the scheduling and control co-design problem.

2.1 Scheduling of Real-time Systems

The problems of real-time scheduling arise in many practical situations such as telecommunication and computer systems. The scheduling behavior depends on the types of a real-time system and scheduling algorithms used for the system.

2.1.1 Real-time Systems

Real-time systems are first developed for computing systems. In real-time computing systems, multiple tasks are computed simultaneously on the processor. A successful real-time computing system requires that all tasks can be computed before their respective deadlines. Since the processor can only compute one task at a time, a proper real-time scheduling is required to determine the order of task execution so that each task can meet its deadline. Later, real-time systems have been developed for distributed environments. In distributed environments, each operation is realized by a set of distributed nodes exchanging information over some form of communication networks. To meet deadlines of operations, it is required that the communication among distributed nodes to be real-time.

Historically, the order of task execution in real-time systems were designed by static scheduling. In static scheduling, the schedule of tasks was constructed in an *ad hoc* manner off-line, based on the prior knowledge of task parameters, timing requirements, and scheduling constraints [47]. Each task is assigned with a distinct time interval such that it can access the shared resource without any conflict with tasks from other systems during the designated time intervals. Once a static schedule is made, the tasks are executed according to that schedule. The static scheduling was widely used in the 1960s. However, during the 1970s and 1980s, it was understood that static scheduling can be very inflexible and difficult to maintain because the schedule produced offline cannot be modified online [6]. This understanding leads to an explosion of research and publications on dynamic scheduling by assigning the systems with priorities.

2.1.2 Priority-based Scheduling Methods

In dynamic scheduling, the order of tasks' access to a shared resource is determined by continuously comparing priorities and selecting one task with highest priority from a set of contended tasks. Based on whether preemption is allowed [48], a real-time system can be classified into two categories, preemptive and non-preemptive.

In preemptive system, if a task with higher priority requests access to the shared resource, then it interrupts a lower prioritized task that is occupying the resource. The processing of the low prioritized task can be resumed or repeated once the higher prioritized task is completed. Many earlier works study properties of preemptive scheduling, such as reliability, schedulability and time delay [49, 50]. One practical application for preemptive system is a networked control system (or NCS) where feedback control loops are closed through a preemptive real-time network. In non-preemptive system, if a task is occupying the shared resource, no other tasks can interrupt the current task until it completes the usage of the share resource [51]. Traffic intersection scheduling is a typical non-preemptive system, because once a vehicle has entered the intersection, it is unreasonable and unconventional for it to backup and be interrupted by the vehicles arriving later. Existing analysis techniques for preemptive scheduling are either invalid for non-preemptive scheduling or exhibit high computational complexity. Many works contribute towards establishing new results for non-preemptive scheduling [51, 52, 53]. In our work, we will study both preemptive and non-preemptive systems.

Despite whether preemption is allowed, different priority assignment methods, or can be called scheduling algorithms, also result in different timing behavior in the real-time systems. The scheduling algorithms can be classified to fixed priority scheduling and dynamic priority scheduling. For fixed priority scheduling, rate monotonic scheduling (or RMS) presented in [9], which assigns higher priorities to tasks with shorter periods, is probably the most famous one. RMS is proved to the optimal fixed priority scheduling algorithm for preemptive system with periodic tasks in the sense of resource utilization. For non-preemptive system, such as controller area network (or CAN) [54], fixed priority scheduling algorithms are also widely utilized due to the easy implementation, where priorities are pre-determined by the importance of each sub-system. For dynamic priority scheduling, earliest deadline first (or EDF) scheduling is the optimal one for preemptive real-time system with periodic tasks. EDF dynamically assign priorities such that a task with earliest deadline has highest priority and it can further improve the resource utilization compared to the system using RMS. Using similar ideas as EDF, researchers developed various dynamic priority scheduling methods. A method, named maximum-error-first with try-once-discard (or MEF-TOD), is designed for networked control systems to assign highest priority to the control loop with largest error [55]. For traffic intersection management, many works utilize a first-come-first-serve (or FCFS) scheduling strategy to assign higher priorities to vehicles which arrive earlier at the intersection and coordinate them to cross the intersection first.

Dynamic scheduling is flexible and adaptive because the schedule is constructed online. However, the implementation of priority-based scheduling requires a schedulability test to determine whether each task in a given set of tasks can be excuted before their deadlines.

2.1.3 Schedulablility Test for Real-time System

When dealing with real-time systems, the first question that needs to be answered is whether a system is schedulable under certain priority assignments. Schedulability means that each task in a real-time system can be executed before its deadline. If there exists no priority assignment that can lead to schedulability, then it means that the scheduling and control co-design problem has no feasible solution.

Research of schedulability test starts from the work of Liu and Layland in [9] that is generally regarded as the foundational work in real-time scheduling. Liu and Layland [9] considered a real-time computing system with the following assumptions: (1) all tasks are periodic, preemptive, and synchronized; (2) all tasks have their relative deadlines equal to their periods. They introduced an idea of *critical time instant analysis* to study the worst case when all the scheduling tasks are requested at the same time. At the critical time instant, a task will endure the longest response time. Liu and Layland introduced timing analysis into the study of real-time scheduling. They proved that a set of tasks on the processor is schedulable under RMS algorithm if their total processor utilization satisfies $\sum_{i=1}^{N} U_i \leq N(2^{1/N} - 1)$, in which U_i represents the processor utilization of an individual task i, and N is the number of total tasks on the processor. Also, they proved that a set of tasks is schedulable under EDF algorithm if their total processor utilization satisfies $\sum_{i=1}^{N} U_i \leq 1$. Based on the similar timing analysis in [9] , extensive research has been conducted to improve the results made in Liu and Layland's work. Bini et al [56] derived a hyperbolic bound for tasks scheduled under RMS. The hyperbolic bound is less pessimistic than Liu and Layland's utilization bound. Lehoczky et al [57] showed that the average processor utilization, for a large set of randomly chosen tasks schedulable under RMS, is approximately 88%. Abdelzaher et al [58] relaxed the periodic restriction on tasks and derived an utilization bound for non-periodic tasks. In work [51, 59], the authors present the a necessary and sufficient schedulabilty condition for non-preemptive periodic system. In [60], the schedulability analysis for a combination of non-preemptive periodic tasks and preemptive sporadic tasks under fixed priority scheduling is presented. The timing analysis in these work focus on the state of scheduled tasks at critical time instant. However, the state of real-time system cannot be fully described at the critical time instant, especially for more complex real-time systems.

Research work in [61] introduces a concept of *significant moment analysis*, which can represent the complete status of scheduled tasks in a real-time system at any time instant and perform timing analysis beyond the critical time instant. Schedulability test for non-preemptive periodic tasks is presented in [19], where fixed priority scheduling is used for the system. In [62], the authors establish a timing model for a discrete real-time system and present schedulability analysis under the discrete time setup. However, these work can only perform the schedulability test for a finite time window assuming the priority assignments are known. They cannot guarantee schedulability if priorities are undetermined. In this thesis, we derived the necessary and sufficient conditions of schedulabilty using the *significant moment analysis*.

2.2 Scheduling and Control Co-design

If more than one feasible priority assignments exist to ensure schedulability of coupled control systems, then finding an optimal priority assignment which can minimize the degradation caused by contentions is the goal of the scheduling and control co-design problem. The idea of co-design scheduling and control is first introduced in [63]. It shows that the timings in real-time scheduling will affect the control performance.

2.2.1 Networked Control Systems

Control systems in modern industry are characterized by using shared communication networks to increase modularity and flexibility [64]. Sensors, controllers and actuators connected to the network are regarded as nodes of networked control systems (NCSs). The bandwidth for communication between nodes is mostly limited in NCSs, disallowing sensor messages to transmit immediately after generation, and this causes time delays in the NCSs [6].

Two different types of systems occur in networked control systems namely: time-triggered and event-triggered NCSs [65]. In time-triggered NCSs, an activity in each node is assigned with a distinct time interval such that it can access the communication network without any conflict with other nodes. No contention occurs in the time-triggered control network and the time delays are deterministic and fixed, at least in ideal operating conditions. In event-triggered NCSs, the transmission requests of each node are triggered by its own timer or by certain values of system state variables [66]. Contentions are unavoidable in event-triggered NCSs because of the lack of explicit timing control of events. Usually, priorities are assigned to events to resolve the contentions. This priority-based scheduling introduces time-varying delays for control loops, which may dramatically degrade control performance if not compensated by the controllers. A challenge for controlling event-triggered network systems lies in the integration of control with time delays caused by contentions [67, 68, 18]. In many cases, priorities need to be designed for NCSs because poor priority assignment can violate the stability of the NCSs [69].

Zhang et al further explores the relationship between real-time task periods and the control performance of physical plants [7]. They define an operation point as a collection of periods of all real-time tasks. The goal is to find an optimal operation point that maximizes control performance under the schedulability constraint. The similar idea is extended to control design on automotive ECUs [70]. Since the automotive ECUs only support a finite number of task periods, the paper focuses on finding a possible sequence of task periods such that the resulting average task period is close to optimal ones. In [71], the authors proposed to compensate for the time delays or packet dropout induced by the scheduling strategy. Similar approaches are adopted in [15, 16, 18, 19]. Another approach is to use optimization-based methods to solve a mixed-integer optimization problem to optimize scheduling decisions along with the control laws. There are relatively fewer studies which take this approach. In [22], the co-deign optimization problem of non-preemptive scheduling of control tasks and control law is formulated using $\mathcal{H}_2$ norm. Then the problem is decomposed into two sub-problems. The first sub-problem aims to find the optimal off-line schedule and is solved using the famous branch and bound method [72] for mixed integer optimization. The second sub-problem determines the optimal control gains based on the solution of the first sub-problem. Another widely used method for solving mixed integer optimization is the genetic algorithm [73]. In [24], authors present a modified genetic algorithm to solve the joint optimization of the scheduling and control of electrical loads. Using optimization packages is another approach used in works [25, 74] to solve the co-design problem.

2.2.2 Traffic Intersection

Traffic intersection is another example of resource in our daily life where vehicles need to share the physical space in the intersection. For traffic intersection, the traffic light has been the most commonly used device for intersection scheduling since 1868. However, while traffic lights ensure the safety of conflicting movements at intersections, they also cause increased delay, fuel use, and tailpipe emissions. Frequent stops and starts caused by traffic lights also frustrates drivers and passengers. Smarter intersection management and scheduling are needed to better control vehicles at intersections [4].

Connected and automated vehicles provide significant new opportunities for improving intersection efficiency. A recent study showed that changing the intersection scheduling from traffic lights to coordinating automated vehicles has the potential of doubling the intersection capacity and reducing traffic delays [75]. With the introduction of vehicle-to-vehicle as well as vehicle-to-infrastructure communication, automated vehicles can assist drivers with better decision making and reduce fuel consumption, emissions, and traffic congestion. Numerous research efforts have explored the scheduling and control of automated vehicles [76, 77, 78]. Many works assume that the arrival times of automated vehicles at an intersection satisfy a certain random process. Then based on the arrival time, they utilize a first-come-first-serve (or FCFS) scheduling mechanism, so that controllers can be designed to coordinate the crossing speed of vehicles [11, 13, 12]. However, the FCFS mechanism may lead to poor scheduling and possible congestion. For instance, FCFS schemes can give crossing orders in which several faster vehicles must slow down to favor a slow vehicle, which may not be optimal in the context of total traveling time or energy consumption. FCFS is conservative in the sense that it prevents the intersection scheduler from reordering the entrance of automated vehicles to the intersection. In those cases, the highest-speed-first (or HSF) scheduling, which schedules the vehicles with higher speed to pass the intersection first, is another strategy for intersection scheduling.

Optimization-based approaches have been proposed to compute the optimal schedule for coordinating automated vehicles. The works in [79, 80] utilized a genetic algorithm to dynamically coordinate traffic at intersections and their results were verified through simulation using real traffic data. In [74], the authors used mixed integer quadratic program (or MIQP) to compute an approximate solution to schedule the order of vehicles crossing the intersection. In [81], the intersection scheduling problem was formulated as a mixed integer linear program (or MILP) problem, and was solved by the IBM CPLEX optimization package. Although these methods can obtain an optimal or a local optimal solution for intersection scheduling, the major disadvantage is the computation requirement.

2.2.3 Human and Multi-robot Collaboration System

Recent advances in robotics have enabled the reduction in price, size, and operational complexity of robots. A natural outgrowth of these advances are systems comprised of large numbers of robots that collaborate autonomously in diverse applications. However, even though the autonomous task execution capabilities of robots have progressed rapidly, the human's advantage in high-level reasoning and planning is still needed. As a consequence, the form of human and multi-robot collaboration systems has become a popular and important topic of research [82, 83]. For human and robots collaboration systems, as the human labor cost increases, it can be envisioned that the number of robots that one human operator needs to work with will increase to a large extent. However, a human has limitation on attention capacity. In a famous psychology paper [84], it was revealed that one human cannot efficiently pay attention to more than about seven meaningful items. In more recent studies, researchers discovered that a human can pay attention to only two to four items at the same time [85, 86]. Therefore, when a human is collaborating with multiple robots, the human operator cannot effectively serve or collaborate with all robots at the same time. Which robot the human operator should collaborate with first is a general question for human and multi-robot collaboration systems.

How to allocate or schedule a human's attention to each robot is a research topic studied in real-time scheduling. Inappropriately scheduling a human operator to collaborate with robots has been found to have a negative effect on overall performance in human-robot systems [87]. Numerous research efforts have explored how to better schedule a human's attention to robots. In [88], the authors compared two types of scheduling methods, open-queue (or OP) and shortest job first (or SJF) scheduling, and showed that SJF scheduling can provide more stable robot performance. In [62], the authors proposed a highest trust first (or HTS) based on a robot performance model from [89] and a human–robot mutual trust model, to determine the human operator's schedule to interact with one robot at each time such that the human–robot trust level can always be maintained within a proper range. However, both the SJF and HTS cannot guarantee that the overall performance of robots can be optimized. Murray et al. formulated an integer programming problem to effectively schedule multiple unmanned aerial vehicles and humans to time-sensitive geographically-dispersed tasks and optimize the overall system performance [90].

2.3 Model Predictive Control

Model predictive controllers (or MPCs) is originally developed for industrial process control [91, 92]. Instead of considering the whole design time window, MPC performs a prediction-optimization procedure iteratively, using a predefined cost function (which usually considers the overall performance and efficiency) while receding a finite optimization time horizon [27]. Specially designed MPC can tolerate uncertainty, disturbance and can deal with complicated system constraints. These advantages make MPC attractive to solve the scheduling and control co-design challenge.

Many works utilize MPC to design the schedule and control laws. For networked control system, works (including [21, 38, 39]) have shown the effectiveness of using MPC to compensate for time delays in event-triggered NCSs. However, these methods assume that a pre-defined priority assignment is chosen and do not consider time delay induced by contentions. For traffic system, MPC are used to schedule vehicles and design the vehicle speed for ramp metering [29, 30]. In [33, 30, 93, 94], model predictive control is applied to control and coordinate urban traffic networks. However, due to the nonlinearity of the prediction model, the optimization formulated in MPC is a nonlinear and non-convex optimization problem. The authors need to reformulate the problem into a mixed-integer linear optimization problem to increase the real-time feasibility of solving MPC. In energy storage systems, a special type of MPC, called Economic MPC, is applied to scheduling the operations of operations of lighting and powering heating, ventilation, and air conditioning systems while aiming to minimize the total cost of energy consumption [36, 35, 34]. MPC is also used to solve the integrated scheduling and control problem for chemical processes [95, 37]. The integrated approach can improve the overall process performance by incorporating process dynamics into scheduling considerations, but the computational time is generally too long for online implementation. In summary, while very promising, MPC is largely based on prediction models that are usually nonlinear and non-convex. Therefore, a major challenge in implementing MPC for complex control systems is real-time computational performance. This motivates us to develop a new MPC design which can produce a computationally tractable for the optimal control and priority assignment of the co-design problem for real-time systems.

CHAPTER 3
PROBLEM FORMULATION

Multiple control systems share the same resource, such as computation resource (e.g. processor), communication resource (e.g. communication bus or limited bandwidth), physical spaces (e.g. traffic intersection) or collaborators for their operations. Each control system consists of a sequence of tasks that are repeatedly executed. Each activation of a task may have its own release time, execution time and deadline. The proposed contention-resolving MPC is a general theoretical framework to address resource allocation for coupled control systems with asynchronous task release time, execution time and deadline. In this chapter, we present the setup and formulation of co-design problems.

3.1 Problem Setup

Consider N control systems that must share a limited resource.

3.1.1 System Dynamics

Assume that the i-th control system for $i = 1, 2, ..., N$ is modeled in the form

$$\dot{x}_i(t) = f_i(x_i(t), u_i(t)), \quad y_i(t) = g_i(x_i(t)) \tag{3.1}$$

where x_i, u_i, and y_i represent the state vector, control, and output, respectively. Here and in the sequel, we make the following assumption about the system dynamics, where we use measurability and essential boundedness in the Lebesgue measure sense of [96]:

Assumption 3.1.1 *The function f_i for $i = 1, ..., N$ in (3.1) is such that this holds for each i: For each measurable essentially bounded function u_i and each initial state x_i^0 and each*

$T > 0$, the initial value problem for the dynamics f_i and the initial condition $x_i(0) = x_i^0$ has a unique solution on $[0, T]$.

The preceding assumption is satisfied under standard Lipschitzness conditions, e.g., from [97, Chapter 7].

3.1.2 Task Characteristics

In real-time scheduling, each control system is viewed as a *customer* that must be served to access the shared resource. The i-th customer has a sequence of tasks, denoted by $\{\tau_i[1], \tau_i[2], ..., \tau_i[k], ...\}$, where $k \geq 1$ is the task index of customer i. The completion of each task requires a certain time amount usage of the shared resource.

The timing characteristics of customer i is shown by figure 3.1. The time instant when task $\tau_i[k]$'s request to the shared resource is generated is denoted by $\alpha_i[k]$, which uses the same index k as the task $\tau_i[k]$. The task generation time $\alpha_i[k]$ are usually determined by an event generator of the form $h_i(x_i(\alpha_i[k]), y_i(\alpha_i[k])) = 0$ [98, 99, 100]. The function $h_i(\cdot)$ is defined so that when the state x_i and the system output y_i enter a certain set at time $\alpha_i[k]$, an event will be generated. Event based control is an effective way to reduce the use of shared resources, and is gaining popularity in the control of complex systems [100, 101, 102]. The generation time instants do not need to be determined by event-triggered control, since we also allow the task generation time $\alpha_i[k]$ to be pre-determined, such as the periodic sampling time used by digital control systems. The amount of time for which the task $\tau_i[k]$ needs to occupy the resource is denoted by $C_i[k]$. The task occupation time can be pre-determined or can be a function of control command of system i. The completion time instant when task $\tau_i[k]$ finishes the occupation of the shared resource is denoted as $\gamma_i[k]$.

Assumption 3.1.2 *At any given time, only one customer can occupy the shared resource.*

Assumption 3.1.2 is valid in many real world applications. In the automotive industry, the vehicle communication buses such as the control area network (or CAN) [54] and FlexRay

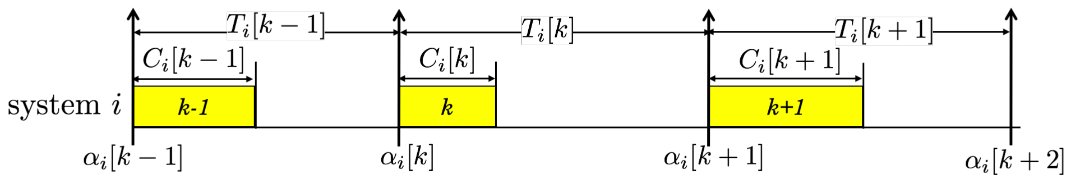


Figure 3.1: Three generated tasks of system i indexed by $k-1$, k, and $k+1$.

[103] only allow one device to transmit messages at any time. Also, in a warehouse, a passageway (e.g., a narrow space between two aisles) may only allow one forklift to enter and transport packages. For traffic intersections, only one vehicle can occupy the intersection to guarantee that no collisions occur among vehicles when they are passing the intersection.

In a real-time system, it is required that each task must be completed before its deadline, in order for the system to be schedulable. In our setup, we define the deadline for a task $\tau_i[k]$ to be the time instant when the next task of customer i is generated, i.e., $\alpha_i[k+1]$. Therefore, for a task to be schedulable, the inequality $\gamma_i[k] \leq \alpha_i[k+1]$ must be satisfied. We also use $T_i[k]$ to denote the amount of time between two successive resource occupation requests from customer i, i.e., $T_i[k] = \alpha_i[k+1] - \alpha_i[k]$, which we assume satisfy:

Assumption 3.1.3 *For each $i \in \{1, \ldots, N\}$, there is a constant $T_i^{\min} > 0$ such that $T_i[k] \geq T_i^{\min}$ for all k.*

Since the time interval between two successive requests is bounded below by $T_i^{\min}$ for each customer i, and since there are only a finitely number N of customers, the total number of requests should be less or equal to $\sum_{i=1}^{N}(t_f - t_0)/T_i^{\min}$, which is a finite number. Therefore, it follows that there are only finitely many requests for access on the time interval $[t_0, t_f]$. Also, the request for the resource will be modeled by a tuple $(\alpha_i, C_i, T_i)[k]$.

3.2 Priority-based Scheduling

When there are no contentions among customers, the following equation is always satisfied:

$$\gamma_i[k] = \alpha_i[k] + C_i[k]. \tag{3.2}$$

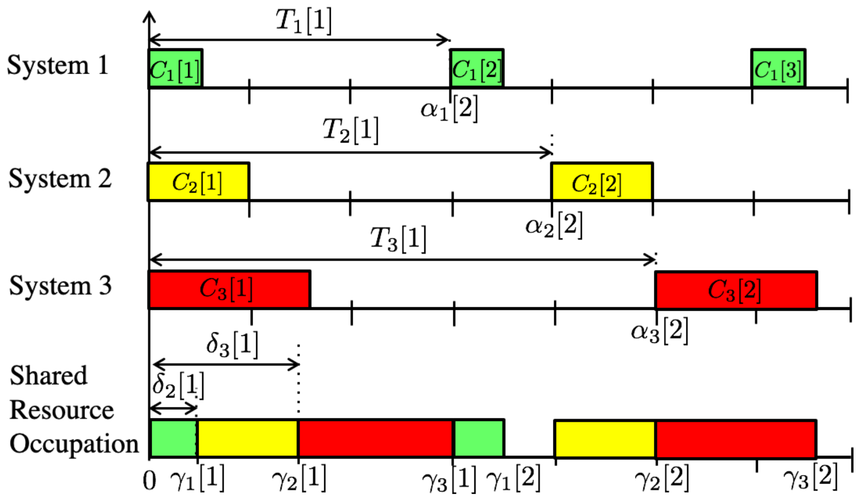


Figure 3.2: Illustration of scheduling three systems. The upper three sub-figures show the task request times when contentions are not considered. The bottom sub-figure shows the resource occupation time after priorities are assigned to resolve the contention that occurs at time 0.

When multiple customers request the shared resource at the same time, a contention occurs and equation (3.2) will not hold. An example of three systems sharing one resource is shown in Figure 3.2. A contention occurs at time 0. The scheduling algorithm determines the order of customers' access to the resource by assigning them priorities. Each customer i is assigned a unique priority number $p_i(t)$, in which case contentions can be resolved by comparing the priorities p_i among all customers who are competing for the resource. In what follows, $\mathcal{P}(\{1, ..., N\})$ denotes the set of all permutations of $\{1, ..., N\}$.

Definition 3.2.1 *A priority assignment is a tuple* $\mathbf{P}(t) = (p_1(t), ..., p_i(t), ..., p_N(t)) \in \mathcal{P}(\{1, ..., N\})$*, where* $p_i(t)$ *is the priority assigned to customer* i *at time* t *and such that for each* i *and* j *in* $\{1, ..., N\}$*, we have* $p_i(t) < p_j(t)$ *if and only if customer* i *is assigned higher priority than customer* j *at time* t*. For each* $t \in [t_0, t_f]$*, the value of* $p_i(t)$ *is a positive integer in* $\{1, \ldots, N\}$*, such that* $p_i(t) \neq p_j(t)$ *if* $i \neq j$*.*

Assumption 3.2.1 *When a contention occurs, only the control system with the smallest* p_i *will be granted access.*

This assumption follows the convention in the scheduling literature of giving smaller numbers to the higher prioritized tasks [48]. Based on the Definition 3.2.1, each task has a unique priority number. Therefore, there exist no ties among the priority assignments when a contention occurs.

The resource access times of lower prioritized systems are delayed by higher prioritized vehicles. A lower prioritized system can gain the access to the shared resource right at or after the time instant when all the higher prioritized contended systems finish the occupation of the shared resource.

Assumption 3.2.2 *If a contention occurs at time t and $p_i(t) + 1 = p_j(t)$, then system j start to utilize the shared resource at time $\gamma_i[k_i]$, where k_i is the index of task of system i.*

This assumption is also used in the priority-based real-time scheduling mechanism from [48], where no inserted idle time should be allowed if there are one or more tasks waiting to use the shared resource. The other scheduling strategies such as RMS, EDF and FCFS are also based on this assumption. For the convenience of later references, we call this assumption the *condition of immediate access (or CIA)*. We will show that the CIA is a necessary condition for finding a global optimal solution for the co-design problem in Chapter 7.

When a contention occurs, the completion times of the tasks of lower prioritized customers are delayed by the higher prioritized customers. We introduce the delay $\delta_i[k]$ so that $\alpha_i[k] + C_i[k] + \delta_i[k]$ is the task completion time for all i and k, i.e. $\gamma_i[k] = \alpha_i[k] + C_i[k] + \delta_i[k]$.

Definition 3.2.2 *If $\alpha_i[k] + C_i[k] + \delta_i[k] \leq \alpha_i[k+1]$ for all i and k, then we say system i is schedulable or the schedulability of the system i is guaranteed.*

This definition means that all tasks are able to be completed before or at their deadlines.

In order to check whether a system is schedulable or not, it is essential to compute the value of time delays $\delta_i[k]$. However, the computation of $\delta_i[k]$ is not trivial.

Example 1 *Consider tasks τ_1, τ_2 and τ_3 with*

$$(C_1[k], C_2[k], C_3[k]) = (0.5, 1, 1.5) \text{ and } (T_1[k], T_2[k], T_3[k]) = (3, 4, 5) \text{ for all } k \geq 1$$

as illustrated in Figure 3.2. Let the priority assignment be $p_1(t) = 1$, $p_2(t) = 2$, and $p_3(t) = 3$. Due to the occupation times of systems 1 and 2, system 3 has the longest time delay. If we exchange the priority assignments between system 1 and 3, i.e., $p_1(t) = 3$, $p_2(t) = 2$ and $p_3(t) = 1$, then system 1 has the longest time delay.

This simple example shows that time delays depend on priority assignments. In Chapter 4, we will present a timing model which can accurately compute the time delays given a specific priority assignment.

3.3 Formulation of Model Predictive Control

We formulate and solve a contention-resolving model predictive control problem to compute optimal priority assignments $\mathbf{P}^*(t) = (p_1^*(t), ..., p_N^*(t))$ and an optimal control command $\mathbf{u}^*(t) = (u_1^*(t), ...u_N^*(t))$ on a time interval $[t_0, t_f]$. The times t_0 and t_f are the starting and ending points of the MPC time horizon, respectively, and t_0 and t_f will move forward in time when the MPC is initiated. Given initial states $\mathbf{x}(t_0) = (x_1(t_0), ..., x_N(t_0))$, initial controls $\mathbf{u}(t_0) = (u_1(t_0), ..., u_N(t_0))$, starting time t_0 and ending time t_f, the co-design method is to find values for the optimal $\mathbf{P}^*(t)$ and $\mathbf{u}^*(t)$ by solving the optimization problem

$$\min_{\mathbf{P}(t),\mathbf{u}(t)} \sum_{i=1}^{N} V_i\big(x_i(t, \mathbf{P}(t_0 \sim t), u_i(t_0 \sim t)), u_i(t, \mathbf{P}(t_0 \sim t))\big) \tag{3.3}$$

over all $\mathbf{u}$ and $\mathbf{P}$ where the cost functions V_i for $i = 1, 2, ..., N$ incorporate the control effort and tracking error. The notation $\mathbf{P}(t_0 \sim t)$ represents all priority assignments $\mathbf{P}(\ell)$ for all $\ell \in [t_0, t)$. The term $x_i(t, \mathbf{P}(t_0 \sim t), u_i(t_0 \sim t))$ represents that the system state x_i is an implicit function of priority assignment $\mathbf{P}(t)$ and control laws $\mathbf{u}(t)$ from the initial time t_0 to time t. Similarly, $u_i(t, \mathbf{P}(t_0 \sim t))$ represents that the control law u_i is also an implicit function of priority assignment $\mathbf{P}(t)$ from the time t_0 to time t. The specific functions will be introduced in Section 4.4 once we presented the analytical timing model to compute the timing and formulate the contention constraints. For example, if the system i is linear and time-invarying, i.e., $\dot{x}_i(t) = A_i x_i(t) + B_i u_i(t)$, then V_i can take a quadratic form

$$\begin{aligned}
&V_i\big(x_i(t, \mathbf{P}(t_0 \sim t), u_i(t_0 \sim t)), u_i(t, \mathbf{P}(t_0 \sim t))\big) = \\
&\frac{1}{2}\int_{t_0}^{t_f} \big(|x_i(t, \mathbf{P}(t_0 \sim t), u_i(t_0 \sim t)) - \bar{x}_i(t)|_{Q_i}^2 + |u_i(t, \mathbf{P}(t_0 \sim t)) - \bar{u}_i(t)|_{R_i}^2\big)\,\mathrm{d}t \\
&\quad + \rho|x_i(t_f, \mathbf{P}(t_0 \sim t_f), u_i(t_0 \sim t_f)) - \bar{x}_i(t_f)|_{K_i}^2,
\end{aligned} \tag{3.4}$$

where $|v|_M^2 = v^T M v$ for any vector v and matrix M for which the matrix multiplication is defined, and where Q_i, R_i, and K_i are positive definite. The parameter $\rho > 0$ is a constant. The notations $\bar{x}_i$ and $\bar{u}_i$ are fixed choices of the corresponding trajectory and control inputs that tracks a given reference signal $\lambda_i(t)$, and $\bar{x}_i(t_f)$ is the terminal state of the corresponding trajectory $\bar{x}_i(t)$ at time t_f. If contentions occur, then time-varying delays can degrade the control performance and increase the tracking errors [19].

While minimizing the cost function, a set of constraints need to be satisfied for all $t \in [t_0, t_f]$. One constraint is the system dynamics $\dot{x}_i(t) = f_i(x_i(t), u_i(t))$ that must be satisfied for each i. Then the control needs to satisfy $u_i(t) \in \mathbb{U}_i$ for all t, where $\mathbb{U}_i$ is a given constraint set for control commands. These constraints appear in most MPC formulations and we assume these sets are compact. The mathematical formulations of these constraints will be presented in Section 4.4.

Since $\mathbf{u}(t)$ is a vector of real numbers and $\mathbf{P}(t)$ is a vector of integers at each time t, the contention-resolving MPC problem is a mixed integer optimization problem (or MIP). It is a nonlinear and non-convex optimization problem that is difficult to solve [104]. Mixed integer programming problems are usually solved by two categories of optimization methods. The first category is combinatorial optimization [105], such as genetic algorithms. However, since the decision variables $\mathbf{u}$ and $\mathbf{P}$ are functions of time, the search space of the solution is very large and does not lend itself to genetic algorithms in real time. The second category of optimization algorithms comprise the branch-and-bound type of algorithms [72]. In branch-and-bound algorithms, the integers are first relaxed to real numbers so that convex optimization algorithms can apply, and then the real valued solutions are rounded up to the nearest higher integer values. Multiple choices of the integer values lead to different "branches" of sub-problems where convex optimization will be applied again. The branch-and-bound algorithm searches for branches with lower estimated cost first, so that the optimal solution can be found without exhausting all permutations of the integers. The branch-and-bound algorithm is computationally efficient but cannot be used to solve the MIP problem associated with contention-resolving MPC, for two reasons. First, the priority assignments $p_i(t)$ cannot be relaxed to be real numbers. Second, the cost function V_i for each i is not an explicit function of the priority assignment $\mathbf{P}(t)$, therefore convex optimization cannot be applied.

We now describe how to refine this problem for contention-resolving MPC.

Assumption 3.3.1 *A controller is triggered at each time instant when a task is completed.*

Hence, each model predictive controller only generates one control command for each request. The resulting control command is applied to the control system, and remains constant until the control system's next task completion time. Therefore, the control u_i is piece-wise constant. This design follows the idea of zeroth-order-hold (or ZOH) mechanism that is frequently used in sampling based control [106, 107]. At each $\gamma_i[k]$, the control command is updated based on the measurement $x_i(\gamma_i[k])$ of customer i and the control value computed by MPC based on the state value $x_i(\gamma_i[k])$. Then with ZOH, the continuous-time control $u_i(t)$ is a piece-wise constant function of the form

$$u_i(t) = u_i[k] \text{ for all } t \in [\gamma_i[k], \gamma_i[k+1]) \text{ and } k, \tag{3.5}$$

which defines the control u_i at all times when customer i can access the shared resource. As mentioned in Section 3.2, the time delays $\delta_i[k]$ depend on the priority assignment among the customers. The priority assignment and control design are coupled through $\delta_i[k]$. With this problem set up, our goal is to solve the MPC problem formulated in Section 3.3 and compute optimal priorities and optimal controls to compensate for the performance degradation caused by contentions and delays.

CHAPTER 4
SIGNIFICANT MOMENT ANALYSIS AND SCHEDULABILITY TEST

Even though the control systems evolve continuously in time, there are certain moments in time that are more significant than other moments. The moments when a control system requests access and finishes the usage of the shared resource are called *significant moments*. They are significant because the status of the system changes at these moments due to whether access to the shared resource is granted or not. The time instants that systems request access to the shared resource, i.e. $\alpha_i[k]$, are significant because these are the times when contentions may start and new priority vectors $\mathbf{P}(t)$ will be assigned. The time when a control system finishes the usage of the shared resource, i.e., the task completion moments $\gamma_i[k]$, are significant because these are the times when the control law $u_i(t)$ will be updated as shown in (3.5).

In order to obtain the significant moments, it is important to compute the value of the $\delta_i[k]$, which is not easy to compute since we need to consider how many control systems are competing for the shared resource and whether they will be delayed based on different scheduling disciplines. In scheduling theory [108], priority-based scheduling can be classified into two categories, preemptive and non-preemptive scheduling. Therefore, in this thesis, we model the scheduling behavior of both preemptive and non-preemptive real-time systems. In our previous work [19] and [109], we developed a significant moment analysis to show how the priority assignment changes the delays. In this section, we present analytical timing models which can determine all significant moments and compute the delays under preemptive scheduling. In Chapter 6, the timing model of continuous time non-preemptive systems will be presented. In Chapter 7, a continuous-time preemptive-repeat task model will be used to model the scheduling behavior at a traffic intersection and its corresponding timing model will be presented. In Chapter 8, the timing model of a discrete time non-preemptive system will be presented.

4.1 Timing States

At each time $t \in [t_0, t_f]$, we define the timing state variable $Z(t) = (D(t), R(t), O(t))$ using the following variables from [109], where a task is a request for access to the shared resource:

Definition 4.1.1 *The vector $D(t) = (d_1(t), ..., d_i(t), ..., d_N(t))$ is the deadline variable, where $d_i(t)$ denotes how long after time t the next task of customer i will be generated, i.e.,*

$$d_i(t) = \alpha_i[k+1] - t, \textit{if}\, t \in [\alpha_i[k], \alpha_i[k+1]).$$

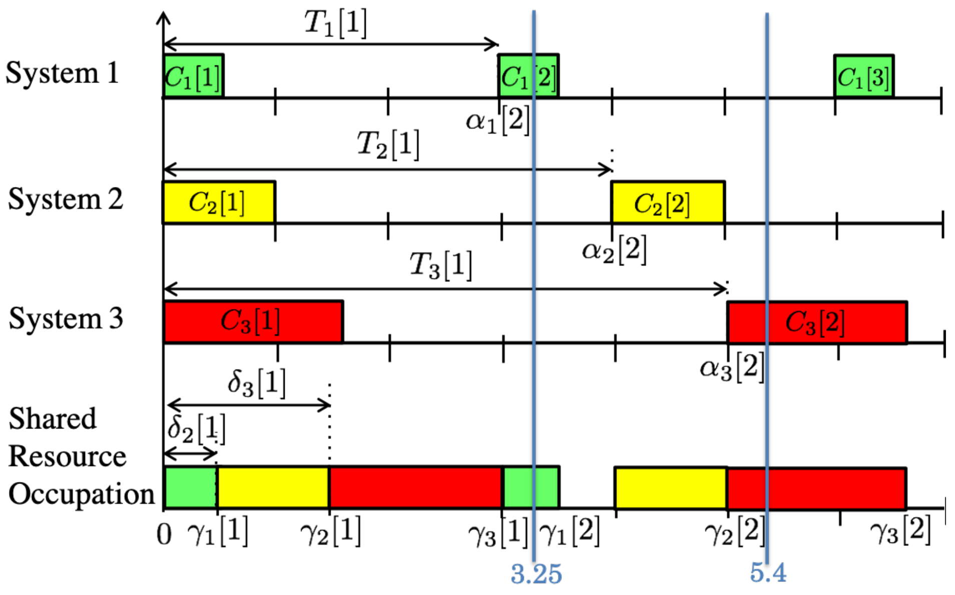
Definition 4.1.2 *The vector $R(t) = (r_1(t), ..., r_i(t), ..., r_N(t))$ is the remaining time variable, where $r_i(t)$ is the remaining time after time t that is required to complete the most recently generated task of customer i, i.e.,*

$$r_i(t) = \begin{cases} \gamma_i[k] - t, & \textit{if}\, t \in [\alpha_i[k], \gamma_i[k]] \\ 0, & \textit{otherwise} \end{cases}.$$

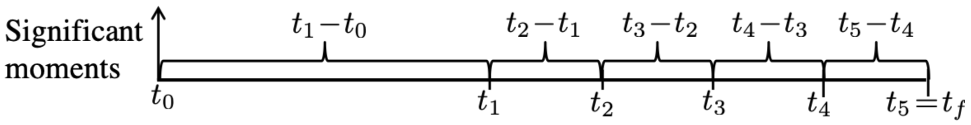
Definition 4.1.3 *The vector $O(t) = (o_1(t), ..., o_i(t), ..., o_N(t))$ is the dynamic response time variable, where $o_i(t)$ denotes the length of time from the most recent request from customer i to the minimum of (a) the time when the most recent request from customer i is completed and (b) the current time t, i.e.,*

$$o_i(t) = \min\{\gamma_i[k], t\} - \alpha_i[k], \textit{if}\, t \in [\alpha_i[k], \alpha_i[k+1]).$$

We use the example in Figure 4.1 to further explain D, R and O.



(a) Scheduled behavior.



(b) Significant moments t_w.

Figure 4.1: Illustration of the timing states.

Example 2 *Again, consider the three periodic tasks are scheduled under a priority assignment* $p_1(t) = 1$*,* $p_2(t) = 2$ *and* $p_3(t) = 3$*. At time* $t = 3.25$*, the next tasks* $\tau_1[3]$*,* $\tau_2[2]$ *and* $\tau_3[2]$ *will be generated at times* 6*,* 4 *and* 5 *respectively. Thus, according to Definition 4.1.1, the deadline are* $\big(d_1(3.25), d_2(3.25), d_3(3.25)\big) = \big(6-3.25, 4-3.25, 5-3.25\big) = \big(2.75, 0.75, 1.75\big)$*. After* $t = 3.25$*, only the request of* $\tau_1[2]$ *has not been finished and will be completed at time* 3.5*. The remaining time for* $\tau_1[2]$ *at time* 3.25 *is* $3.5-3.25=0.25$*, i.e.* $r_1(3.25) = 0.25$*. Therefore, by Definition 4.1.2, we have* $\big(r_1(3.25), r_2(3.25), r_3(3.25)\big) = \big(0.25, 0, 0\big)$*. To compute the dynamic response time, for* $\tau_1[2]$*, its request is generated at* 3 *and will be completed at time* 3.5*, which is greater than the current time* 3.25*. Therefore, the dynamic response time for* $\tau_1[2]$ *at time* 3.25 *is* $3.25-3 = 0.25$*, i.e.* $o_1(3.25) = 0.25$*. For* $\tau_2[1]$*, its request is generated at time* 0 *and is finished at time* 1.5*, which is less than the current time* 3.25*. Therefore, the dynamic response time for* $\tau_2[1]$ *at time* 3.25

is $1.5-0=1.5$. *For* $\tau_3[1]$*, its request is generated at time* 0 *and finishes at time* 3*, which is less than time* 3.25*. Therefore, the dynamic response time for* $\tau_3[1]$ *at time* 3.25 *is* $3-0=3$*, i.e.* $o_3(3.25)=3$*. Thus,* $\big(o_1(3.25), o_2(3.25), o_3(3.25)\big) = \big(0.25, 1.5, 3\big)$*. Similarly, at time* $t=5.4$*, if we assume that* $\alpha_1[3]=\alpha_2[3]=\alpha_3[3]=7$*, then we have the timing state vectors* $D(5.4)=\big(0.6, 2.6, 2.6\big)$*,* $R(5.4)=\big(0, 0, 1.1\big)$ *and* $O(5.4)=\big(0.5, 1.0, 0.4\big)$*.*

For non-preepmtive scheduling, in addition to the above variables, we need:

Definition 4.1.4 *The index variable is* $\mathrm{ID}(t)$ *denotes the index of the control system which is occupying the shared resource at time* t*. We use the convention that if no control system is occupying the resource at time* t*, then* $\mathrm{ID}(t)=0$ *and* $r_0(t)=0$*.*

Therefore, for non-preepmtive scheduling, the timing state variable is

$$Z(t) = (D(t), R(t), O(t), \mathrm{ID}(t)).$$

To support the continuous timing model, we redefine the characteristics tuple of a task in the continuous time domain as follows:

Definition 4.1.5 *At any time* t *within* $[t_0, t_f]$*, we define* $C_i(t)$*,* $T_i(t)$ *and* $P_i(t)$ *to be the execution time, the period, and the priority of task* i *in continuous time domain, respectively. The values of these functions are*

$$C_i(t) = C_i[k],\ T_i(t) = T_i[k] \text{ and } P_i(t) = P_i[k] \tag{4.1}$$

where k *is the largest integer satisfying* $\alpha_i[k] \le t$ *and* $\alpha_i[1] = t_0$*.*

By this definition, we can convert the discrete-time timing characteristics into piece-wise constant functions in continuous time, which will be used in the formulas for the analytical timing model.

The evolution rules for $Z(t)$ within a time interval $[t_0, t_f]$ can be expressed by mathematical equations. These equations lead to a timing model. It is an analytical model that is efficient to compute, and it supports the implementation of real-time model predictive control.

4.2 Delay Prediction Using Timing Model

We use this notation to represent the timing model:

$$Z(t) = \mathbb{H}\Big(t; Z(t_0), \mathbf{S}, \mathbf{P}(t_0 \sim t)\Big), \tag{4.2}$$

where t_0 is a starting time, $\mathbf{S}$ is the set of all triples of the form (α_i, C_i, T_i) for $i = 1, 2, \ldots N$. The timing model consists of a set of analytical algebraic and differential equations that can account for time-varying priorities and interruption of access to the resource by higher priority tasks. By the definition of the state variable $O(t)$, we have

$$\delta_i[k] = o_i(\alpha_i[k+1]^-) - C_i[k]$$

for all k and i, where $\alpha_i[k+1]^-$ denotes the limit from the left.

Different real-time systems may have different timing models, which depends on whether the real-time systems allow pre-emptions or not, and the systems are in continuous time domain or discrete time domain. In this section, we will take the preemptive system as an example to show how the analytical timing model can be derived. In Chapters 6, 7 and 8, the timing models of specific applications will be presented.

4.3 Timing Model for Preemptive Network

The work [61] established a dynamic timing model for the preemptive scheduling discipline. This section consists of a brief review of the timing model from [61]. We divide $[t_0, t_f]$ into disjoint sub-intervals $[t_w, t_{w+1})$ such that tasks are only generated at t_w, but not at any other time point within (t_w, t_{w+1}). The difference between two successive task generating times is defined by

$$t_{w+1} - t_w = \min\{d_1(t_w), ..., d_N(t_w), t_f - t_w\}. \quad (4.3)$$

Example 3 *Consider the example in Figure 4.1. The division of* $[0, 7]$ *into consecutive sub-intervals is carried out using the following procedure. At the beginning of the first sub-interval, let* $t_0 = 0$. *We choose the first window* $t_1 - t_0 = \min\{d_1(0), d_2(0), d_3(0), 7-0\} = \min\{3, 4, 5, 7\} = 3$ *and the end of the sub-interval is* t_1. *Then we choose the window length* $t_2 - t_1$ *and let the end point of this time interval be* t_2. *The process is repeated until one sub-interval reaches the ending time* 7.

After we divide the optimization horizon into sub-intervals. The evolution of $Z(t)$ within any sub-interval $[t_w, t_{w+1})$ can be derived as follows:

At time t_w: We first discuss the value of $[d_i(t), r_i(t), o_i(t)]$ at times t_w. For any task τ_i, the values of the state vector at time t_w, i.e. $[d_i(t_w), r_i(t_w), o_i(t_w)]$, depend on whether an new task of τ_i is released at t_w.

(1) if no task of τ_i is released at t_w, we have that $d_i(t_w^-) > 0$. In this case, the state vector holds its values from t_w^- to t_w where t_w^- denotes the limit from left

$$d_n(t_w) = d_n(t_w^-),\ r_n(t_w) = r_n(t_w^-),\ o_n(t_w) = o_n(t_w^-). \quad (4.4)$$

(2) if a new task of τ_i is released at t_w and the old task of τ_i is completed, then we have that $d_i(t_w^-) = 0$ and $r_i(t_w^-) = 0$. In this case, the state vector $[d_i(t), r_i(t), o_i(t)]$ is updated as

$$d_i(t_w) = T_i(t_w),\ r_i(t_w) = C_i(t_w),\ o_i(t_w) = 0. \quad (4.5)$$

According to equations (4.4) and (4.5), the evolution rules at the times t_w can be summarized as:

$$
\begin{aligned}
&d_i(t_w) = d_i(t_w^-) + \big(1 - \mathsf{sgn}(d_i(t_w^-))\big)\, T_i(t_w),\\
&r_i(t_w) = \mathsf{sgn}(d_i(t_w^-) + r_i(t_w^-))\, r_i(t_w^-) + \big(1 - \mathsf{sgn}(r_i(t_w^-))\big)\, \big(1 - \mathsf{sgn}(d_i(t_w^-))\big)\, C_i(t_w),\\
&o_i(t_w) = o_i(t_w^-)\, \mathsf{sgn}(d_i(t_w^-)) + o_i(t_w^-)\, \mathsf{sgn}(r_i(t_w^-))\big(1 - \mathsf{sgn}(d_i(t_w^-))\big),
\end{aligned}
\tag{4.6}
$$

where sgn is defined by $\mathsf{sgn}(q) = 1$ if $q > 0$ and $\mathsf{sgn}(q) = 0$ if $q = 0$ and the superscripts $-$ indicate a limit from the left.

On the Intervals (t_w, t_{w+1}): For the deadline variable $d_i(t)$, it decreases constantly with rate $\dot{d}_i(t) = -1$ within time interval (t_w, t_{w+1}). Therefore, the equation for $d_i(t_w + \Delta t)$ for values $\Delta \in (0, t_{w+1} - t_w)$ is written as

$$
d_i(t_w + \Delta t) = d_i(t_w) - \Delta t. \tag{4.7}
$$

For the remaining time $r_i(t)$, we know that the resource occupation time of τ_i is preempted until the occupation of all higher priority tasks are completed. Then, the amount of time within $[t_w, t_w + \Delta t]$ that is available to τ_i is

$$
\max\left\{ 0, \Delta t - \sum_{q \in \mathrm{HP}_i(t_w)} r_q(t_w) \right\},
$$

where $\mathrm{HP}_i(t_w) = \{j \in \{1, \ldots, N\} : p_j(t_w) < p_i(t_w)\}$ is the set of all indices of control systems which have higher priorities than control system i at time t_w. The function max guarantees that it will not give a negative result. Therefore, the remaining time of τ_i at time $t_w + \Delta t$ is

$$
r_i(t_w + \Delta t) = \max\left\{ 0, r_i(t_w) - \max\left\{ 0, \Delta t - \sum_{q \in \mathrm{HP}_i(t_w)} r_q(t_w) \right\} \right\}. \tag{4.8}
$$

For the deadline variable $o_i(t)$, we know that $o_i(t)$ will continuously increase before τ_i finishes the occupation of the shared resource. Therefore, if τ_i has finished the occupation before t_w, i.e. $r_i(t_w) = 0$, we have

$$o_i(t_w + \Delta t) = o_i(t_w). \tag{4.9}$$

On the other hand, if τ_i has not finished the occupation before t_w, i.e. $r_i(t_w) > 0$, then we have that

$$o_i(t_w+\Delta t)=o_i(t_w)+\min\left\{\Delta t, r_i(t_w)+\sum_{q\in \mathrm{HP}_i(t_w)} r_q(t_w)\right\}$$

where $r_i(t_w)+\sum_{q\in \mathrm{HP}_i(t_w)} r_q(t_w)$ denotes the time needed for τ_i to complete its most recently generated task. Our use of the min guarantees that the increase of $o_i(t)$ on $[t_w, t_w + \Delta)$ will not exceed Δt. Based on the above analysis, obtain

$$o_i(t_w+\Delta t) = o_i(t_w)+\mathsf{sgn}(r_i(t_w)) \min\left\{\Delta t, r_i(t_w)+\sum_{q\in \mathrm{HP}_i(t_w)} r_i(t_w)\right\} \tag{4.10}$$

Combining all of the evolution rules in (4.6)$-$(4.10) leads to the timing model (4.2) of preemptive scheduling.

4.4 Summary of Constraints

We have refined the contention-resolving MPC design problem by making the constraints related to timing more explicit. In summary, the co-design problem is

$$\min_{\mathbf{P}(t),\mathbf{u}(t)} \sum_{i=1}^{N} V_i\big(x_i(t,\mathbf{P}(t_0\sim t),u_i(t_0\sim t)),u_i(t,\mathbf{P}(t_0\sim t))\big); \tag{4.11a}$$

$$\text{s.t } Z(t)=\mathbb{H}\Big(t;Z(t_0),\mathbf{S},\mathbf{P}(t_0\sim t)\Big),\ \delta_i[k]=o_i(\alpha_i[k+1]^-)-C_i[k],$$
$$\gamma_i[k]=\alpha_i[k]+C_i[k]+\delta_i[k] \text{ for } k=1,...,\overline{K}_i; \tag{4.11b}$$

$$\dot{x}_i(t)=f_i(x_i(t),u_i(t)),\ \ y_i(t)=g_i(x_i(t)),\ \text{with } u_i(t)=u_i(t_0),\ t\in[\gamma_i[0],\gamma_i[1])$$
$$\text{and } u_i(t)=u_i[k] \ \text{ for all } t\in[\gamma_i[k],\gamma_i[k+1]), k=1,...,\overline{K}_i; \tag{4.11c}$$

$$u_i(t)\in\mathbb{U}_i,\ \ \mathbf{P}(t)\in\mathcal{P}(\{1,...,N\}); \tag{4.11d}$$

where $\overline{K}_i$ is the largest index k satisfying $\gamma_i[k+1] < t_f$ and we define $\gamma_i[0]=t_0$ for all i. Equation (4.11b) is the timing model to compute $\delta_i[k]$ which has been introduced in Sections 4.3 and 6.2. Equation (4.11c) represents the system dynamics, which summarizes (3.1) and (3.5). Equation (4.11d) represents the control constraints and the priority assignments are constrained to be in the set $\mathcal{P}(\{1,...,N\})$ of all permutations of $\{1,2,\dots,N\}$ following Definition 3.2.1.

4.5 Schedulability Test Using Worst-case Condition

In this section, we reproduce the same results as in Liu and Layland's work [9] using the timing state and significant moment analysis, which can provide guidance for deriving the schedulability condition for more complicated systems. To achieve this goal, we consider N periodic tasks characterized by (C_i,T_i) where $i=1,...,N$, assuming $T_1<T_2<\cdots<T_N$, which is the same as [9].

4.5.1 Worst-case Scenario Analysis

The infinite-time feasibility condition is derived when considering the worst-case scenario. If at the moment when the worst case occurs, the feasibility is satisfied, then the feasibility is guaranteed at any time. We first study the case where $N = 2$ and later in this section we will extend N to general cases. For systems with two periodic tasks, there are only two possible priority assignments.

1. Task 1 has higher priority

The first priority assignment is that system 1 has higher priority than system 2. It is trivial that if $C_1 \leq T_1$, then system 1 is always schedulable. We define the $\lfloor \cdot \rfloor$ to be the rounding down operator, i.e. $\lfloor x \rfloor$ is the largest integer that is less than or equal to x. And the operator $\{\cdot\}$ takes the fractional part of a real number, i.e., $\{x\} = x - \lfloor x \rfloor$.

Lemma 4.5.1 *Task 2 is schedulable within time $[t_l^c, t_l^c + T_2]$ if $\mathrm{OP}_1(d_1(t_l^c)) + C_2 \leq T_2$ where $\mathrm{OP}_1(d_1(t_l^c))$ is the resource occupation time of task 1 within $[t_l^c, t_l^c + T_2]$ satisfying*

$$\mathrm{OP}_1(d_1(t_l^c)) = \begin{cases} C_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor + \min\left(T_1 \left\{ \frac{T_2 - d_1(t_l^c)}{T_1} \right\}, C_1\right), \textit{ if } 0 \leq d_1(t_l^c) \leq T_1 - C_1 \\ d_1(t_l^c) - (T_1 - C_1) + C_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor + \min\left(T_1 \left\{ \frac{T_2 - d_1(t_l^c)}{T_1} \right\}, C_1\right), \\ \textit{if } T_1 - C_1 < d_1(t_l^c) \leq T_1 \end{cases} \tag{4.12}$$

Proof. Since the time interval we consider is one complete period of system 2, there is only one task from system 2. And because system 2 has lower priority than system 1, not all the time occupied by system 1 can be used by system 2. We first need to compute the time occupied by system 1. The formula to compute the time occupied by system 1 need to consider two cases, which is illustrated by Figure 4.2.

Case 1: if $0 \leq d_1(t_l^c) \leq T_1 - C_1$ as the Case 1 illustrated in Figure 4.2, then within the time $[t_l^c, t_l^c + d_1(t_l^c))$, the task from system 1 has been executed. The number of complete

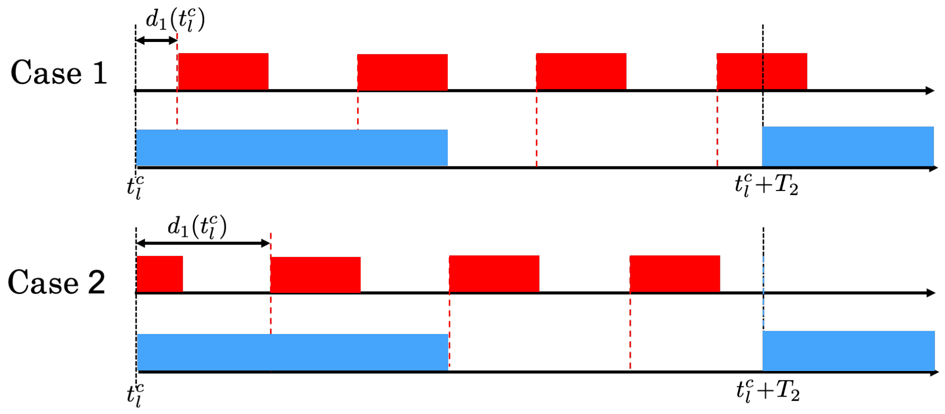


Figure 4.2: Cases of scheduling two systems where system 1 has higher priority. Red rectangles represent the time occupied by system 1 and blue rectangles represent the time occupied by system 2.

periods of system 1 is

$$\left\lfloor \frac{t_l^c + T_2 - (t_l^c + d_1(t_l^c))}{T_1} \right\rfloor = \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor.$$

And the end of the last complete period of system 1 within the time interval $[t_l^c + d_1(t_l^c), t_l^c + T_2]$ is $t_l^c + d_1(t_l^c) + T_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor$.

Within $\left[t_l^c + d_1(t_l^c), t_l^c + d_1(t_l^c) + T_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor\right]$, the time occupied by system 1 is $C_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor$. And within the time interval $\left[t_l^c + d_1(t_l^c) + T_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor, t_l^c + T_2\right]$, the maximal time that will be occupied by system 1 is C_1 since within this time interval, there is no complete period of system 1. But the time that can be occupied by system 1 may be less than C_1 if the total duration of $\left[t_l^c + d_1(t_l^c) + T_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor, t_l^c + T_2\right]$ is less than C_1, which is illustrated by Case 1 in Figure 4.2. Therefore, we need to compare the value of the duration and C_1 and take the smaller value as the time that can be occupied by system 1. The duration of the time interval $\left[t_l^c + d_1(t_l^c) + T_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor, t_l^c + T_2\right]$ is

$$(t_l^c + T_2) - \left(t_l^c + d_1(t_l^c) + T_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor\right) = T_2 - d_1(t_l^c) - T_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor$$
$$= T_1 \left(\frac{T_2 - d_1(t_l^c)}{T_1} - T_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor\right) = T_1 \left\{\frac{T_2 - d_1(t_l^c)}{T_1}\right\}.$$

Therefore, the time occupied by system 1 within the time interval $[t_l^c, t_l^c + T_2]$ can be computed as

$$\mathrm{OP}_1(d_1(t_l^c)) = C_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor + \min\left(T_1 \left\{ \frac{T_2 - d_1(t_l^c)}{T_1} \right\}, C_1\right). \tag{4.13}$$

Case 2: if $T_1 - C_1 < d_1(t_l^c) \leq T_1$ as the Case 2 illustrated in Figure 4.2, then within the time $[t_l^c, t_l^c + d_1(t_l^c))$, the time duration $d_1(t_l^c) - (T_1 - C_1)$ is occupied by system 1. The time occupied by system 1 within time interval $[t_l^c + d_1(t_l^c), t_l^c + T_2]$ has the same formula as Case 1. Therefore, we have

$$\mathrm{OP}_1(d_1(t_l^c)) = d_1(t_l^c) - (T_1 - C_1) + C_1 \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor + \min\left(T_1 \left\{ \frac{T_2 - d_1(t_l^c)}{T_1} \right\}, C_1\right) \tag{4.14}$$

Combining the above two cases, we have the formula (4.12). □

To extend Lemma 4.5.1 to infinite-time window, we need to find the worst case. Since C_2 and T_2 are constant, the worst case occurs when $\mathrm{OP}_1(d_1(t_l^c))$ is maximized.

Theorem 4.5.1 *The maximal value of* $\mathrm{OP}_1(d_1(t_l^c))$*, denoted as* $\mathrm{OP}_{1,\max}$*, is obtained when* $d_1(t_l^c) = 0$ *or* T_1 *and*

$$\mathrm{OP}_{1,\max} = C_1 \left\lfloor \frac{T_2}{T_1} \right\rfloor + \min\left(T_1 \left\{ \frac{T_2}{T_1} \right\}, C_1\right). \tag{4.15}$$

Proof. For the Case 1 where $0 \leq d_1(t_l^c) \leq T_1 - C_1$, if $d_1(t_l^c)$ decreases, the term $\frac{T_2 - d_1(t_l^c)}{T_1}$ increases. Denote a $d_1'(t_l^c)$ which satisfies $0 \leq d_1'(t_l^c) < d_1(t_l^c) \leq T_1 - C_1$. Then we have $\frac{T_2 - d_1'(t_l^c)}{T_1} > \frac{T_2 - d_1(t_l^c)}{T_1}$. Take the difference

$$\begin{aligned}\mathrm{OP}_1(d_1'(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) &= C_1 \left(\left\lfloor \frac{T_2 - d_1'(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor \right) \\ &+ \min\left(T_1 \left\{ \frac{T_2 - d_1'(t_l^c)}{T_1} \right\}, C_1\right) - \min\left(T_1 \left\{ \frac{T_2 - d_1(t_l^c)}{T_1} \right\}, C_1\right).\end{aligned}$$

The term $\left\lfloor \frac{T_2 - d_1'(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor$ can only be either 0 or 1 because of the rounding down operator and $d_1(t_l^c) - d_1'(t_l^c) < T_1$.

If $\left\lfloor \frac{T_2-d_1'(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2-d_1(t_l^c)}{T_1} \right\rfloor = 0$, i.e., the integer part does not change, then the fractional part $\left\{ \frac{T_2-d_1'(t_l^c)}{T_1} \right\}$ must be greater than $\left\{ \frac{T_2-d_1(t_l^c)}{T_1} \right\}$. Therefore, $\mathrm{OP}_1(d_1'(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) \geq 0$. If $\left\lfloor \frac{T_2-d_1'(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2-d_1(t_l^c)}{T_1} \right\rfloor = 1$, then

$$\begin{aligned}\mathrm{OP}_1(d_1'(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) = C_1 + \min\left(T_1 \left\{ \frac{T_2 - d_1'(t_l^c)}{T_1} \right\}, C_1\right) \\ - \min\left(T_1 \left\{ \frac{T_2 - d_1(t_l^c)}{T_1} \right\}, C_1\right).\end{aligned}$$

Since $\min\left(T_1 \left\{ \frac{T_2-d_1'(t_l^c)}{T_1} \right\}, C_1\right) \geq 0$ and $\min\left(T_1 \left\{ \frac{T_2-d_1(t_l^c)}{T_1} \right\}, C_1\right) \leq C_1$, we have

$$\min\left(T_1 \left\{ \frac{T_2 - d_1'(t_l^c)}{T_1} \right\}, C_1\right) - \min\left(T_1 \left\{ \frac{T_2 - d_1(t_l^c)}{T_1} \right\}, C_1\right) \geq -C_1$$

which leads to $\mathrm{OP}_1(d_1'(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) \geq C_1 - C_1 = 0$.

Therefore, for Case 1, we can conclude that if $d_1(t_l^c)$ decreases, then $\mathrm{OP}_1(d_1(t_l^c))$ increases. The maximal value is obtained when $d_1(t_l^c) = 0$ and $\mathrm{OP}_1(0) = C_1 \lfloor T_2/T_1 \rfloor + \min\left(T_1 \{T_2/T_1\}, C_1\right)$.

For the Case 2 where $T_1 - C_1 < d_1(t_l^c) \leq T_1$, denote a $d_1''(t_l^c)$ which satisfies $T_1 - C_1 < d_1(t_l^c) < d_1''(t_l^c) \leq T_1$. Then we have $\frac{T_2-d_1''(t_l^c)}{T_1} < \frac{T_2-d_1(t_l^c)}{T_1}$. Take the difference

$$\begin{aligned}\mathrm{OP}_1(d_1''(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) = d_1''(t_l^c) - d_1(t_l^c) + C_1 \left(\left\lfloor \frac{T_2 - d_1''(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2 - d_1(t_l^c)}{T_1} \right\rfloor \right) \\ + \min\left(T_1 \left\{ \frac{T_2 - d_1''(t_l^c)}{T_1} \right\}, C_1\right) - \min\left(T_1 \left\{ \frac{T_2 - d_1(t_l^c)}{T_1} \right\}, C_1\right).\end{aligned} \tag{4.16}$$

The term $\left\lfloor \frac{T_2-d_1''1(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2-d_1(t_l^c)}{T_1} \right\rfloor$ can only be either 0 or -1 because of the rounding down operator. Similarly as the discussion of Case 1, if $\left\lfloor \frac{T_2-d_1''1(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2-d_1(t_l^c)}{T_1} \right\rfloor = 0$, then

$$\left\{\frac{T_2 - d_1''(t_l^c)}{T_1}\right\} - \left\{\frac{T_2 - d_1(t_l^c)}{T_1}\right\} = \frac{T_2 - d_1''(t_l^c)}{T_1} - \frac{T_2 - d_1(t_l^c)}{T_1} = \frac{d_1(t_l^c) - d_1''(t_l^c)}{T_1} < 0 \text{ and}$$

$$\begin{aligned}\mathrm{OP}_1(d_1''(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) &= d_1''(t_l^c) - d_1(t_l^c) \\ &\quad + \min\left(T_1\left\{\frac{T_2 - d_1''(t_l^c)}{T_1}\right\}, C_1\right) - \min\left(T_1\left\{\frac{T_2 - d_1(t_l^c)}{T_1}\right\}, C_1\right).\end{aligned}$$

Because of the minimal operator, we discuss the following three cases:

1. If $C_1 \leq T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\} < T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}$, then we have $\min\left(T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\}, C_1\right) = C_1$ and $\min\left(T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}, C_1\right) = C_1$. Therefore, $\mathrm{OP}_1(d_1''(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) = d_1''(t_l^c) - d_1(t_l^c) > 0$.

2. If $T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\} < T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} \leq C_1$, then we have $\min\left(T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\}, C_1\right) = T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\}$ and $\min\left(T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}, C_1\right) = T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}$. Therefore, $\mathrm{OP}_1(d_1''(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) = d_1''(t_l^c) - d_1(t_l^c) + T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\} - T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} = d_1''(t_l^c) - d_1(t_l^c) + T_1\frac{d_1(t_l^c)-d_1''(t_l^c)}{T_1} = d_1''(t_l^c) - d_1(t_l^c) + d_1(t_l^c) - d_1''(t_l^c) = 0$.

3. If $T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\} < C_1 \leq T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}$, then we have $\min\left(T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\}, C_1\right) = T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\}$ and $\min\left(T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}, C_1\right) = C_1$. Therefore, $\mathrm{OP}_1(d_1''(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) = d_1''(t_l^c) - d_1(t_l^c) + T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\} - C_1 \geq d_1''(t_l^c) - d_1(t_l^c) + T_1\left\{\frac{T_2-d_1''(t_l^c)}{T_1}\right\} - T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} = 0$.

Summarizing the above three cases, we have $\mathrm{OP}_1(d_1''(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) \geq 0$.

If $\left\lfloor \frac{T_2-d''_1 1(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2-d_1(t_l^c)}{T_1} \right\rfloor = -1$, then

$$\begin{aligned}
&\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\} - \left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} \\
&= \frac{T_2-d''_1(t_l^c)}{T_1} - \left\lfloor \frac{T_2-d''_1 1(t_l^c)}{T_1} \right\rfloor - \left(\frac{T_2-d_1(t_l^c)}{T_1} - \left\lfloor \frac{T_2-d_1(t_l^c)}{T_1} \right\rfloor\right) \\
&= \frac{T_2-d''_1(t_l^c)}{T_1} - \frac{T_2-d_1(t_l^c)}{T_1} - \left(\left\lfloor \frac{T_2-d''_1 1(t_l^c)}{T_1} \right\rfloor - \left\lfloor \frac{T_2-d_1(t_l^c)}{T_1} \right\rfloor\right) \\
&= \frac{d_1(t_l^c)-d''_1(t_l^c)}{T_1} + 1 = \frac{d_1(t_l^c)-d''_1(t_l^c)+T_1}{T_1} \text{ and}
\end{aligned}$$

$$\begin{aligned}
\mathrm{OP}_1(d''_1(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) = {} & d''_1(t_l^c) - d_1(t_l^c) - C_1 \\
& + \min\left(T_1\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\}, C_1\right) - \min\left(T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}, C_1\right).
\end{aligned}$$

Because $T_1 - C_1 < d_1(t_l^c) < d''_1(t_l^c) \le T_1$, we have $d''_1(t_l^c) - d_1(t_l^c) < T_1 - (T_1 - C_1) = C_1$. Therefore, $\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\} - \left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} > \frac{T_1-C_1}{T_1} > 0$. If $T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} \ge C_1$, i.e., $\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} \ge \frac{C_1}{T_1}$, then $\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\} > \frac{C_1}{T_1} + \frac{T_1-C_1}{T_1} = 1$, which is contradict to the fact that $\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\}$ is a fractional part. Therefore, we must have $T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} < C_1$. We will discuss the following two cases:

1. If $T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} < C_1 \le T_1\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\}$, then we have $\min\left(T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}, C_1\right) = T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}$ and $\min\left(T_1\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\}, C_1\right) = C_1$. Therefore, $\mathrm{OP}_1(d''_1(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) = d''_1(t_l^c) - d_1(t_l^c) - C_1 + C_1 - T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} \ge d''_1(t_l^c) - d_1(t_l^c) - T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} \ge 0$.
2. If $T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} < T_1\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\} \le C_1$, then we have $\min\left(T_1\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\}, C_1\right) = T_1\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\}$ and $\min\left(T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}, C_1\right) = T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\}$. Therefore, $\mathrm{OP}_1(d''_1(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) = d''_1(t_l^c) - d_1(t_l^c) - C_1 + T_1\left\{\frac{T_2-d''_1(t_l^c)}{T_1}\right\} - T_1\left\{\frac{T_2-d_1(t_l^c)}{T_1}\right\} = d''_1(t_l^c) - d_1(t_l^c) - C_1 + T_1\frac{d_1(t_l^c)-d''_1(t_l^c)+T_1}{T_1} = T_1 - C_1 > 0$.

Summarizing the above two cases, we have $\mathrm{OP}_1(d''_1(t_l^c)) - \mathrm{OP}_1(d_1(t_l^c)) \ge 0$. Therefore, for Case 2, we can conclude that if $d_1(t_l^c)$ increases, then $\mathrm{OP}_1(d_1(t_l^c))$ increases. The maximal value is obtained when $d_1(t_l^c) = T_1$ and $\mathrm{OP}_1(T_1) = C_1\lfloor T_2/T_1\rfloor + \min(T_1\{T_2/T_1\}, C_1)$.

□

Remark 1 *Here $d_1(t_l^c) = 0$ or T_1 is actually the critical time instant introduced in [9].*

Theorem 4.5.1 shows the great advantage of using the SMA because the worst case can be justified by derivations from mathematical equations.

Then based on the Theorem 4.5.1, we can have

Corollary 4.5.1 *Tasks from system* 2 *are schedulable at any time* t *satisfying* $t \geq t_l^c$ *if* $C_1 \lfloor T_2/T_1 \rfloor + \min\left(T_1 \{T_2/T_1\}, C_1\right) + C_2 \leq T_2$.

Proof. If a task from system 2 is schedulable at the worst-case moments, i.e., the moment shown in Theorem 4.5.1, then it is always schedulable. □

2. Task 2 *has higher priority*

The other priority assignment is that system 2 has higher priority than system 1. It is trivial that if $C_2 \leq T_2$, then system 2 is always schedulable.

Lemma 4.5.2 *Task* 1 *is schedulable within time* $[t_l^c, t_l^c + T_1]$ *if* $\mathrm{OP}_2(d_1(t_l^c)) + C_1 \leq T_1$ *where* $\mathrm{OP}_2(d_1(t_l^c))$ *is the resource occupation time of task* 2 *within* $[t_l^c, t_l^c + T_1]$ *satisfying*

$$\mathrm{OP}_2(d_2(t_l^c)) = \begin{cases} \min\left(T_1 - d_2(t_l^c), C_2\right), \textit{ if } 0 \leq d_2(t_l^c) \leq T_2 - C_2 \\ T_1 - T_2 + C_2, \textit{if } T_2 - C_2 < d_2(t_l^c) \leq T_1 \\ d_2(t_l^c) - (T_2 - C_2), \textit{if } T_1 < d_2(t_l^c) \leq T_2. \end{cases} \tag{4.17}$$

Proof. Since the time interval we consider is one complete period of system 1, there is at most one task from system 2. And because system 1 has lower priority than system 2, not all the time occupied by system 2 can be used by system 1. We first need to compute the time occupied by system 2. The formula to compute the time occupied by system 2 need to consider three cases, which is illustrated by Figure 4.3.

Case 1: if $0 \leq d_2(t_l^c) \leq T_2 - C_2$, then a task from system 2 is generated at time $t_l^c + d_2(t_l^c)$. If the end of the execution of this task may end earlier than or at $t_l^c + T_1$, then $\mathrm{OP}_2(d_2(t_l^c)) = C_2$. If the end of the execution of this task may end later than

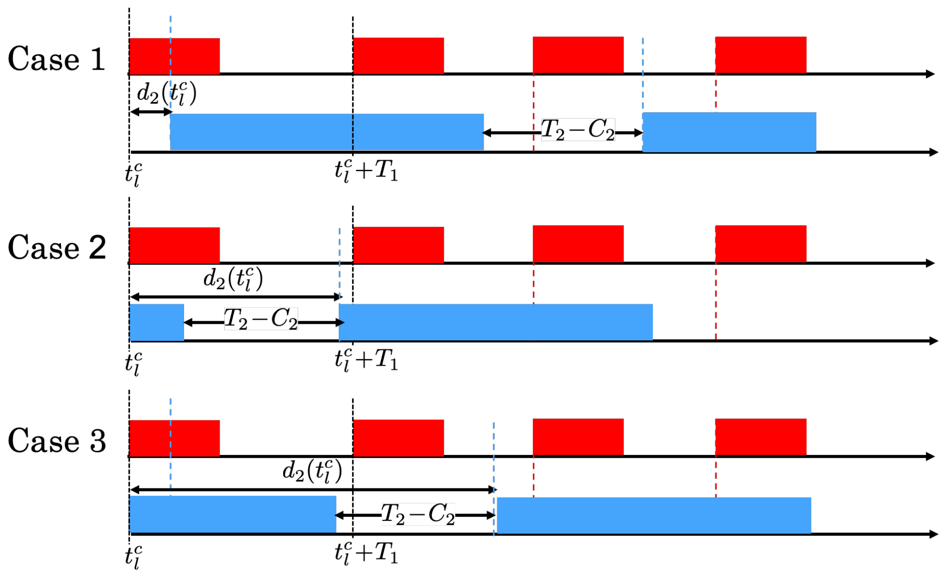


Figure 4.3: Cases of scheduling two systems where system 2 has higher priority. Red rectangles represent the time occupied by system 1 and blue rectangles represent the time occupied by system 2.

$t_l^c + T_1$, illustrated in Figure 4.2, then $\mathrm{OP}_2(d_2(t_l^c)) = T_1 - d_2(t_l^c)$. Therefore, we have $\mathrm{OP}_2(d_2(t_l^c)) = \min\left(T_1 - d_2(t_l^c), C_2\right)$.

Case 2: if $T_2 - C_2 < d_2(t_l^c) \leq T_1$, then part of the task from system 2 is executing at time t_l^c. The occupation time of the current task from system 2 is $d_2(t_l^c) - (T_2 - C_2)$. And since $d_2(t_l^c) \leq T_1$, the generation time of next task from system 2 $t_l^c + d_2(t_l^c)$ is within the interval $[t_l^c, t_l^c + T_1]$. Therefore, the idle time within the interval $[t_l^c, t_l^c + T_1]$ is $T_2 - C_2$ and we have $\mathrm{OP}_2(d_2(t_l^c)) = T_1 - (T_2 - C_2) = T_1 - T_2 + C_2$.

Case 3: if $T_1 < d_2(t_l^c) \leq T_2$, then part of the task from system 2 is executing at time t_l^c and the generation time $t_l^c + d_2(t_l^c)$ of next task from system 2 is greater than $t_l^c + T_1$. Therefore, the occupation time of the task from system 2 is $\mathrm{OP}_2(d_2(t_l^c)) = d_2(t_l^c) - (T_2 - C_2)$. □

Theorem 4.5.2 *The maximal value of* $\mathrm{OP}_2(d_2(t_l^c))$*, denoted as* $\mathrm{OP}_{2,\max}$*, equals to* C_2*, which is obtained when* $d_2(t_l^c) = T_2$ *if* $T_1 < C_2$ *or* $d_2(t_l^c) = T_2$ *and* $0 \leq d_2(t_l^c) \leq T_1 - C_2$.

Proof. From Case 3 in (4.17), $\mathrm{OP}_2(d_2(t_l^c))$ increases if $d_2(t_l^c)$ increases. Therefore, the maximal value of $\mathrm{OP}_2(d_2(t_l^c))$ is obtained when $d_2(t_l^c) = T_2$ and $\mathrm{OP}_{2,\max} = C_2$.

For Case 2 in (4.17), $\mathrm{OP}_2(d_2(t_l^c))$ is a constant $T_1 - T_2 + C_2$. Since $T_1 \le T_2$, we have $T_1 - T_2 + C_2 \le C_2$.

For Case 1 in (4.17), if $T_1 < C_2$, then the maximal value of $T_1 - d_2(t_l^c)$ is T_1 and $T_1 < C_2$. Therefore, $\min(T_1 - d_2(t_l^c), C_2) = T_1 - d_2(t_l^c)$. And since $0 \le d_2(t_l^c) \le T_2 - C_2$, then maximal value of $T_1 - d_2(t_l^c)$ is T_1 which is less than C_2 in Case 3. If $T_1 \ge C_2$, then the maximal value of $T_1 - d_2(t_l^c)$ is T_1 and $T_1 > C_2$. Therefore, $\min(T_1 - d_2(t_l^c), C_2) = C_2$ when $0 \le d_2(t_l^c) \le T_1 - C_2$, which equals to C_2 in Case 3. □

Corollary 4.5.2 *Tasks from system* 1 *are schedulable at any time* t *satisfying* $t \ge t_l^c$ *if* $C_1 + C_2 \le T_1$.

Proof. If a task from system 1 is schedulable at the worst-case moments, i.e., the moment shown in Theorem 4.5.2, then it is always schedulable. □

3. Task 1 should have higher priority

Then we will prove a better priority assignment is that system 1 has higher priority than system 2 in the sense of schedulability, which is the priority assignment under RMS.

Theorem 4.5.3 *If task* 2 *is assigned with higher priority and the system is schedulable at any time, then the system must be schedulable if task* 1 *is assigned with higher priority than task* 2*.*

Proof. We will show that $C_1 + C_2 \le T_1$ implies $C_1 \lfloor T_2/T_1 \rfloor + \min(T_1 \{T_2/T_1\}, C_1) + C_2 \le T_2$. Since $T_2 \ge T_1$, we have $\lfloor T_2/T_1 \rfloor \ge 1$ and

$$\begin{aligned} C_1 \lfloor T_2/T_1 \rfloor + \min(T_1 \{T_2/T_1\}, C_1) &\le C_1 \lfloor T_2/T_1 \rfloor + T_1 \{T_2/T_1\} + C_2 \\ &\le C_1 \lfloor T_2/T_1 \rfloor + T_1 \{T_2/T_1\} + C_2 \lfloor T_2/T_1 \rfloor \\ &= (C_1 + C_2) \lfloor T_2/T_1 \rfloor + T_1 \{T_2/T_1\}. \end{aligned} \tag{4.18}$$

Because $C_1 + C_2 \leq T_1$, we have $(C_1 + C_2)\lfloor T_2/T_1 \rfloor + T_1 \{T_2/T_1\} \leq T_1 \lfloor T_2/T_1 \rfloor + T_1 \{T_2/T_1\} = T_1 \cdot T_2/T_1 = T_2$. □

4. Schedulability for N Tasks

We now extend our results to N tasks using the results of two tasks.

Corollary 4.5.3 *If a feasible fixed priority assignment exists for some task set, the RMS priority assignment is feasible for that task set.*

Proof. Let $\tau_1, \tau_2, ..., \tau_N$ be a set of N tasks with a certain feasible priority assignment $\mathbf{P}$. Let τ_i and τ_j be two tasks of adjacent priorities in such an assignment with τ_i being the higher priority one, i.e., $p_j = p_i + 1$, and satisfying that $T_i > T_j$. If we can not find such a pair. Then the feasible priority assignment $\mathbf{P}$ is assigned by RMS strategy. If we can find such pair, then we will interchange the priorities of τ_i and τ_j which leads to a new priority assginment $\mathbf{P}'$ with $p'_j = p_i$, $p'_i = p_j$ and $p'_n = p_n$ if $n \neq i$ and $n \neq j$. Then we will show that if the tasks $\tau_1, \tau_2, ..., \tau_N$ are schedulable under $\mathbf{P}$, they must also be schedulable under $\mathbf{P}'$. Since tasks i and j have adjacent priorities, interchanging the priorities of τ_i and τ_j does not affect the scheduling behaviors of the other tasks that are neither i nor j. This means that a task n with $n \neq i$ and $n \neq j$ occupies the exactly same time intervals under $\mathbf{P}'$ as under $\mathbf{P}$. Therefore, we only need to consider two tasks case as discussed in Theorem 4.5.3, which has already shown that if task i with larger period is assigned with higher priority and the system is schedulable at any time, then the system must be schedulable if task j with smaller period is assigned with higher priority than task i. And since the rate-monotonic priority assignment can be obtained from any priority ordering by a sequence of pairwise priority re-orderings as the way we interchange the priorities between i and j, we prove this corollary. □

Based on Corollary 4.5.3, RMS is the optimal fixed scheduling methods. In the rest of this subsection, we will directly use the priority assignments under RMS, i.e., if there are systems j and i with indices $j < i$, then $p_j < p_i$ because $T_j < T_i$.

Lemma 4.5.3 *Task i where $i \geq 2$ is schedulable within time $[t_l^c, t_l^c+T_i]$ if $\sum_{j=1}^{i-1} \mathrm{OP}_j(d_j(t_l^c))+ C_i \leq T_i$ where $\mathrm{OP}_j(d_j(t_l^c))$ is the resource occupation time of task j within $[t_l^c, t_l^c+T_i]$ satisfying*

$$\mathrm{OP}_j(d_j(t_l^c))=\begin{cases} C_j \left\lfloor \frac{T_i - d_j(t_l^c)}{T_j} \right\rfloor + \min\left(T_j \left\{\frac{T_i - d_j(t_l^c)}{T_j}\right\}, C_j\right), \textit{ if } 0 \leq d_j(t_l^c) \leq T_j - C_j \\ d_j(t_l^c) - (T_j - C_j) + C_j \left\lfloor \frac{T_i - d_j(t_l^c)}{T_j} \right\rfloor + \min\left(T_j \left\{\frac{T_i - d_j(t_l^c)}{T_j}\right\}, C_j\right), \\ \textit{if } T_j - C_j < d_j(t_l^c) \leq T_j \end{cases} \tag{4.19}$$

Proof. Since the optimal fixed priority assignment is RMS, for any task from system i, the tasks from system with index $j < i$ have higher priority than i. For the task from system i to be schedulable, the total occupation time of all the higher prioritized tasks should be less than or equal to $T_i - C_i$. The occupation time of tasks from system j can be derived similarly as 4.12 in Lemma 4.5.1 with system j replacing system 1 and system i replacing system 2. Then we can obtain (4.19) □

Theorem 4.5.4 *The maximal value of $\mathrm{OP}_j(d_j(t_l^c))$, denoted as $\mathrm{OP}_{j,\max}$, is obtained when $d_j(t_l^c) = 0$ or T_j and $\mathrm{OP}_j(d_j(t_l^c)) = C_j \left\lfloor \frac{T_i}{T_j} \right\rfloor + \min\left(T_j \left\{\frac{T_i}{T_j}\right\}, C_j\right)$.*

Proof. The proof is similar as Theorem 4.5.1 with system j replacing system 1 and system i replacing system 2. □

This theorem shows that the worst-case occurs when tasks from all systems are generate at the same time, which is the critical instant proved in Theorem 1 in [9]. Then we can have the infinite-time schedulability test for tasks from system i as

Corollary 4.5.4 *Task i is schedulable at any time $t \geq t_l^c$ if $\sum_{j=1}^{i-1} \mathrm{OP}_{j,\max} + C_i \leq T_i$.*

Proof. Since the optimal fixed priority assignment is RMS, for any task i, the tasks with index $j < i$ have higher priority than i. Therefore, the worst-case scenario for task i is the occupation time of the higher prioritized systems reach their maximal and $\sum_{j=1}^{i-1} \mathrm{OP}_{j,\max} + C_i$ is less than or equal to one period of task i. □

4.5.2 Least Upper Bound of Utilization

In real-time system, utilization factor $U = \sum_{i=1}^{N}(C_i/T_i)$ is used to test feasibility. Since RMS is optimal fixed priority assignment, the utilization factor achieved by the RMS is greater than or equal to the utilization factor for any other priority assignment for that task set. Therefore, the least upper bound of utilization factor is the infimum of the utilization factors corresponding to the RMS over all possible request periods and run-times for the tasks. The bound is first determined for two tasks, then extended for an arbitrary number of tasks.

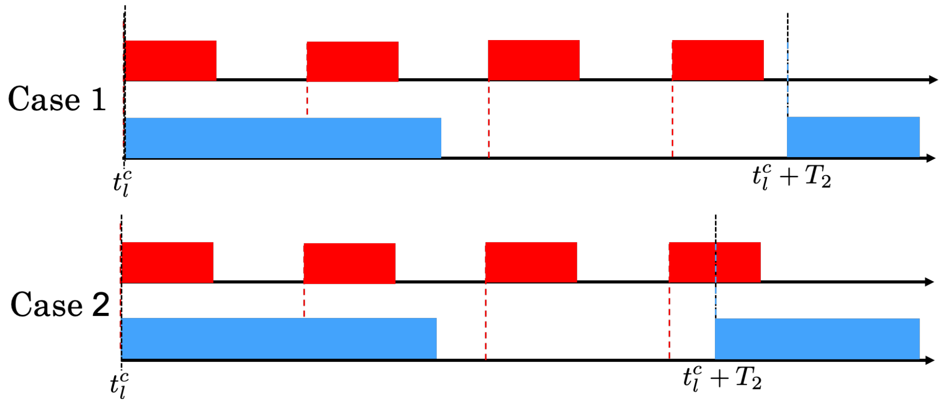


Figure 4.4: Two cases of utility bounds.

Theorem 4.5.5 *For a set of two tasks with fixed priority assignment, the least upper bound to the processor utilization factor is* $\overline{U}_{\min} = 2(2^{\frac{1}{2}} - 1)$.

Proof. Let τ_1, τ_2 be two tasks with periods T_1 and T_2, respectively. Assume that $T_1 \leq T_2$. According to RMS, τ_1 has higher priority. We will adjust C_2 to fully utilize the available shared resource time. Similar as Lemma 4.5.1, we need to consider two cases as illustrated in Figure 4.4:

Case 1: if $C_1 \leq T_1\{T_2/T_1\}$, then $\min(T_1\{T_2/T_1\}, C_1) = C_1$. Therefore, $C_2 \leq T_2 - C_1\lfloor T_2/T_1 \rfloor - C_1 = T_2 - C_1\lceil T_2/T_1 \rceil$ where $\lceil\cdot\rceil$ is the rounding up operator. Then $U = \frac{C_1}{T_1} + \frac{C_2}{T_2} \leq 1 + C_1\left[(1/T_1) - (1/T_2)\lceil T_2/T_1 \rceil\right]$ and U decreases if C_1 increases.

Case 2: if $C_1 > T_1\{T_2/T_1\}$, then $\min(T_1\{T_2/T_1\}, C_1) = T_1\{T_2/T_1\}$. Therefore, $C_2 \leq$

$T_2 - C_1 \lfloor T_2/T_1 \rfloor - T_1 \{T_2/T_1\} = -C_1 \lfloor T_2/T_1 \rfloor + T_2 \lfloor T_2/T_1 \rfloor$. Then $U = \frac{C_1}{T_1} + \frac{C_2}{T_2} \leq (T_2/T_1)\lfloor T_2/T_1 \rfloor + C_1 [(1/T_1) - (1/T_2)\lfloor T_2/T_1 \rfloor]$ and U increases if C_1 increases.

We define the upper bound of the utilization factor as

$$\overline{U}\left(C_1, \frac{T_2}{T_1}\right) = \begin{cases} 1 + C_1 [(1/T_1) - (1/T_2)\lceil T_2/T_1 \rceil], & \text{if } 0 \leq C_1 \leq T_1\{T_2/T_1\} \\ (T_2/T_1)\lfloor T_2/T_1 \rfloor + C_1 [(1/T_1) - (1/T_2)\lfloor T_2/T_1 \rfloor], & \text{if } C_1 > T_1\{T_2/T_1\} \end{cases}$$

When $C_1 = T_1\{T_2/T_1\}$, we have

$$\overline{U}\left(T_1\left\{\frac{T_2}{T_1}\right\}, \frac{T_2}{T_1}\right) = 1 - (T_1/T_2)\left[\lceil T_2/T_1 \rceil - (T_2/T_1)\right]\left[(T_2/T_1) - \lfloor T_2/T_1 \rfloor\right] \tag{4.20}$$

Let $I = \lfloor T_2/T_1 \rfloor$ and $f = \{T_2/T_1\}$, we can rewrite (4.20) as

$$\overline{U}(I, f) = 1 - f(1-f)/(I+f).$$

Since $\overline{U}(I, f)$ is monotonic increasing with I, the minimum of $\overline{U}(I, f)$ occurs at the smallest possible value of I, namely, $I = 1$. Then when minimizing U over f, we can take the derivative of $\overline{U}(1, f)$ with respect to f and have

$$\frac{\partial \overline{U}(I, f)}{\partial f} = \frac{f^2 + 2f - 1}{(1+f)^2}.$$

When $f = \sqrt{2} - 1$, $\frac{\partial \overline{U}_{\min}(1,f)}{\partial f} = 0$. And if $0 \leq f < \sqrt{2} - 1$, $\frac{\partial \overline{U}_{\min}(1,f)}{\partial f} < 0$ and if $\sqrt{2} - 1 < f < 1$, $\frac{\partial \overline{U}_{\min}(1,f)}{\partial f} > 0$, $\overline{U} = \overline{U}(2, \sqrt{2} - 1) = 2(\sqrt{2} - 1)$, which is the relation we want to prove. □

Corollary 4.5.5 *For a set of N tasks with fixed priority order, and the restriction that the ration between any two request periods is less than 2, the least upper bound to the processor utilization factor is $\overline{U}_{\min} = N(2^{\frac{1}{N}} - 1)$.*

Proof. The proof of this corollary is the same as the proofs of Theorem 4 in [9]. Here we will recap the proof. Let $C_1, C_2, ..., C_N$ be the execution times of the tasks that fully utilize the processor and minimize the processor utilization factor. Assume that $T_N > T_{N-1} > \cdots > T_2 > T_1$. Let U denote the processor utilization factor. We will show that $C_1 = T_2 - T_1$ by proof of contradiction.

First we assume that $C_1 = T_2 - T_1 + \Delta$ with $\Delta > 0$ and $C_2, ..., C_N$ be the execution times of the tasks that fully utilize the processor and minimize the processor utilization factor. Let $C'_1 = T_2 - T_1$. In order to let the tasks fully utilize the processor, we can let $C'_2 = C_2 + \Delta$ and we have $C_1 + C_2 = C'_1 + C'_2$ which means that C'_1 and C'_2 utilize the same amount of time as C_1 and C_2 in the time horizon $[0, T_2]$. Then we assume that $C'_3 = C_3, ..., C'_N = C_N$. Let U' denote the corresponding utilization factor of $C'_1, C'_2, ..., C'_N$. Since $C_1, C_2, ..., C_N$ fully utilize the processor and minimize the processor utilization factor, we have $U' \geq U$. However, if we compute the difference between U and U', then we have

$$\begin{aligned} U - U' &= \left(\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_N}{T_N}\right) - \left(\frac{C'_1}{T_1} + \frac{C'_2}{T_2} + \cdots + \frac{C'_N}{T_N}\right) = \frac{C_1 - C'_1}{T_1} + \frac{C_2 - C'_2}{T_2} \\ &= \frac{\Delta}{T_1} - \frac{\Delta}{T_2} \end{aligned}$$

Since $T_1 < T_2$, we have $\frac{\Delta}{T_1} - \frac{\Delta}{T_2} > 0$ which means that $U > U'$. It contradicts to the fact that $C_1, C_2, ..., C_N$ minimize the processor utilization factor.

Alternatively, we assume that $C_1 = T_2 - T_1 - \Delta$ with $\Delta > 0$ and $C_2, ..., C_N$ be the execution times of the tasks that fully utilize the processor and minimize the processor utilization factor. Let $C''_1 = T_2 - T_1$, $C''_2 = C_2 - 2\Delta$,..., and $C''_N = C_N$. Again, $C''_1, C''_2, ..., C''_N$ fully utilize the processor. Let U'' denote the corresponding utilization factor. We have

$$\begin{aligned} U - U'' &= \left(\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_N}{T_N}\right) - \left(\frac{C''_1}{T_1} + \frac{C''_2}{T_2} + \cdots + \frac{C''_N}{T_N}\right) = \frac{C_1 - C'_1}{T_1} + \frac{C_2 - C'_2}{T_2} \\ &= -\frac{\Delta}{T_1} + \frac{2\Delta}{T_2} > 0 \end{aligned}$$

which contradicts to the fact that $C_1, C_2, ..., C_N$ minimize the processor utilization factor.

Therefore, if U is the minimum utilization, then $C_1 = T_2 - T_1$. In a similar way, we can show that $C_2 = T_3 - T_2$, $C_4 = T_4 - T_3$, ..., $C_{N-1} = T_N - T_{N-1}$ and $C_N = T_N - 2(C_1 + C_2 + \cdots + C_{N-1})$.

To simplify the notation, we define $g_i = (T_N - T_i)/T_i$ where $i = 1, 2, ..., N$. Thus, $C_i = T_{i+1} - T_i = g_i T_i - g_{i+1} T_{i+1}$ where $i = 1, 2, ..., N-1$ and $C_N = T_N - 2g_1 T_1$.

And we have

$$\begin{aligned}\overline{U} &= \sum_{i=1}^{N} \frac{C_i}{T_i} = \sum_{i=1}^{N-1} \left[g_i - g_{i+1} \frac{T_{i+1}}{T_i} \right] + 1 - 2g_1 \frac{T_1}{T_N} \\ &= \sum_{i=1}^{N-1} \left[g_i - g_{i+1} \frac{g_i + 1}{g_{i+1} + 1} \right] + 1 - 2\frac{g_1}{g_1 + 1} \\ &= 1 + g_1 \frac{g_1 - 1}{g_1 + 1} + \sum_{i=2}^{N-1} g_i \frac{g_i - g_{i-1}}{g_i + 1}. \end{aligned} \tag{4.21}$$

To find the least upper bound to the utilization factor, (4.21) must be minimized over the g_j's. This can be done by setting the first derivative of U with respect to each of the g_j's equal to zero, and solving the resultant difference equations:

$$\frac{\partial \overline{U}}{\partial g_i} = \frac{g_j^2 + 2g_j - g_{j-1}}{(g_j + 1)^2} - \frac{g_{j+1}}{g_{j+1} + 1} = 0,\ j = 1, 2, ..., N-1. \tag{4.22}$$

with the definition $g_0 = 1$ for the convenience. The general solution to (4.22) is

$$g_j = 2^{(m-j)/m} - 1,\ j = 0, 1, ..., N-1. \tag{4.23}$$

It follows that

$$\overline{U}_{\min} = N\left(2^{\frac{1}{N}} - 1\right)$$

which is the result we want to prove. □

Then, we will remove the restriction that the ration between any two request periods is less than 2, which we can state as:

Theorem 4.5.6 *For a set of N tasks with fixed priority order, the least upper bound to the processor utilization factor is* $\overline{U}_{\min} = N(2^{\frac{1}{N}} - 1)$.

Proof. Let $\tau_1, \tau_2, ..., \tau_N$ be a set of N tasks that fully utilize the processor. Let U denote the corresponding utilization factor. Suppose that for some i, $\lfloor T_N/T_i \rfloor \geq 2$. To be specific, let $T_N = qT_i + r$, $q \geq 2$ and $r \geq 0$. We can replace the task τ_i by a task τ_i' such that $T_i' = qT_i$ and $C_i' = C_i$, and increase C_N by the amount needed to again fully utilize the processor. This increase is at most $C_i(q-1)$, the time within the critical time zone of τ_N occupied by τ_i but not by τ_i'. Let U' denote the utilization factor of such a set of tasks. We have $U' < U + [(q-1)C_i/T_N] + (C_i/T_i') - (C_i/T_i)$ or $U' \leq U + C_i(q-1)[1/(qT_i+r) - (1/qT_i)]$. Since $q - 1 > 0$ and $1/(qT_i + r) - (1/qT_i) \leq 0$, we have $U' \leq U$. Therefore we conclude that in determining the least upper bound of the processor utilization factor, we need only consider task sets in which the ratio between any two request periods is less than 2. The theorem thus follows directly from Corollary 4.5.5. $\square$

CHAPTER 5
CONTENTION-RESOLVING MPC ALGORITHM

In this Chapter, we propose a novel method to solve the mixed integer programming problem associated with contention-resolving MPC formulated in Chapter 3.3 and introduce a general framework for the contention-resolving MPC algorithm. The proposed method converts the difficult MIP into a path planning problem that can be solved iteratively. The key idea of this method is based on two insights. First, we only need to assign priorities at the significant moments when contentions occur, which are a finite number of time instances on $[t_0, t_f]$. Besides, at each contention moment, there are only a finite number of customers competing for the resource. Each assignment of the priority to the finite number of customers will produce a branch of a decision tree, as illustrated by Figure 5.1. The tree will contain a finite number of branches, and an optimal solution must be a path from the root of the tree at the starting time t_0 to one of the leaves at time t_f. There are only finitely many such paths that can be searched. Second, among the finitely many paths, not all need to be searched to find the optimal solution. A search algorithm such as the A-star can efficiently search the branches that most likely constructing the optimal path.

5.1 Contention Detection

The first step of our method is to find the significant moments when contentions occur. The significant moment analysis and timing model offer a natural way to detect the contention moments. The following propositions explain how to detect contentions:

Proposition 5.1.1 *In preemptive scheduling, a contention starts at time t if and only if the* *following three conditions hold:*

$$\sum_{i=1}^{N} \mathsf{sgn}(r_i(t)) \geq 2, \ \sum_{i=1}^{N} \mathsf{sgn}(r_i(t^-)) \leq 1, \ \textit{and } t = \alpha_i[k] \textit{ for some } i \textit{ and some } k. \tag{5.1}$$

Proof. Based on Definition 4.1.2, if a control system i has not finished the current task at t, then $r_i(t) > 0$ and $\mathsf{sgn}(r_i(t)) = 1$. Since $r_i(t)$ is always non-negative, $\mathsf{sgn}(r_i(t)) \geq 0$ for all t. Therefore,

$$\sum_{i=1}^{N} \mathsf{sgn}(r_i(t)) \geq 2$$

is equivalent to two or more customers wanting to access the shared resource, which means a contention is occurring at time t. Since

$$\sum_{i=1}^{N} \mathsf{sgn}(r_i(t^-)) \leq 1$$

means that no contention happens at time instants before t that are close to t, the result follows. □

Proposition 5.1.2 *In non-preemptive scheduling, a contention starts at time t if and only if t is a significant moment t_w that satisfies the following two conditions hold:*

$$\sum_{i=1}^{N} [1 - \mathsf{sgn}(C_i(t_w) - r_i(t_w))] \geq 2, \ r_{\mathrm{ID}}(t_w^-) = 0 \tag{5.2}$$

where $r_{\mathrm{ID}}(t)$ is a simplified notation for the remaining time $r_{\mathrm{ID}(t)}(t)$ of timing state variable ID *at any t and t_w is a significant moment computed by equation (6.5).*

Proof. For non-preemptive scheduling, a task for a control system i is waiting to get access to the shared resource at a time t or is generated at time t if and only if $r_i(t) = C_i(t)$, i.e., $1 - \mathsf{sgn}(C_i(t) - r_i(t)) = 1$. Therefore, $\sum_{i=1}^{N} [1 - \mathsf{sgn}(C_i(t) - r_i(t))] \geq 2$ if and only if two or more control systems are waiting for access to the shared resource at time t or generating tasks at time t. Therefore, for necessity, if a contention starts at time t, then t

is one significant moment t_w for some i and w, and the highest prioritized control system among the contending control systems at time t will either finish a task at time t and then start a new task at time t using the shared resource, or else it will go from not occupying the shared resource to occupying the shared resource at time t, so the condition $r_{\mathrm{ID}}(t^-) = 0$ from (5.2) holds. For sufficiency, if the two conditions (5.2) are satisfied, then at time t_w, multiple control systems are in contention for the shared resource which must be a time when some control system requests usage of the shared resource, so a contention starts at time t. $\square$

Based on the contention moments, we introduce a tree structured directed graph which will be used to model how different priority assignments affect the system behavior and analyze our algorithm. Figure 5.1 shows an example of decision tree. In the decision tree, each leaf represents a contention time satisfying Proposition 5.1.1 or 5.1.2. In this figure, the blue circle represents the root of the decision tree, and grey circles and dots represent internal leaves. The decision tree is expanded in the direction of the arrows, which represent branches. The integers in brackets represent the priorities. The red cross means that the branch does not satisfy the schedulability test and will lead to infeasible solutions, therefore, it has been cut off. The bottom sub-figure shows the schedule along the green path. Colored rectangles without diagonal lines in the lower figure represent the time delay δ_i. Colored rectangles with diagonal lines represent times when each control system occupies the resource.

We denote the contention times by t_l^c where l is the index of its corresponding leaf. At each contention time, there are only a finite number of control systems competing for the resource. Each possible assignment of the priority to the finite number of control systems will produce a branch of a decision tree.

Remark 2 *The construction of the entire decision tree is not necessary for contention resolving MPC to search for an optimal solution. However, for the purpose of clearly presenting the concept for the sampling based optimization method, we will discuss how the*

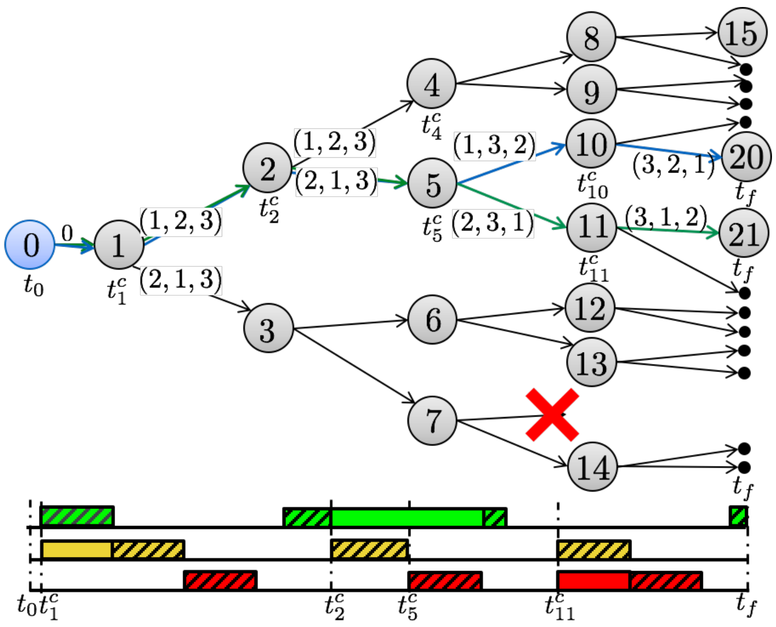


Figure 5.1: Decision tree to solve the co-design problem for preemptive scheduling within a finite time window.

tree can be fully constructed.

5.2 Construction of Decision Tree

The decision tree construction starts from the root v_0 associated with the MPC starting time t_0. The construction is performed iteratively. During the construction, if a leaf has no branches pointing out from it, then it is called *unexpanded*. At each iteration, new branches are generated from each unexpanded leaf and new leaves are generated at the end of each branch. For an unexpanded leaf l, let $\Lambda(t_l^c)$ denote the set of control systems having contentions at a contention time t_l^c, where $\Lambda(t_l^c) = \{i \in \{1, .., N\} : r_i(t_l^c) > 0\}$ for preemptive scheduling and $\Lambda(t_l^c) = \{i \in \{1, .., N\} : r_i(t_l^c) = C_i(t_l^c)\}$ for non-preemptive scheduling. Also, M is the number of elements of $\Lambda(t_l^c)$. Let $\mathbf{P}_m$ denote the m-th permutation in $\mathcal{P}(\{1, ..., M\})$, so $m \in \{1, 2, .., M!\}$. For leaf l, we generate $M!$ branches from it. Each branch corresponds to a unique choice of the priority assignment in $\mathcal{P}(\{1, ..., M\})$. The m-th branch expands from v_l and connects to a new leaf v_{j+m} based on $\mathbf{P}_m$, where j is the number of existing leaves in the tree before we generate new branches from leaf v_l.

We say that the leaf v_{j+m} is a child leaf of v_l or leaf v_l is the parent leaf of v_{j+m}. The contention time associated with the leaf v_{j+m} is the next contention time occurring after t_l^c that is scheduled by priority assignment $\mathbf{P}_m$. Different branches may end with different next contention times after t_l^c. The iterative construction terminates when the contention times of all unexpanded leaves are greater than or equal to t_f. We call these unexpanded leaves *terminal leaves* and assign t_f to them as their contention times.

Let us revisit the example shown in Figure 5.1. Contentions happen four times across the time interval $[t_0, t_f]$. At the first contention time t_1^c, control system 1 and 2 have a contention. The leaf 1 has two branches corresponding to the $2! = 2$ different priority assignments. Similarly, at each of the following three contention times, two control systems have contentions, and each leaf has two branches corresponding to two different priority assignments.

5.3 Schedulability (Feasibility) Test

To guarantee normal operation of the real-time system, the first task is to check whether all the control systems are schedulable under the priority assignment along one branch. When constructing the decision tree, it may expand a branch (l, j) with its associated priority assignment $\mathbf{P}_m$ which will unavoidably lead to unschedulablility. This means that some tasks scheduled under the priority assignment $\mathbf{P}_m$ within $\left[t_l^c, t_j^c\right]$ are not schedulable or based on the timing state values of leaf v_j, some tasks are not schedulable under any priority assignments. Then we should prune branch (l, j) instead of generating this branch and leaf v_j. To identify the branch and leaves that will lead to unschedulablility, we need conditions to test the schedulability given the timing states of a leaf.

When constructing a branch (l, j) with its associated priority assignment $\mathbf{P}_m$. The schedulability test are divided into two steps as illustrated in Figure 5.2:

1. *Finite-time schedulability test* for branch (l, j) under fixed priority assignment $\mathbf{P}_m$ within the finite-time window $\left[t_l^c, t_j^c\right]$, where t_l^c and t_j^c are the contention time instants of leaves v_l

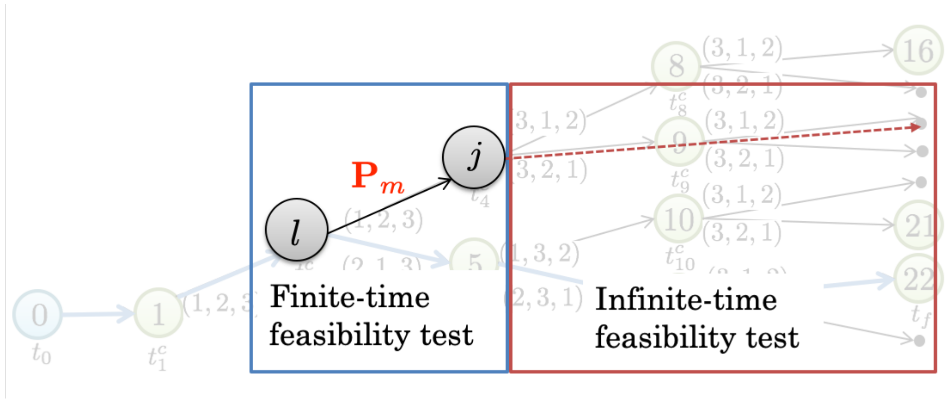


Figure 5.2: Illustration of the schedulability test when constructing a branch (l, j) with associated priority assignment $\mathbf{P}_m$.

and v_j, respectively.

2. *Infinite-time schedulability test* after the contention time instant of leaf v_j under the "best" priority assignment. For example, for periodic and preemptive systems, EDF has been proved to be the best scheduling methods in [9].

If the branch (l, j) fails any of the above schedulability tests, then this branch leads to infeasible solutions. Therefore, it need to be pruned when constructing the decision tree, as the one be cover by the red cross in Figure 5.1.

5.3.1 Finite-time Window Schedulability Test

Since the priority assignment within the time interval is selected to be the fixed $\mathbf{P}_m$, we need to perform the schedulability analysis as follows:

Definition 5.3.1 *A finite-time schedulability test over a time interval $[t_l^c, t_j^c]$ checks if all tasks for all control systems are able to meet their deadlines within $[t_l^c, t_j^c]$ under the priority assignment $\mathbf{P}_m$.*

Consider a set of tasks $\Gamma = \{\tau_1, ..., \tau_i, ..., \tau_N\}$ within $[t_l^c, t_j^c]$. The task set Γ is schedulable within $[t_l^c, t_j^c]$ if and only if Γ is schedulable within each sub-interval $[t_w, t_{w+1}) \subseteq [t_l^c, t_j^c]$, and Γ is schedulable within a sub-interval $[t_w, t_{w+1})$ if and only if each individual task $\tau_i \in \Gamma$ is schedulable within $[t_w, t_{w+1})$. The following theorem states the necessary and sufficient conditions for the schedulability of τ_i within a sub-interval $[t_w, t_{w+1})$.

Theorem 5.3.1 *A task τ_i is schedulable within $[t_w, t_{w+1})$ if and only if it satisfy ONE of the following conditions:*

1. $o_i(t_{w+1}^-) = T_i(t_{w+1}^-)$ *and* $r_i(t_{w+1}^-) = 0$*;*

2. $o_i(t_{w+1}^-) < T_i(t_{w+1}^-)$.

Proof. If the dynamic response time of τ_i is equal to its relative deadline at time t_{w+1}^-, i.e. $o_i(t_{w+1}^-) = T_i(t_{w+1}^-)$, then the schedulability of τ_i within $[t_w, t_{w+1})$ is satisfied if and only if the effective task of τ_i has completed before time t_{w+1}^-, i.e. $r_i(t_{w+1}^-) = 0$.

If the dynamic response time of τ_i is smaller than its relative deadline at time t_{w+1}^-, i.e. $o_i(t_{w+1}^-) < T_i(t_{w+1}^-)$, the schedulability of τ_i within $[t_w, t_{w+1}^-)$ is automatically guaranteed. $\square$

At any time t_l^c, given the task characteristics $\{C_i(t), T_i(t)\}_{i=1}^N$ for $t \in \left[t_l^c, t_j^c\right]$, we can use Algorithm 1 to perform the dynamic schedulability test over the time interval $\left[t_l^c, t_j^c\right]$. Algorithm 1 iteratively checks the schedulability of Γ within each sub-interval in the following ways:

1. At the beginning of any sub-interval, it calculates the end of the current sub-interval according to equation (4.3), as shown in Lines 12 of Algorithm 1.

2. It utilizes the dynamic timing model in equation (4.2) to obtain the values of the timing state variables at the end of the current sub-interval, as indicated by Line 12.

3. It evaluates the schedulability of τ_i, where $i = 1, ..., N$, within $[t_w, t_{w+1}^-]$ according to Theorem 5.3.1, as shown in Lines $14 - 22$. To make the sub-interval propagates seamlessly within $\left[t_l^c, t_j^c\right]$, it assigns the starting time of the next sub-interval to be the ending time of the current sub-interval, as indicated by Line 23.

The variable $\mathrm{ds}_i[w]$ indicates the dynamic schedulability test result of τ_i within $[t_w, t_{w+1}^-]$: when τ_i is schedulable within $[t_w, t_{w+1}^-]$, $\mathrm{ds}_i[w] = 1$; otherwise, $\mathrm{ds}_i[w] = 0$. The set

Algorithm 1 Dynamic Schedulability Test

```
1: **Data**: $t_l^c$, $t_j^c$, $Z(t_l^c)$, $\{C_i(t), T_i(t)\}_{i=1}^N$
2: **Result**: $\{\mathrm{DS_i}\}_{\mathrm{i=1}}^{\mathrm{N}}$
3: $t_w = t_l^c$;
4: for each $\tau_i \in \Gamma$ do
5:     $\mathrm{DS_i} = [\,]$;
6: while $t_w < t_j^c$ do
7:     for each $\tau_i \in \Gamma$ do
8:         if $d_i(t_w^-) = 0$ then
9:             $d_i(t_w) = T_i(t_w)$;
10:        else
11:            $d_i(t_w) = d_i(t_w^-)$;
12:    $t_{w+1} = t_w + \min\{d_1(t_w), ..., d_N(t_w), t_j^c - t_w\}$;
13:    $Z(t_{w+1}^-) = \mathbb{H}\Big(t_{w+1}^-; Z(t_w^-), \mathbf{S}, \mathbf{P}_m\Big)$;
14:    for each $\tau_i \in \Gamma$ do
15:        if $d_i(t_{w+1}^-) = 0$ then
16:            if $o_i(t_{w+1}^-) < T_i(t_{w+1}^-)$ then
17:                $\mathrm{ds_i} = 1$;
18:            else
19:                $\mathrm{ds_i} = 0$;
20:        else
21:            $\mathrm{ds_i} = 1$;
22:        $\mathrm{DS_i} = \{\mathrm{DS_i}, \mathrm{ds_i}\}$;
23:    $w = w + 1$;
```

$\mathrm{DS}_i = \{\, ds_i[1],\, ds_i[2], \cdots\}$ contains the dynamic schedulability test results of τ_i within all sub-intervals that belong to $\left[t_l^c, t_j^c\right]$. The task τ_i is schedulable within $\left[t_l^c, t_j^c\right]$ if and only if $\min_w\{\mathrm{DS_i}\} = 1$.

Condition 5.3.1 ***(Finte-time Schedulability)****: The task set Γ is schedulable within $\left[t_l^c, t_j^c\right]$ if and only if all individual tasks are dynamically schedulable within $\left[t_l^c, t_j^c\right]$, i.e.*

$$\min_{1\le i\le N}\left\{\min_{\underline{w}\le w\le \overline{w}} ds_i[w]\right\} = 1.$$

where $\underline{w}$ is the smallest integer w satisfying $t_w \ge t_l^c$ and $\overline{w}$ is the largest integer w satisfying $t_w \le t_j^c$.

5.3.2 Infinite-time Window Schedulability Test

The condition can be formualted as

Condition 5.3.2 ***(Infinte-time Schedulability)****: For a leaf j, if $\mathrm{DS}^{\infty}\left(Z(t_j^c)\right) \leq 0$, then the system is schedulable for any time $t \geq t_j^c$.*

The function $\mathrm{DS}^{\infty}\left(Z(t_j^c)\right)$ can be viewed as the maximal utility bound given the initial timing state of leaf j minus a constant. For systems with different task models, we need to derive different formula for $\mathrm{DS}^{\infty}\left(Z(t_j^c)\right)$.

For example, as we already shown in Section 4.5, for a preemptive and periodic system under fixed priority assignment, the function is

$$\mathrm{DS}^{\infty}\left(Z(t_j^c)\right) = \sum_{i=1}^{N} \frac{C_i}{T_i} - N(2^{\frac{1}{N}} - 1).$$

For a preemptive and periodic system under dynamic priority assignment, the function is

$$\mathrm{DS}^{\infty}\left(Z(t_j^c)\right) = \sum_{i=1}^{N} \frac{C_i}{T_i} - 1. \tag{5.3}$$

As already proved in [110], if (1) all the timing characteristics of all system are integers and (2) there is no inserted idle time, then the sufficient and necessary condition for infinte-time schedulability of non-preemptive system is

Theorem 5.3.2 *A set of tasks is schedulable without preemption if and only if the following two conditions are satisfied:*

1. $\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$;
2. $\forall i,\ 1 < i \leq N; \forall L,\ T_1 < L < T_i:\ L \geq C_i + \sum_{j=1}^{i-1} C_j \left\lfloor \frac{L-1}{T_j} \right\rfloor$.

The assumption (1) that the timing parameters are integers can be satisfied in a discrete-time systems, which we will discuss in Chaper 8. The assumption (2) that there is no inserted idle time is equivalent to the CIA assumption. Therefore, for a discrete-time non-preemptive system, the function are

$$\begin{aligned}&\mathrm{DS}_0^\infty = \sum_{i=1}^{N} \frac{C_i}{T_i} - 1 \text{ and}\\ &\mathrm{DS}_i^\infty (L) = C_i + \sum_{j=1}^{i-1} C_j \left\lfloor \frac{L-1}{T_j} \right\rfloor - L, \forall i,\, 1 < i \leq N;\, \forall L,\, T_1 < L < T_i.\end{aligned} \tag{5.4}$$

5.4 Branch Cost

After constructing the decision tree and pruning the infeasible branches, we define a cost for each branch. Along one branch (l, j) whose associated priority assignment is $\mathbf{P}_m$, we first calculate the significant moments $\gamma_i[k]$ for all i and k such that $t_l^c \leq \gamma_i[k] \leq t_j^c$,

$$Z(t) = \mathbb{H}\Big(t; Z(t_l^c), \mathbf{S}, \mathbf{P}_m\Big) \text{ and } \gamma_i[k] = \alpha_i[k] + o_i(\alpha_i[k+1]^-) \tag{5.5}$$

where $o_i(\alpha_i[k+1]^-)$ for each k is generated by the timing model except with a known priority assignment $\mathbf{P}_m$ instead of all possible priority assignments as in (6.4c). Then the branch cost $w_{l,j}$ is defined as

$$w_{l,j} = \sum_{i=1}^{N} w_{l,j}^i \tag{5.6}$$

where $w_{l,j}^i$ is the cost of control system i and it can be computed by solving the following optimization problem based on the significant moments calculated along a branch. For each i such that there is a completion time $\gamma_i[k+1] \in (t_l^c, t_j^c]$, let $\underline{k}_i$ be the smallest index k satisfying $\gamma_i[k+1] > t_l^c$ and $\overline{k}_i$ be the largest index k satisfying $\gamma_i[k+1] \leq t_j^c$. Then we set

$$w_{l,j}^i = \sum_{k=\underline{k}_i}^{\overline{k}_i} \min_{u_i[k]} V_i(u_i[k]; x_i(\alpha_i[k]), \gamma_i[k], \gamma_i[k+1]) \text{ subject to } (4.11c) \text{ and } (4.11d). \tag{5.7}$$

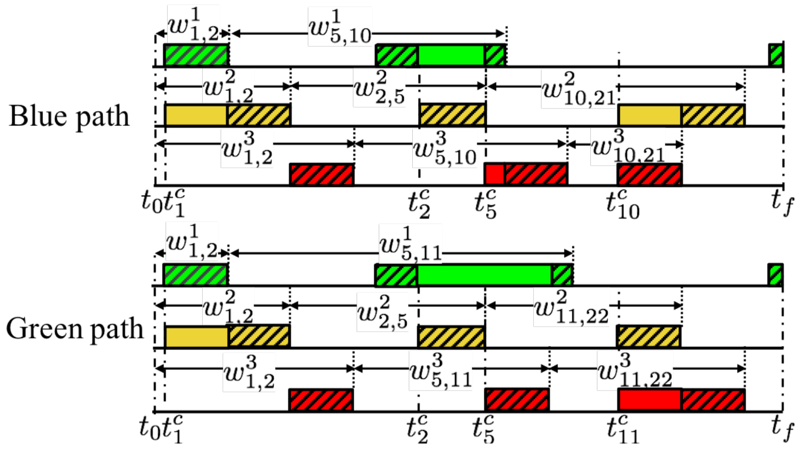


Figure 5.3: Illustration of branch cost along a path. The ending time of the colored rectangles with diagonal lines represent the task completion time γ_i.

The meaning of (5.7) is as follows. If the $(k+1)$-st task of control system i is completed between the contention times t^c_l and t^c_j, i.e. $\gamma_i[k+1] \in (t^c_l, t^c_j]$, then the cost within the time interval $[\gamma_i[k], \gamma_i[k+1]]$ is included in the branch cost $w_{l,j}$. If no task request of control system i is completed within $(t^c_l, t^c_j]$, then we set $w^i_{l,j} = 0$. This branch cost formulation ensures that all costs included in one branch are determined and will not be changed by the priority assignments at or after time t^c_j. The cost of the uncompleted $(\overline{k}_i+1)$-st task will be included by the branches following the branch (l, j).

Figure 5.3 shows an illustration of the defined branch cost for the blue path and green path in Figure 5.1. The different priority assignments at t^c_5 caused different branch cost computation. In the blue path, the second cost of control system 1 considers a shorter time interval than the second cost of control system 1 in the green path.

Remark 3 *Along any arbitrary path in the decision tree, all the significant moments are deterministic and can be computed by the timing model. For any $\gamma_i[k+1]$ along this path, we can always find the consecutive contention times t^c_l and t^c_j such that $\gamma_i[k+1] \in [t^c_l, t^c_j)$ and the cost of the task before $\gamma_i[k+1]$ is added in the branch cost.*

Remark 4 *The optimal control design is embedded in the branch cost calculation. To calculate $w^i_{n,j}$ in (5.7), we need to solve the optimization problem (5.7) by optimizing the* *control law $u_i(t)$. Since the priority along one branch is already knowm, we can use the MPC design methods from [27] and [111] to solve (5.7) and compute $u_i^*(t)$. After solving (5.7) for each control system i, we obtain the optimal control $\mathbf{u}^*(t)$ between two successive contention time instants.*

Based on the decision tree model, the MIP formulated in (4.11a) can now be converted to the problem of finding a path from the root v_0 to a terminal leaf such that the cost along the entire path is the lowest. The constructed decision tree contains multiple paths and the total path cost has the same formula as the cost function in (4.11a). Among all the paths, the lowest cost path can be found by path planning algorithms [112] and the priority assignments and control commands along the lowest cost path will be solutions for the MIP problem.

However, constructing the entire decision tree would be exhaustive and unrealistic when considering a relatively large number of control systems or a long time window. This motivates Section 5.5, where we propose a search algorithm that only needs to construct a subtree of the decision tree while searching for an optimal path. This method is inspired by the A-star algorithm [40] that has been widely used for online path planning in robotics, which has been found to significantly reduce computation time. We present proofs to show that optimality is guaranteed using our proposed algorithm in Section 5.7.

5.5 Costs for Search Algorithm

The A-star algorithm will iteratively generate and search the leaves starting from the root and terminate when it reaches a terminal leaf. To use the A-star algorithm, we define leaves in two categories: i) If a leaf has been generated and all its child leaves have been generated by the search algorithm, then we call such a leaf *closed*. ii) If a leaf has been generated and at least one of its child leaves has not been generated by the search algorithm, then we call such a leaf *open*. If a leaf is open and its parent leaf is closed, then the leaf is called a *frontier leaf*. All frontier leaves are added to a set called the *frontier list*, which keeps track of the leaves that can be expanded by the A-star algorithm. The *frontier list* is a sorted list where all the leaves in it are sorted according to a function

$$C_f(v_l) = C_g(v_l) + C_h(v_l) \tag{5.8}$$

from the smallest to largest value where l is the index of a leaf. The function $C_g(v_l)$ is called the stage cost, which is the sum of branch costs along the path starting from the root to the current leaf v_l and $C_h(v_l)$ is the minimal future cost from the current leaf v_l to a terminal leaf where the minimization is over all priority assignments and allowable controls.

Since the path from the root v_0 to a leaf v_l is unique, the stage cost can be computed using $C_g(v_l) = C_g(v_p) + w_{p,l}$ where p is the index of the parent leaf of v_l. For the A-star algorithm to work, an estimation $\hat{C}_h(v_l)$ of future cost (also called the heuristic cost) is needed for which $\hat{C}_h(v_l) \leq C_h(v_l)$ for all v_l, so the estimated cost $\hat{C}_f(v_l) = C_g(v_l) + \hat{C}_h(v_l)$ equals to the actual cost $C_f(v_l)$ when v_l is a terminal leaf. The value of the MPC cost function may be increased because of the contentions. Using this monotone property of the cost function, we can estimate the future cost $\hat{C}_h(v_l)$ by solving the following MPC optimization problem

$$\hat{C}_h(v_l) = \min_{\mathbf{u}^h(t)} \sum_{i=1}^{N} V_i(u_i^h(t); x_i(t_l^c), u_i(t_l^c), t_l^c, t_f), \tag{5.9}$$

$$\text{s.t. } \dot{x}_i(t) = f_i(x_i(t), u_i^h(t)),\ u_i^h(t) \in \mathbb{U}_i, \text{for all } t$$

where $\mathbf{u}^h(t) = (u_1^h(t), ..., u_i^h(t), ..., u_N^h(t))$ is Lebesgue measurable and essentially bounded (as defined for instance in [96]), and t_l^c is the contention time instant corresponding to leaf v_l. Notice that the above optimization problem does not have the contention constraints from (6.4c).

During the search, all leaves v in the *frontier list* are sorted according to their $\hat{C}_f(v)$

value, from the smallest to the largest. At each iteration, the algorithm expands the leaf with the smallest $\hat{C}_f$ by generating all its child leaves and then removes the expanded leaf from the *frontier list*. All of its child leaves are added to the *frontier list*. The heuristic cost $\hat{C}_h(v_l)$ will make it possible to search the most promising paths first, and the optimal solution can be found without examining all possible paths. Therefore, the search algorithm leveraging A-star does not generate the entire decision tree.

In addition to the *frontier list*, we also have a *generated set* which consists of all leaves that have been generated by the A-star algorithm. Each leaf v_l in the *generated set* is also assigned a pointer $PT(v_l)$ which equals the index of its parent leaf so that the A-star algorithm can backtrack from it to its parent leaf.

5.6 Contention-resolving MPC Algorithm

Algorithms 2-4 present the pseudocode for our proposed algorithm based on the A-star algorithm to solve the optimization problem (3.3). Algorithm 2 presents the search algorithm. The optimal path search starts from the root v_0. The search algorithm keeps updating two sets, which are the *frontier list* and the *generated set*. At the beginning of the search algorithm, the root leaf v_0 is added in the *frontier list*. The *generated set* only contains the root leaf v_0 initially. Let $\hat{C}_f(v_0)$ equal the heuristic cost $\hat{C}_h(v_0)$. At each iteration of the main program in Algorithm 2, the algorithm determines which leaf to expand further by selecting the leaf v_l with minimal $\hat{C}_f$ cost in the frontier list. After selecting the leaf v_l, there are two cases that need to be considered:

1. If the contention time instant of the selected leaf equals t_f, then the search algorithm has found the path from the root leaf to a terminal leaf with the lowest $\hat{C}_f$ cost, which equals the actual cost C_f. The search algorithm is terminated.

2. If the contention time instant of the selected leaf does not equal t_f, then leaf v_l will be expanded by generating its children leaves and all of its children leaves are added

Algorithm 2 Main Program

```
1: Data: $t_0$, $t_f$, $\lambda_i$ for $1 \leq i \leq N$, $\mathbf{x}(t_0)$, $\mathbf{u}(t_0)$, $Z(t_0)$
2: Result: $\mathbf{P}^*(t)$, $\mathbf{u}^*(t)$
3: Let frontier list=generated set= $\{v_0\}$;
4: $\hat{C}_f(v_0) = \hat{C}_h(v_0), t = t_0$;
5: while $t_l^c \leq t_f$ do
6:     $v_l$ is the leaf in frontier list with minimal $\hat{C}_f$ cost;
7:     $t_l^c$ is the contention time instant corresponding to $v_l$;
8:     Let $p = PT(v_l)$;                    ▷ $v_p$ is the parent leaf of leaf $v_l$.
9:     if $t_l^c = t_f$ then
10:        return Reconstruct($v_l$); Break;
11:    else
12:        $j$ is the number of elements in generated set;
13:        for m-th permutation $\mathbf{P}_m \in \mathcal{P}(\{1, ..., M\})$ do
14:            $(v_{j+m}, t_{j+m}^c, w_{l,j+m})$=Expand($v_l$, $\mathbf{P}_m$, $t_l^c$);
15:            if $(l, j+m)$ passes both schedulability conditions under $\mathbf{P}_m$ then
16:                Add $v_{j+m}$ into frontier list and generated set;
17:                $C_g(v_{j+m}) = C_g(v_l) + w_{l,j+m}$;
18:                Solve (5.9) to obtain $\hat{C}_h(v_{j+m})$;
19:                $\hat{C}_f(v_{j+m}) = C_g(v_{j+m}) + \hat{C}_h(v_{j+m})$;
20:                $PT(v_{j+m}) = l$;
21:        Remove $v_l$ from frontier list;
```

to *frontier list* and *generated set*. Then the algorithm calculates the costs $\hat{C}_f$ for the children leaves. Since the leaf v_l has child leaves after the expansion, it is not a *frontier leaf*. The search algorithm removes the expanded leaf v_l from *frontier list*. Then the algorithm goes to the next iteration.

Algorithm 3 backtracks the path from the selected terminal leaf to v_0 when case (1) is satisfied in the search algorithm. The backtracking starts from the terminal leaf v_l and utilizes the pointer $PT(v_l)$ to obtain the parent leaf v_p. The optimal priority assignment $\mathbf{P}^*(t)$ for the time interval between the contention time instants of v_p and v_l equals the priority assignment along the branch connecting v_p and v_l. Then we repeat this process with v_l and v_p replaced by v_p and the parent leaf of v_p, respectively. We repeat the backtracking process to obtain the optimal priority assignment $\mathbf{P}^*(t)$ until the contention time instant equals t_0. Algorithm 3 returns the optimal priority assignment $\mathbf{P}^*(t)$ for all $t \in [t_0, t_f]$ to

Algorithm 3 Reconstruct

```
1: Data: $v_l$
2: Let $t = t_f$ and $p = PT(v_l)$;
3: while $t > t_0$ do
4:     Let $\mathbf{P}^*(t)$ be the priority assigned to the branch that connects $v_p$ and $v_l$, from the
       contention time $t_p^c$ of leaf $v_p$ to the contention time $t_l^c$ of $v_l$;
5:     Let $l = p$ and $p = PT(v_p)$;
6:     Let $t$ be the corresponding contention time of $v_l$;
7: return $\mathbf{P}^*(t)$;
```

Algorithm 4 Expand

```
1: Data: $v_l$, $\mathbf{P}_m$, $t$
2: Find the next contention time under priority $\mathbf{P}_m$ based on (5.1) or (5.2), and denote
   this contention time as $t_{j+m}^c$;
3: Check the finite-time using Algorithm 1 and infinite-time schedulability condition;
4: if schedulable then
5:     Solve the optimization formulated by (5.7) to obtain $u_i^*(t)$ and compute $w_{l,j+m}^i$ for
       each $i = 1, ..., N$;
6:     Compute $w_{l,j+m}$ using (5.6);
7: return $v_{j+m}$, $t_{j+m}^c$, $w_{l,j+m}$;
```

the main program in Algorithm 2.

Algorithm 4 expands the selected leaf from the *frontier list* when case (2) is satisfied in the search algorithm. It utilizes Proposition 5.1.1 or 5.1.2 to determine the next contention time after a contention time t. Then it checks both finite-time and infinite-time schedulability conditions. If the branch is schedulable, then the algorithm solves the optimization problem (5.7) to obtain the optimal control $u_i^*(t)$ and compute the branch cost $w_{l,j+m}$. Algorithm 4 returns the child leaf v_{j+m}, the next contention time t_{j+m}^c and the branch cost $w_{l,j+m}$ to the main program in Algorithm 2.

Figure 5.4 is an illustration of the subtree constructed by our algorithm described above, using the same example as Figure 5.1. Compared with the entire decision tree in Figure 5.1, some internal leaves in the subtree are open because our algorithm does not expand every leaf but intelligently expands a subset of leaves without losing optimality. Once the construction of the subtree reaches the terminal leaf, our algorithm backtracks the path along the red arrows. The total number of branches generated by the algorithm is 11,

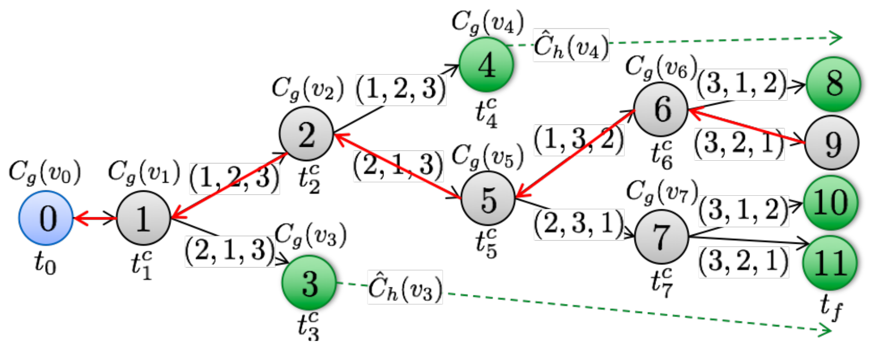


Figure 5.4: Illustration of the subtree constructed by the proposed search algorithm. The blue circle represents the root v_0 and the red circle represents the terminal leaf. Green circles represent leaves in the *frontier list*. Solid black arrows represent branches generated by the algorithm and dashed green arrows represent the estimate cost $\hat{C}_h(v_l)$. The red arrows represent the path with lowest cost.

reducing the computational workload for generating the entire tree which has totally 28 branches as shown in Figure 5.1.

5.7 Proof of Optimality

In this section, we prove that our algorithm finds the optimal solutions $\mathbf{P}^*(t)$ and $\mathbf{u}^*(t)$ which minimize (3.3). We first show that the heuristic cost $\hat{C}_h(v_l)$ defined in Section 5.5 satisfies the requirements for the A-star algorithm.

Proposition 5.7.1 *The condition $\hat{C}_h(v_l) \leq C_h(v_l)$ holds for all v_l in the decision tree.*

Proof. The estimated cost $\hat{C}_h(v_l)$ is obtained by solving the optimization problem (5.9). The actual future cost $C_h(v_l)$ is obtained by solving the optimization problem defined by (3.3) given the initial condition $\mathbf{x}(t_l^c)$. Comparing (5.9) and (3.3) with $\mathbf{x}(t_l^c)$, these two optimization problems have the same cost function and initial conditions. The differences are that the decision variable $\mathbf{u}(t)$ in (3.3) is constrained to be piecewise constant function that depends on the priorities, while $\mathbf{u}^h(t)$ in (5.9) can be any arbitrary real valued function as long as it is Lebesgue measurable and essentially bounded. Therefore, the optimal solution $\mathbf{u}^*(t)$ in (3.3) must be feasible but may not be an optimal solution for (5.9). Hence, $\hat{C}_h(v_l)$ is less or equal to $C_h(v_l)$ for all v_l. □

Theorem 5.7.1 *Based on Assumptions 3.1.1, 3.1.2 and 3.2.2, Algorithm 2 finds an optimal solution $\mathbf{P}^*(t)$ and $\mathbf{u}^*(t)$ for the optimization problem (4.11a).*

Proof. From [40, Theorem 1], the A-star algorithm finds the minimal total cost from v_0 to a terminal leaf if $\hat{C}_h(v_l) \leq C_h(v_l)$ for all v_l. Since we already showed that this condition is satisfied in Proposition 5.7.1, the theorem follows.

Remark 5 *Since $\mathcal{P}(\{1, ..., M\})$ contains all possible priority assignments, it also includes the priorities following RMS, EDF, FCFS, HSF and HTF rules. Therefore, the priorities assigned by the RMS, EDF, FCFS, HSF or HTF strategies are represented by paths in the decision tree, but not necessarily the path with the minimal cost. Therefore, our method guarantees that we find a better or the same solution as these strategies.*

CHAPTER 6
APPLICATION 1: NETWORKED CONTROL SYSTEM

The networked control system is a system in which feedback control loops are closed via shared communication media. The integration of the communication and feedback control loops can bring many benefits, such as system flexibility and easy maintenance. In recent years, a number of results have been reported on designing networked control system [64]. Sensors, controllers and actuators connected to the network are regarded as nodes of networked control systems (NCSs). The bandwidth for communication between nodes is mostly limited in NCSs, disallowing sensor messages to transmit immediately after generation, and this causes time delays in the NCSs [6]. A challenge for controlling event-triggered network systems lies in the integration of control with time delays. In this chapter, we propose the contention-resolving model predictive control method to dynamically assign priorities for control systems in event-triggered NCSs, to minimize the overall performance degradation caused by time delays in the network.

6.1 NCS Models

We consider an NCS with N independent feedback control loops sharing a priority-based communication bus, as illustrated in Figure 6.1. The control loops consist of distributed sensors, controllers and actuators. We assign distinct priority to each feedback control loop and each loop utilizes the communication bus to send plant sampling data to a controller. At any time, only one control loop can access the communication bus and transmit data. Such a system model has practical applications. For example, in the automotive industry, electrical control units (ECUs) are implemented as the controllers for different vehicle plants. All sensor data are transmitted to ECUs through priority-based control area networks (CAN).

Each sensor in a control loop generates one recurring message chain, denoted as ξ_i,

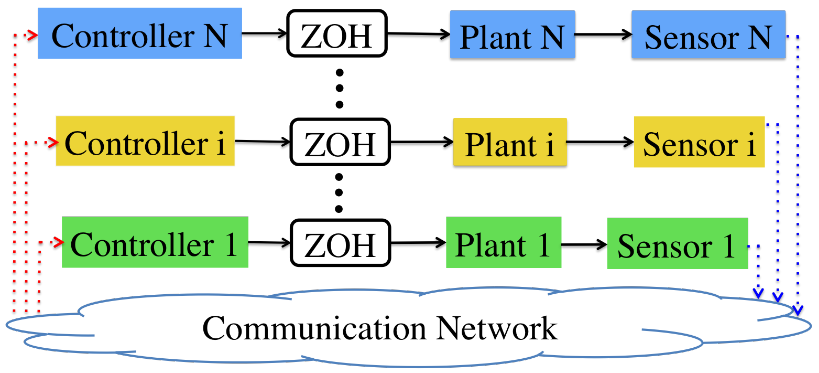


Figure 6.1: Networked system architecture

where $i = 1, ..., N$. Each message chain includes a sequence of sampling messages, denoted as $\{\tau_i[1], \tau_i[2], ..., \tau_i[k], ...\}$, where k is the message index of sensor i. The generating time of sensor message $\tau_i[k]$ is $\alpha_i[k]$. Each sensor message $\tau_i[k]$ contains the measurement of plant i. And $C_i[k]$ is the amount of time needed for sensor i to transmit $\tau_i[k]$ to controller i when no contention occur.

6.1.1 Problem Formulation

For the i-th control loop in Figure 6.1, we assume that the system equation has the form:

$$\dot{x}_i(t) = f_i(x_i(t), u_i(t)),\ y_i(t) = g_i(x_i(t)) \text{ for some functions } f_i \text{ and } g_i \qquad (6.1)$$

where $x_i(t)$ are the plant states, $y_i(t)$ is the plant output, and $u_i(t)$ is the control command. We formulate a continuous-time MPC problem with a dynamic priority assignment. The goal is to find an optimal priority assignment $\mathbf{P}^*(t) = (p_1^*(t), ..., p_N^*(t))$ and an optimal control command $\mathbf{u}^*(t) = (u_1^*(t), ...u_N^*(t))$ within the time interval $[t_0, t_f]$, such that the output of each plant can track a reference signal λ_i. That is to say, we want to steer the state $x_i(t)$ to the target state $\bar{x}_i$ corresponding to the reference signal λ_i that satisfies the following equations,

$$g_i(\bar{x}_i) = \lambda_i, \text{ and } f_i(\bar{x}_i(\lambda_i), \bar{u}_i) = 0. \qquad (6.2)$$

We denote the solution of the above equation by $(\bar{x}_i(\lambda_i), \bar{u}_i(\lambda_i))$. If multiple solutions exist, $(\bar{x}_i(\lambda_i), \bar{u}_i(\lambda_i))$ is selected such that $|x_i(t_0) - \bar{x}_i(\lambda_i)|^2$ is minimal, where $x_i(t_0)$ is the initial state of plant i. The notation indicates that dependence of the reference signal λ_i.

Given initial states $\mathbf{x}(t_0) = (x_1(t_0), ..., x_N(t_0))$, initial control $\mathbf{u}(t_0) = (u_1(t_0), ..., u_N(t_0))$ and message chain parameters $\mathbf{S} = (\alpha_i, C_i, T_i)[k]$ for all k, the contention resolving MPC is to find the value of decision variables $\mathbf{P}(t)$ and $\mathbf{u}(t)$ which solves the following optimization problem:

$$\min_{\mathbf{u}(t),\mathbf{P}(t)} \sum_{i=1}^{N} \frac{1}{2} \int_{t_0}^{t_f} \{ |x_i(t) - \bar{x}_i(\lambda_i)|^2_{Q_i} + |u_i(t) - \bar{u}_i(\lambda_i)|^2_{R_i} \} \mathrm{d}t + \rho |x_i(t_f) - \bar{x}_i(\lambda_i)|^2_{K_i}, \tag{6.3}$$

where $|x_i(t) - \bar{x}_i(\lambda_i)|^2_{Q_i} = [x_i(t) - \bar{x}_i(\lambda_i)]^T Q_i [x_i(t) - \bar{x}_i(\lambda_i)]$, $|u_i(t) - \bar{u}_i(\lambda_i)|^2_{R_i} = [u_i(t) - \bar{u}_i(\lambda_i)]^T R_i [u_i(t) - \bar{u}_i(\lambda_i)]$. Q_i, R_i and K_i are positive definite matrices and $\rho > 0$ is a constant. The optimization problem should satisfy the following constraints:

$$\dot{x}_i(t) = f_i(x_i(t), u_i(t)), y_i(t) = g_i(x_i(t)); \tag{6.4a}$$

$$u_i(t) = u_i(t_0), t \in [0, \gamma_i[1]),\ u_i(t) = u_i[k], t \in [\gamma_i[k], \gamma_i[k+1]); \tag{6.4b}$$

$$Z(\alpha_i[k+1]^-) = \mathbb{H}(\alpha_i[k+1]^-; Z(t_0), \mathbf{S}, \mathbf{P}(t_0 \sim \alpha_i[k+1])), \tag{6.4c}$$

$$\delta_i[k] = Z_{2N+i}(\alpha_i[k+1]^-),\ \gamma_i[k] = \alpha_i[k] + \delta_i[k],\ \text{for all } k \text{ such that } t_0 \leq \gamma_i[k] \leq t_f;$$

$$u_i(t) \in \mathbb{U}_i, \mathbf{P}(t) \in \mathcal{P}(\{1, ..., N\}) \subseteq \mathbb{N}^N. \tag{6.4d}$$

6.2 Timing Model for Non-preemptive Network

The work [19] presented a timing model for the CAN bus, which is a non-preemptive communication network. Here, we propose evolution rules for general non-preemptive real-time systems. We divide $[t_0, t_f]$ into sub-intervals $[t_w, t_{w+1}]$ such that tasks are only generated at t_w, but not at any other time instant within (t_w, t_{w+1}). Also the occupation is the shared resource can only be completed at t_w, not at any other time within (t_w, t_{w+1}). If the shared resource is not occupied at time t_w, i.e. $1 - \mathsf{sgn}(\mathrm{ID}(t_w)) = 1$, then it is the same case as preemptive scheduling where $t_{w+1} - t_w = \min\{d_1(t_w), ..., d_N(t_w), t_f - t_w\}$. If the shared resource is occupied by task $\mathrm{ID}(t_w)$ at time t_w, i.e. $\mathsf{sgn}(\mathrm{ID}(t_w)) = 1$, then we will require that $t_{w+1} - t_w \leq \min\{d_1(t_w), ..., d_N(t_w), t_f - t_w\}$, and in addition, $t_{w+1} - t_w$ should be less or equal to $r_{\mathrm{ID}}(t_w)$ so that the occupation completion time $t_w + r_{\mathrm{ID}}(t_w) \geq t_{w+1}$. Summarizing the above two cases, we have

$$\begin{aligned} t_{w+1} - t_w = {} & \mathsf{sgn}(\mathrm{ID}(t_w)) \min\{r_{\mathrm{ID}}(t_w), d_1(t_w), ..., d_N(t_w), t_f - t_w\} \\ & + (1 - \mathsf{sgn}(\mathrm{ID}(t_w))) \min\{d_1(t_w), ..., d_N(t_w), t_f - t_w\} \end{aligned}$$

for all w. The evolution rules of the timing state variables $Z(t)$ can also be derived through two steps.

At t_w: The changes of variables d_i, r_i and o_i at the times t_w are the same as (4.6). For the timing state variable ID, if $r_{\mathrm{ID}}(t_w^-) > 0$, which means the task $\mathrm{ID}(t_w^-)$ that was occupying the shared resource has not completed the occupation at time t_w, then $\mathrm{ID}(t_w)$ is the same as $\mathrm{ID}(t_w^-)$ because the system is non-preemptive. If $r_{\mathrm{ID}}(t_w^-) = 0$, which means the task $\mathrm{ID}(t_w^-)$ completed the occupation of the shared resource at time t_w, then $\mathrm{ID}(t_w)$ needs to switch to the task which is scheduled to access the shared resource. Combining these two cases, the evolution rule for the timing state ID can be expressed as

$$\mathrm{ID}(t_w) = \mathrm{ID}(t_w^-)\,\mathsf{sgn}(r_{\mathrm{ID}}(t_w^-)) + \left(\operatorname*{argmin}_{i \in \Lambda(t_w)} p_i(t_w)\right) \left(1 - \mathsf{sgn}(r_{\mathrm{ID}}(t_w^-))\right) \tag{6.5}$$

when $\Lambda(t_w) \neq \emptyset$, where $\Lambda(t_w) = \{i \in \{1, ..., N\} : r_i(t_w) = C_i(t_w)\}$ is the set of all indices of control systems which request access to the shared resource at time t_w. If the set $\Lambda(t_w)$ is empty, then $\mathrm{ID}(t_w) = 0$.

On (t_w, t_{w+1}): The state $\mathrm{ID}(t_w + \Delta t)$ remains unchanged because $t_{w+1} - t_w \leq r_{\mathrm{ID}}(t_w)$. If

$\mathrm{ID}(t_w) \neq 0$, the evolution rules for control system $\mathrm{ID}(t_w)$ are

$$d_{\mathrm{ID}}(t_w+\Delta t)=d_{\mathrm{ID}}(t_w)-\Delta t,\ r_{\mathrm{ID}}(t_w+\Delta t)=r_{\mathrm{ID}}(t_w)-\Delta t \text{ and } o_{\mathrm{ID}}(t_w+\Delta t)=o_{\mathrm{ID}}(t_w)+\Delta t \tag{6.6}$$

where $d_{\mathrm{ID}}(t)$ and $o_{\mathrm{ID}}(t)$ are defined analogously to $r_{\mathrm{ID}}(t)$. For a control system i where $i \neq \mathrm{ID}(t_w)$, the evolution rules are

$$d_i(t_w+\Delta t) = d_i(t_w) - \Delta t,\ r_i(t_w+\Delta t) = r_i(t_w) \text{ and } o_i(t_w+\Delta t) = o_i(t_w) + \mathsf{sgn}(r_i(t_w))\Delta t. \tag{6.7}$$

Combining all of the evolution rules in (6.5)$-$(6.7) leads to the timing model (4.2) of non-preemptive scheduling.

6.3 Simulation Results

The systems simulated are four scalar systems, although our assumptions allow nonlinear cases as well

$$\dot{x}_i(t) = a_i x_i(t) + u_i(t),\ y_i(t) = x_i(t),\ i = 1, 2, 3, 4$$

with parameters $(a_1, a_2, a_3, a_4) = \left(1, \frac{6}{5}, \frac{4}{3}, \frac{3}{2}\right)$. The initial conditions are $x_i(0) = 1$ and $u_i(0) = 0$ for each i. The control constraints are $u_i(t) \in [-3, 3]$ for $i = 1, ..., 4$. Notice that four plants are all stabilizable from the initial condition if no contention exists.

The time horizon $[t_0, t_f]$ for the simulation is from 0 to 6 seconds. The cost function is

$$V_i(x_i(0), 0) = \frac{1}{2}\int_0^6 \left\{x_i^2(t) + 0.0001u_i^2(t)\right\}\mathrm{d}t + x_i^2(6).$$

The reference signal is $\lambda_i(t) = 0$ for all i and $t \in [0, 6]$. The task parameters are

$$(C_1[k], C_2[k], C_3[k], C_4[k]) = (0.3, 0.3, 0.2, 0.2) \text{ and}$$
$$(T_1[k], T_2[k], T_3[k], T_4[k]) = (1, 1.25, 1.5, 2) \text{ in seconds.}$$

6.3.1 Preemptive Scheduling

For the preemptive scheduling, we compare the optimal priority assignment computed by our proposed algorithm with the priority assignments under RMS and EDF. The priorities assigned by EDF are $\mathbf{P}(t) = (1, 2, 3, 4)$ for all t, which are the same as the priorities assigned by RMS. The optimal priority assignments computed by our method are different from priorities assigned by RMS and EDF. The communication network occupation result scheduled by the optimal priority assignments is shown in Figure 6.2. Eight contentions occur in time window $[0, 6]$, represented by the crosses in the figure. At time 0, the first contention occurs among the four systems and the optimal priority assignment computed by our method is that $\mathbf{P}^*(t) = (4, 3, 2, 1)$ for $t \in [0, 1.25)$ seconds, i.e. system 4 has highest priority and system 1 has lowest priority. Therefore, system 4 gains access to the communication network at time 0 and all the other three systems are delayed. At time 1.25, the second contention occurred between systems 1 and 2. The contention-resolving MPC assigns system 1 with higher priority than system 2. System 1 gain access to the network and system 2 is delayed for 0.05 seconds.

6.3.2 Non-preemptive Scheduling

For non-preemptive scheduling, we compare the optimal priority assignment computed by our proposed algorithm with the priority assignments under RMS. The priorities assigned by RMS are $\mathbf{P}(t) = (1, 2, 3, 4)$ for all t. The optimal priority assignments computed by our method are different from priorities assigned by RMS. The communication network occupation result scheduled by the optimal priority assignments is shown in Figure 6.3. Five

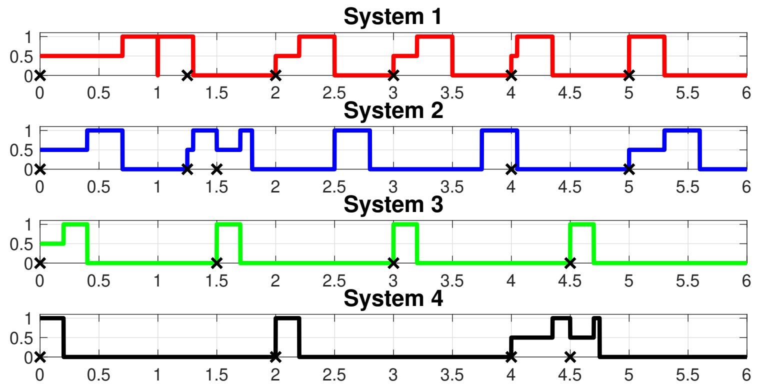


Figure 6.2: Communication network occupation of scheduling four scalar systems under preemptive scheduling. The occupation value 1 means the system is occupying the network, 0 means the system does not require access to the network, and 0.5 means the system access request is delayed by contention. Black crosses mark times when a contention occurs.

contentions occur in time window $[0, 6]$, represented by the crosses in the figure. Similar to the preemptive scheduling case, for $t \in [0, 1)$ seconds, system 4 has highest priority and system 1 has lowest priority. At time 2, the second contention occurs between system 1 and 4. The contention-resolving MPC assigns system 4 with higher priority than system 1. System 4 gains access to the network and system 1 is delayed for 0.2 seconds. At time 3, the third contention occurs between systems 1 and 3. The contention-resolving MPC assigned system 3 a higher priority than system 1. For the forth and fifth contentions at times 4 and 5 seconds, the contention-resolving MPC assigns system 1 a higher priority, which is different from the first three contentions.

6.3.3 Control Performance

The outputs of the four scalar systems under preemptive scheduling are presented in Figure 6.4. Systems 3 and 4 are unstable under the priorities assigned by RMS and EDF, because the third and fourth systems have lower priorities and longer delays. Under the optimal priority assignment, the four systems are all stable because the optimal priority assignment slightly sacrifices the control performance of system 1, by assigning system 1 the lowest priority and system 4 with the highest priority and system 3 with second highest priority

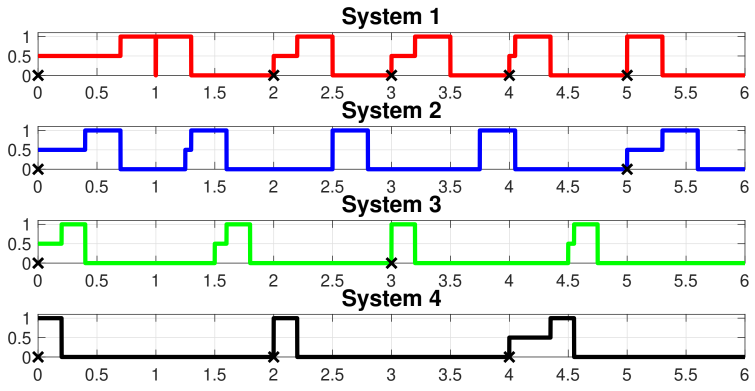


Figure 6.3: Communication network occupation of scheduling four scalar systems under non-preemptive scheduling discipline. Black crosses mark times when a contention occurs.

from 0 to 1 second. The outputs of the four scalar systems under non-preemptive scheduling are the same as Figure 6.4, except that $u_3(t) = -2.69$ during time interval $[0.4, 1.8]$ for the non-preemptive scheduling case, while $u_3(t) = -2.76$ during time interval $[0.4, 1.7]$ for the preemptive scheduling case.

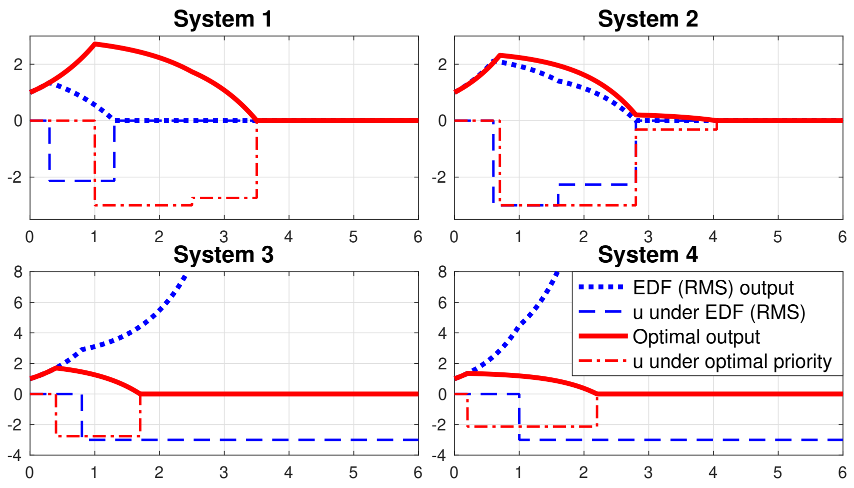


Figure 6.4: Outputs of four scalar systems. The red solid lines show the output under optimal priority assignment, and the blue solid lines show the outputs under EDF. The outputs under RMS are the same as EDF. The dashed lines show the control u_i computed by the MPC in each case.

CHAPTER 7
APPLICATION 2: SCHEDULING AND CONTROLLING VEHICLES AT A TRAFFIC INTERSECTION

In this chapter, we address the problem of optimally scheduling automated vehicles crossing an urban intersection by assigning vehicles with priorities. We formulate the intersection scheduling problem as a MIP problem which co-designs the priority and traveling speed for each vehicle. The co-design aims to minimize the vehicle waiting time at the intersection area, under a set of safety constraints. And the contention-resolving MPC method is applied to dynamically assign priorities and compute the optimal speed for each vehicle. Different from the optimal control design presented in Chapter 6, because of the special form of objective cost function in MPC formulation in the intersection scheduling application, we can design an analytical decentralized control law to control each vehicle to travel with an optimal speed given a specific priority assignment. The optimal priority assignment can be determined by searching the lowest cost path in the decision tree. The solution computed by contention-resolving MPC is proved to be global optimal given the condition of immediate access (or CIA) required in real-time scheduling. The effectiveness of the proposed method is verified through simulation and compared with the first-come-first-serve (or FCFS) and highest-speed-first (or HSF) scheduling strategies.

7.1 Intersection and Vehicle Model

In the traffic intersection scheduling application, we introduce a simplified traffic intersection setup, which illustrates the key features of our proposed contention-resolving MPC. Consider N automated vehicles traveling along a track that has a “figure-eight” shape with just one lane as shown in Figure 7.1. The figure eight consists of a shaded intersection area and two half cycles, which are the portions of the figure eight that are separated by

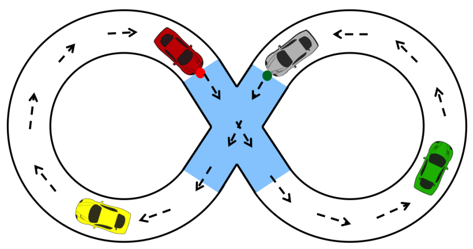


Figure 7.1: Example of a one lane intersection. Vehicles follow directions indicated by arrows.

the shaded intersection area. Vehicles travel in the direction marked by arrows, S is the arc length of a vehicle's path through the intersection, L is the length of a vehicle, and the two half cycles have the same arc length D. The motion of i-th vehicle for $i = 1, ..., N$ is modeled as

$$\dot{x}_i(t) = u_i(t), x_i(t_0) = x_i^0 \tag{7.1}$$

where $x_i(t)$ and $u_i(t)$ denote the position and speed of vehicle i respectively at each time t, and x_i^0 is the initial position where $x_i^0 = 0$ when the front end of a vehicle is at the intersection. We assume that vehicle speed satisfies

$$0 \leq u_i(t) \leq u_{i,\text{max}} \tag{7.2}$$

where $u_{i,\text{max}} > 0$ is the maximum speed limit. In our problem setup, each vehicle passes through the intersection with a universal constant speed u_{int} satisfying $u_{\text{int}} < u_{i,\text{max}}$ for all i.

7.2 Intersection Scheduling

Let $\alpha_i[k_i]$ denote the time when vehicle i gains access to the intersection after having traveled through k_i half cycles. For any index k_i, C is the amount of time from the instant when the front end of the vehicle arrives at the intersection to the time instant when the rear end of the vehicle i leaves the intersection, i.e. $C = \frac{S+L}{u_{\text{int}}}$. An intersection-exit time $\gamma_i[k_i]$ is the time instant when the rear end of vehicle i leaves the intersection for the k_i-th half cycle, satisfying $\gamma_i[k_i] = \alpha_i[k_i] + C$.

When there is no contention among vehicles, it is trivial that all the vehicles should travel with their maximal speed in order to minimize the total traveling time. The following equation needs to be satisfied if no contention occurs:

$$\alpha_i[k_i] = \gamma_i[k_i - 1] + T_i. \tag{7.3}$$

where T_i is the least amount of time for vehicle i to travel through the k_i-th half cycle, namely,

$$T_i = \frac{D - L}{u_{i,\max}}. \tag{7.4}$$

When a contention occurs to vehicle i while it is passing the intersection for the k_i-th time, equation (7.3) will not hold because it may be interrupted by other vehicles. Priorities are needed to determine which vehicle enters the intersection first. Each vehicle i is assigned a unique priority number $p_i(t)$, in which case contentions can be resolved by comparing the priorities p_i among all vehicles that are competing for the access to the intersection. We introduce the delay variable $\delta_j[k_j]$ and $\tilde{\alpha}_j[k_j]$ as the earliest time that vehicle j can arrive at the intersection, so that

$$\alpha_j[k_j]{=}\tilde{\alpha}_j[k_j] + \delta_j[k_j],\ \tilde{\alpha}_j[k_j + 1]{=}\alpha_j[k_j] + C + T_j. \tag{7.5}$$

Using the concept of the earliest arrival time, we can then mathematically define the crucial event of the intersection scheduling problem, which is a contention occurring between vehicles.

Definition 7.2.1 *If there exist indices i and j such* $[\tilde{\alpha}_i[k_i], \tilde{\alpha}_i[k_i]{+}C) \cap [\tilde{\alpha}_i[k_j], \tilde{\alpha}_i[k_i]{+}C) \neq$

$\emptyset$ *for some* i *and* j*, then a contention occurs between vehicle* i *and vehicle* j*.*

The physical meaning of this definition is that if the earliest possible intersection occupation time of vehicle i overlaps with the occupation time of vehicle j, then a contention occurs between vehicles i and j.

7.3 Timing Model for Intersection Scheduling

In task scheduling theory [48], if an on-going task (with process time C) can be interrupted by the arrival of other tasks, the scheduling type is called *preemptive*. Furthermore, if the interrupted task has to repeat its whole processing after the preemption, regardless how much it has been processed before the preemption. This scheduling type is called *preemptive-repeat*. The repeating process ends until there is an uninterrupted time window of length C.

We model the earliest arrival time $\tilde{\alpha}_i[k_i]$ as the resource requesting time of a task in scheduling theory and define the total process time of this task to be the intersection occupation time C. If there are no other vehicles which have the same earliest arrival time as $\tilde{\alpha}_i[k_i]$, this task can start processing. If there are no other tasks satisfying $[\tilde{\alpha}_i[k_i], \tilde{\alpha}_i[k_i] + C) \cap [\tilde{\alpha}_i[k_j], \tilde{\alpha}_i[k_i] + C) = \emptyset$, i.e. no contention occurs to vehicle i within its k_i-th half cycle, then there is no preemption and task i can finish the process. If there exists a task j such that $[\tilde{\alpha}_i[k_i], \tilde{\alpha}_i[k_i] + C) \cap [\tilde{\alpha}_i[k_j], \tilde{\alpha}_i[k_i] + C) \neq \emptyset$ and j is assigned a higher priority than i, then the process of task i is preempted by j and the whole processing of task i will be repeated at $\gamma_j[k_j]$. Task i may be preempted by the arrival of another higher prioritized task again until there is an uninterrupted time window of length C. This time window is the true intersection occupation time interval of vehicle i. The preemption mechanism guarantees that the intersection entrance order follows priority assignment and satisfies the CIA assumption. The task repeating mechanism guarantees the physical constraint that there is no interruption once a vehicle get the access to the intersection.

The evolution rules for the timing state $Z(t)$ of the intersection scheduling on $[t_0, t_f]$ are expressed by mathematical equations. We divide $[t_0, t_f]$ into sub-intervals $[t_w, t_{w+1}]$ such that tasks can only be generated at either t_w or t_{w+1}. Note here t_w is one of the $\tilde{\alpha}_i[k_i]$, i.e. $d_i(t_w^-) = 0$ for some i. At time t_w, the timing states $Z(t)$ of task i exhibit jumps,

$$d_i(t_w) = T_i + C,\ r_i(t_w) = C,\ o_i(t_w) = 0. \tag{7.6}$$

For task j which does not request processing at t_w, i.e. $d_j(t_w^-) \neq 0$, there are three different cases. The first case is that task j has completed processing, i.e. $r_j(t_w^-) = 0$. The second case is that task j is processing, i.e. $r_j(t_w^-) > 0$ and task j has higher priority than task i, i.e. $p_j(t_w^-) < p_i(t_w^-)$. In these two cases, the timing states of task j is not affected by task i. Hence, there exist no jumps for the value of timing states of vehicle j,

$$d_j(t_w) = d_j(t_w^-),\ r_j(t_w) = r_j(t_w^-),\ o_j(t_w) = o_j(t_w^-). \tag{7.7}$$

The third case is that task j is processing, i.e. $r_j(t_w^-) > 0$ and task j has lower priority than task i, i.e. $p_j(t_w^-) > p_i(t_w^-)$. In this case, task i preempts task j, so the timing states of task j are reset

$$d_j(t_w) = T_j + C,\ r_j(t_w) = C,\ o_j(t_w) = o_j(t_w^-) \tag{7.8}$$

where $d_j(t_w)$ and $r_j(t_w)$ are reset to be initial values as in (7.6).

Defining $S_{j,i} = \{t \in [t_0, t_f] : p_j(t) < p_i(t)\}$ following from Equation (7.7) and (7.8), we can express the evolution of the timing states $Z(t)$ for vehicle j from t_w^- to t_w as:

$$\begin{aligned}
d_j(t_w) &= \operatorname{sgn}\left((1 - \operatorname{sgn}(r_j(t_w^-))) + \mathbb{1}_{S_{j,i}^c}(t_w^-)\right) d_j(t_w^-) + \operatorname{sgn}(r_i(t_w^-))\mathbb{1}_{S_{j,i}}(t_w^-)\, T_j, \\
r_j(t_w) &= \operatorname{sgn}\left((1 - \operatorname{sgn}(r_j(t_w^-))) + \mathbb{1}_{S_{j,i}^c}(t_w^-)\right) r_j(t_w^-) \\
&\qquad + \operatorname{sgn}(r_i(t_w^-))\mathbb{1}_{S_{j,i}}(t_w^-)\,(C + T_j), \\
o_j(t_w) &= o_j(t_w^-),
\end{aligned} \tag{7.9}$$

where sgn is defined by $\mathrm{sgn}(p) = 1$ if $p \geq 0$ and $\mathrm{sgn}(p) = 0$ if $p < 0$ and $\mathbb{1}_S(t)$ is defined to be 1 if $t \in S$ and 0 if $t \notin S$ for any set S and S^c is the complement of set S. For any time $t_w + \Delta t \in (t_w, t_{w+1})$, the evolutions for all vehicles are

$$d_i(t_w + \Delta t) = d_i(t_w) - \Delta t, \tag{7.10}$$

$$r_i(t_w + \Delta t) = \max\left\{ 0, r_i(t_w) - \max\left\{ 0, \Delta t - \sum_{q \in HP_i(t_w)} r_q(t_w) \right\}\right\}, \tag{7.11}$$

$$o_i(t_w + \Delta t) = o_i(t_w) + \mathrm{sgn}(r_i(t_w))\min\left\{ \Delta t, r_i(t_w) + \sum_{q \in HP_i(t_w)} r_q(t_w) \right\}, \tag{7.12}$$

where $HP_i(t_w) = \{j \in \{1, \ldots, N\} : p_j(t_w) < p_i(t_w)\}$ is the set of all indices of vehicles which have higher priorities than vehicle i at time t_w.

Combining all of the evolution rules leads to the timing model of intersection scheduling, which computes the value of $Z(t)$ at time t, given the initial state variable $Z(t_0)$, the vehicle timing parameters (C, T_i) for all i and a specific priority assignment $\mathbf{P}(t_0 \sim t)$.

7.4 Contention-resolving Model Predictive Control

7.4.1 Formulation of MPC

We formulate a contention-resolving model predictive control problem to compute optimal priority assignments $\mathbf{P}^*(t) = (p_1^*(t), ..., p_N^*(t))$ and an optimal vehicle speed $\mathbf{u}^*(t) = (u_1^*(t), ...u_N^*(t))$ on a time interval $[t_0, t_f]$. The times t_0 and t_f are the starting and ending points of the MPC time horizon, respectively, and t_0 and t_f will move forward in time when the MPC is initiated. Given initial states $\mathbf{x}(t_0) = (x_1(t_0), ..., x_N(t_0))$ and initial controls $\mathbf{u}(t_0) = (u_1(t_0), ..., u_N(t_0))$, the co-design method is to find values for the optimal $\mathbf{P}^*(t)$

and $\mathbf{u}^*(t)$ by solving the optimization problem

$$\min_{\mathbf{P}(t),\mathbf{u}(t)} \sum_{i=1}^{N}\sum_{k_i=1}^{K_i} \int_{\gamma_i[k_i-1](\mathbf{P}(t),\mathbf{u}(t))}^{\alpha_i[k_i](\mathbf{P}(t),\mathbf{u}(t))} \frac{1}{2}\left[u_{i,\max} - u_i(t)\right]^2 \mathrm{d}t \tag{7.13}$$

$$\begin{aligned} \text{s.t. } & (7.1), (7.2) \text{ and } (7.5), \forall t \in [t_0, t_f], \\ & x_i(\gamma_i[k_i-1]) = S + L + (k_i-1)\cdot(S+D) + x_i^0, \\ & x_i(\alpha_i[k_i]) = k_i \cdot (S+D) + x_i^0 \text{ for all } i \text{ and } k_i \end{aligned} \tag{7.14}$$

where K_i is the largest index of the half cycle which vehicle i has traveled satisfying $\gamma_i[K_i] \leq t_f$. The notation $\gamma_i[k_i-1](\mathbf{P}(t), \mathbf{u}(t))$ and $\alpha_i[k_i](\mathbf{P}(t), \mathbf{u}(t))$ represent that these time instants are implicit functions of priority assignment $\mathbf{P}(t)$ and vehicle speed $\mathbf{u}(t)$. The cost function aims to increase the speed as much as possible to increase the intersection capacity. If a contention happens and a vehicle needs to slow down or stop, then the cost increases. The interval $[\alpha[k_i], \gamma_i[k_i]]$ is not included in the formulation because $u_i(t)$ is fixed to be u_{int}. A set of constraints (7.1), (7.2) and (7.5) need to be satisfied for all times $t \in [t_0, t_f]$. Equation (7.14) is the boundary condition by the definitions of $\gamma_i[k_i]$ and $\alpha_i[k_i]$.

7.4.2 Contention-Resolving MPC Algorithm

First, based the timing model introduced in Section 7.3, the contention time instants can be computed by the following Proposition,

Proposition 7.4.1 *Contention happens at time t if and only if the following conditions hold:*

$$\sum_{i=1}^{N} \operatorname{sgn}(r_i(t)) \geq 2, \sum_{i=1}^{N} \operatorname{sgn}(r_i(t^-)) \leq 1 \textit{ and } t = \tilde{\alpha}_i[k_i] \textit{ for some } i \textit{ and } k_i \tag{7.15}$$

where $r_i(t^-)$ is the limit from the left.

Proof. Based on Definition 4.1.2, if a vehicle i has not finished the current intersection occupation at t, then $r_i(t) > 0$ and $\operatorname{sgn}(r_i(t)) = 1$. Since $r_i(t)$ is always non-negative,

$\mathrm{sgn}(r_i(t)) \geq 0$ for all t. Therefore, $\sum_{i=1}^{N} \mathrm{sgn}(r_i(t)) \geq 2$ is equivalent to two or more vehicles wanting to access the intersection, which means a contention is occurring at time t. Since $\sum_{i=1}^{N} \mathrm{sgn}(r_i(t^-)) \leq 1$ means that no contention happens at time instants before t that are close to t, the result follows. □

Based on the contention times, we can construct a decision tree using the procedure in Section 5.2. branch cost $w_{l,j}$ is defined as

$$w_{l,j} = \sum_{i=1}^{N} w_{l,j}^{i} \tag{7.16}$$

where $w_{l,j}^{i}$ is the cost of vehicle i and it can be computed by solving the following optimization problem based on the significant moments calculated along a branch. Let $\underline{k}_i$ be the smallest index satisfying $\gamma_i[k_i] > t_l^c$ and $\overline{k}_i$ be the largest index satisfying $\gamma_i[k_i] \leq t_j^c$. If $\underline{k}_i \leq \overline{k}_i$, then

$$w_{l,j}^{i} = \sum_{k_i=\underline{k}_i}^{\overline{k}_i} \min_{u_i(t)} \int_{\gamma_i[k_i-1]}^{\alpha_i[k_i]} \frac{1}{2} \left[u_{i,\max} - u_i(t)\right]^2 \mathrm{d}t \tag{7.17}$$

$$\text{s.t. } (7.1), (7.2), (7.14) \text{ and given } \gamma_i[k_i-1], \alpha_i[k_i]$$

where $r_i[0]$ is defined to be t_0 for all i. The meaning of (7.17) is as follows. If the k_i-th intersection occupation of vehicle i is completed between the contention times t_l^c and t_j^c, i.e. $\gamma_i[k_i] \in (t_l^c, t_j^c]$, then the cost of the $(k_i{-}1)$-th half cycle, traveled between $[\gamma_i[k_i-1], \alpha_i[k_i]]$, is included in the branch cost $w_{l,j}$. If no intersection occupation of vehicle i is completed within $[t_l^c, t_j^c]$, i.e. $\underline{k}_i > \overline{k}_i$, then $w_{l,j} = 0$. This branch cost formulation ensures that all costs included in one branch are determined and will not be changed by the priority assignments at or after time t_j^c. The cost of the incompleted $(\overline{k}_i + 1)$-th half cycle will be included by the branches following the branch (l, j). Since if no contention occurs, all the vehicles can travel with their maximal speed limit. Therefore, we can replace the future cost $\hat{C}_h(v_l)$ in

(7.18) with

$$\hat{C}_h(v_l) = 0, \tag{7.18}$$

The optimal vehicle control design is embedded in the branch cost calculation. We need to solve the optimization problem (7.17) to obtain the optimal control law $u_i^*(t)$. In the next section we present an analytical solution for this optimal control problem.

7.4.3 Analytical Solution of the Optimal Vehicle Control

First, we need to show the constrained optimal control problem

$$\min_{u_i(t)} \int_{\gamma_i[k_i-1]}^{\alpha_i[k_i]} \frac{1}{2} \left[u_{i,\max} - u_i(t)\right]^2 \mathrm{d}t \ \text{ s.t. } (7.1), (7.2), (7.14) \tag{7.19}$$

has feasible solutions.

Lemma 7.4.1 *Given $\gamma_i[k_i - 1]$ and $\alpha_i[k_i]$, a feasible solution always exists for constraints (7.1), (7.2) and (7.14).*

Proof. If $\alpha_i[k_i] = \tilde{\alpha}_i[k_i]$, the unique feasible solution is that a vehicle travels with the maximal speed $u_{i,\max}$. If $\alpha_i[k_i] > \tilde{\alpha}_i[k_i]$, vehicle i can travel with a lower speed and arrives at the intersection later than $\tilde{\alpha}_i[k_i]$. And since $u_i(t)$ can be infinitely small, any time $\alpha_i[k_i]$ greater than $\tilde{\alpha}_i[k_i]$ is feasible. □

Then we can compute the optimal control law for vehicle i within the time window $[\gamma_i[k_i - 1], \alpha_i[k_i])$.

Theorem 7.4.1 *The optimal solution for (7.17) must satisfy*

$$u_i^*(t) = \frac{D - L}{\alpha_i[k_i] - \gamma_i[k_i - 1]} \ \textit{for all } k_i \geq 1. \tag{7.20}$$

Proof. We use a constrained optimization argument based on [113]. Using (7.17), the system dynamics (7.1), and the control/state constraints (7.2), it follows that for each vehicle

i, the Lagrangian function is

$$L_i(t,x_i,u_i) = \frac{1}{2}\left(u_{i,\max} - u_i\right)^2 + \lambda_i(t)\cdot u_i + \mu_{i,1}(u_i - u_{i,\max}) + \mu_{i,2}(-u_i) \qquad (7.21)$$

where $\lambda_i(t)$ is the co-state, and $\mu_{i,1}$ and $\mu_{i,2}$ are Lagrange multipliers. The Euler-Lagrange condition becomes

$$\dot{\lambda}_i = -\frac{\partial L_i}{\partial x_i} = 0.$$

Then $\lambda_i(t) = \nu_i$ for all t where ν_i is a constant. Based on [113], the necessary condition for optimality is

$$\frac{\partial L_i}{\partial u_i} = -u_{i,\max} + u_i(t) + \lambda_i(t) + \mu_{i,1} - \mu_{i,2} = 0.$$

Therefore, the optimal vehicle speed is a constant given by $u_i^*(t) = u_{i,\max} - \nu_i - \mu_{i,1} + \mu_{i,2}$. Then (7.1) gives

$$x_i(t) = x_i(\gamma_i[k_i-1]) + u_i^*(t)\left(t - \gamma_i[k_i-1]\right),\ t \in [\gamma_i[k_i-1], \alpha_i[k_i]]. \qquad (7.22)$$

Using the boundary condition $x_i(\alpha_i[k_i]) = x_i(\gamma_i[k_i-1]) + D - L$ from (7.14) and setting $t = \alpha_i[k_i]$ in (7.22), we then have the equality

$$x_i(\gamma_i[k_i-1]) + D - L = x_i(\gamma_i[k_i-1]) + u_i^*(t)\left(\alpha_i[k_i] - \gamma_i[k_i-1]\right)$$

which produces the formula for $u_i^*(t)$ where $k_i \geq 1$. □

Theorem 7.4.1 computes the analytical solution of the optimal speed for vehicle i, given the speed of vehicle j and the significant moments from the dynamic timing model. With this solution, we can directly compute the branch cost. The pseudo code to compute each branch cost is presented by Algorithm 5. The algorithm solves the optimal control design

Algorithm 5 Branch cost

```
1: Data: $t^c_{l-1}$, $t^c_l$, $M$, $\mathbf{P}_m$
2: Result: $\mathbf{u}^*(t)$, $w_{l,j}$
3: $u^*_i(t) = u_{i,\max}, i = 1, ..., N$ and $t \in [t^c_{l-1}, t^c_l]$;
4: for $k = 2 : M$ do
5:     $i$ is the index of vehicle such that $p_i = k$;
6:     $j$ is the index of vehicle such that $p_j = k - 1$;
7:     Compute $u^*_i(t)$ based on (7.20);
8: $w_{l,j} = \sum_{i=1}^{N} \sum_{k_i=\underline{k}_i}^{\overline{k}_i} \min_{u_i(t)} \int_{\gamma_i[k_i-1]}^{\alpha_i[k_i]} \frac{1}{2} \left[u_{i,\max} - u^*_i(t)\right]^2 \mathrm{d}t$
```

iteratively in the order of priorities.

7.5 Optimality of Contention-resolving MPC

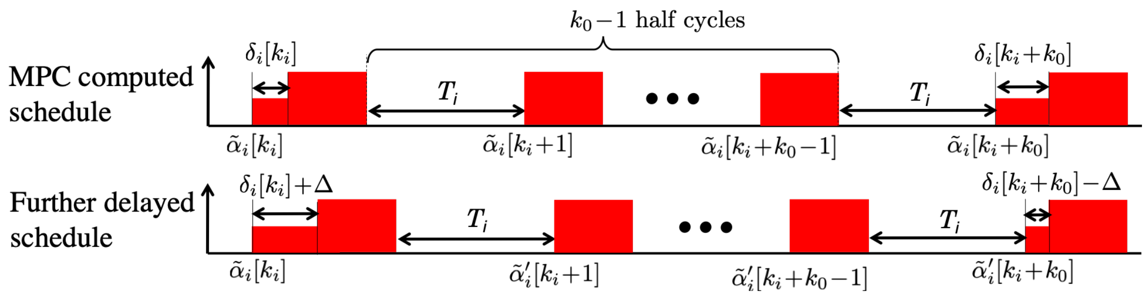
In this section, we prove that our algorithm finds the optimal solutions $\mathbf{P}^*(t)$ and $\mathbf{u}^*(t)$ which minimize (7.13) given the necessary CIA assumption.

Proposition 7.5.1 *The CIA assumption is a necessary condition for contention-resolving MPC algorithm to find the global optimal solution.*

Proof. We prove the contrapositive of this proposition: if the CIA assumption is relaxed, then there are situations where a better solution exists compared to the solution computed by contention-resolving MPC.

Assume that a contention occurs among vehicle i and some other vehicles at time $\tilde{\alpha}_i[k_i]$. Then after the computation of contention-solving MPC, the priority assignment is determined and the time delay of vehicle i, $\delta_i[k_i]$, can be computed by (4.2) given the priorities computed by MPC. And we assume after $k_0 - 1$ half cycles, the second contention occurs to vehicle i at time $\tilde{\alpha}_i[k_i + k_0]$ and the time delay of vehicle i is $\delta_i[k_i + k_0]$. For the $k_0 - 1$ half cycles between the time interval $[\tilde{\alpha}_i[k_i] + \delta_i[k_i] + C, \tilde{\alpha}_i[k_i + k_0]]$, vehicle i can travel with its maximal speed. Figure 7.2 shows an illustration of the considered case. The cost under such schedule can be computed as

$$J_{\mathrm{MPC}} = \left(u_{i,\max} - \frac{D-L}{T_i + \delta_i[k_i]}\right)^2 (T_i + \delta_i[k_i]) + \left(u_{i,\max} - \frac{D-L}{T_i + \delta_i[k_i + k_0]}\right)^2 (T_i + \delta_i[k_i + k_0]).$$



k_0-1 half cycles
$\delta_i[k_i]$
$\delta_i[k_i+k_0]$
MPC computed schedule
T_i
T_i
$\tilde{\alpha}_i[k_i]$
$\tilde{\alpha}_i[k_i+1]$
$\tilde{\alpha}_i[k_i+k_0-1]$
$\tilde{\alpha}_i[k_i+k_0]$
$\delta_i[k_i]+\Delta$
$\delta_i[k_i+k_0]-\Delta$
Further delayed schedule
T_i
T_i
$\tilde{\alpha}_i[k_i]$
$\tilde{\alpha}'_i[k_i+1]$
$\tilde{\alpha}'_i[k_i+k_0-1]$
$\tilde{\alpha}'_i[k_i+k_0]$

Figure 7.2: Illustration of further delaying the arrival of a vehicle.

Now we consider the case where the actual arrival time of vehicle i is further delayed by Δ at the first contention time, i.e., vehicle i arrives at the intersection at $\tilde{\alpha}_i[k_i]+\delta_i[k_i]+\Delta$. All the earliest arrival times after time $\tilde{\alpha}_i[k_i]+\delta_i[k_i]+\Delta$ are also delayed by Δ. And we assume the delay Δ does not affect the schedule of other vehicles and introduce more contentions. The cost under such schedule can be computed as

$$\begin{aligned} J(\Delta)=&\left(u_{i,\max}-\frac{D-L}{T_i+\delta_i[k_i]+\Delta}\right)^2(T_i+\delta_i[k_i]+\Delta)\\ &+\left(u_{i,\max}-\frac{D-L}{T_i+\delta_i[k_i+k_0]-\Delta}\right)^2(T_i+\delta_i[k_i+k_0]-\Delta)\end{aligned}$$

where $\Delta>0$. Then, if we take the difference of these two costs, we have

$$J_{\mathrm{MPC}}-J(\Delta)=\frac{(D-L)^2(2T_i+\delta_i[k_i]+\delta_i[k_i+k_0])(\delta_i[k_i+k_0]-\delta_i[k_i]-\Delta)\Delta}{(T_i+\delta_i[k_i])(T_i+\delta_i[k_i+k_0])(T_i+\delta_i[k_i]+\delta_i[k_i+k_0])T_i} \tag{7.23}$$

From (7.23) we can see, if $0<\Delta<\delta_i[k_i+k_0]$ and $\delta_i[k_i+k_0]-\delta_i[k_i]-\Delta>0$, we have $J_{\mathrm{MPC}}>J(\Delta)$. In other words, if the time delay $\delta_i[k_i+k_0]$ is greater than $\delta_i[k_i]$, then we can always find a Δ satisfying $0<\Delta<\delta_i[k_i+k_0]-\delta_i[k_i]$ and the cost $J(\Delta)$ will be less than J_{MPC}. In the case where $\delta_i[k_i+k_0]>\delta_i[k_i]$, contention-resolving MPC can only find a sub-optimal solution if Assumption 3.2.2 is relaxed. □

A numerical exmaple where the conditions $0<\Delta<\delta_i[k_i+k_0]$ and $\delta_i[k_i+k_0]-\delta_i[k_i]-\Delta>$

0 are satisfied will be presented in Section 7.6 to further justify the proof above.

Remark 6 *And since all the scheduling strategies and the optimal solution are based on Assumption 3.2.2, we will also provide a insight discussion about the cases where Assumption 3.2.2 is relaxed in Section 7.6.3.*

7.6 Case Studies

This section presents the simulation results obtained by the proposed method implemented in Matlab. We compare our proposed method with first come first serve scheduling (or FCFS) strategy and highest speed first (or HSF) strategy and demonstrate that our optimal scheduling method can provide a better solution than FCFS and HSF.

In the simulation, we consider 5 vehicles traveling on the figure eight track. The vehicle length L is 15 feet. We choose S and D such that $S + L = 0.75$ miles, $D - L = 6$ miles. Let $t_f = 25$ minutes and the speed limit of the first vehicle be $u_1^{max} = 1.25$ miles per minute (75 mph). Let the speed limit of the other four vehicles be the same, $u_2^{max} = u_3^{max} = u_4^{max} = u_5^{max} = 1$ mile per minute (60 mph). The intersection speed $u_{\text{int}} = 0.75$ mile per minute (45 mph).

7.6.1 Contention-resolving MPC VS FCFS

For the first studied case, we set the initial positions to be $x_1^0 = 4.25$, $x_2^0 = 4.85$, $x_3^0 = 3.25$, $x_4^0 = 0.75$ and $x_5^0 = 0$ miles. In this setup, vehicle 5 arrives at the intersection at time 0. The earliest arrival times of vehicles $1, 2, 3$ and 4 are 2, 1.9, 3.5 and 6 minutes, respectively. The algorithm only takes 0.12 seconds to find the solution for this example. The total cost is 0.3758.

The intersection occupation result is shown in Figure 7.3. The vehicle speed design is shown in Figure 7.4. Four contentions occur in the time interval $[0, 25]$. The first contention occurs at time 2. Although the earliest arrival time of vehicle 2 is smaller than vehicle 1, the priority assignment computed by our method gives vehicle 1 higher priority (which is different from the priority assigned by the FCFS strategy) because it has a higher speed limit than vehicle 2. At time 3.5, the second contention occurs between vehicles 2 and 3, which creates the possibility that vehicle 2 can be delayed twice. Our method can solve this problem and determines that the optimal priority assignment is that vehicle 2 crosses the intersection before vehicle 3. Two more contentions occur at 14 and 20 minutes, and vehicle 1 is assigned a higher priority to resolve these two contentions.

For comparison, if we always assign higher priority to the vehicle which arrives at the intersection first, i.e. following the FCFS strategy, and use regular MPC to design the vehicle speeds, then the cost would be 0.5114, which is 36% higher than our solution. While the example is simple, the simulation results show that our method performs better than the FCFS. Notice that in this specific simulation case, vehicle 1 always has highest priority, which agrees with the HSF scheduling strategy because vehicle 1 has the highest traveling speed. However, HSF is not always the optimal solution, which will be shown in the next subsection.

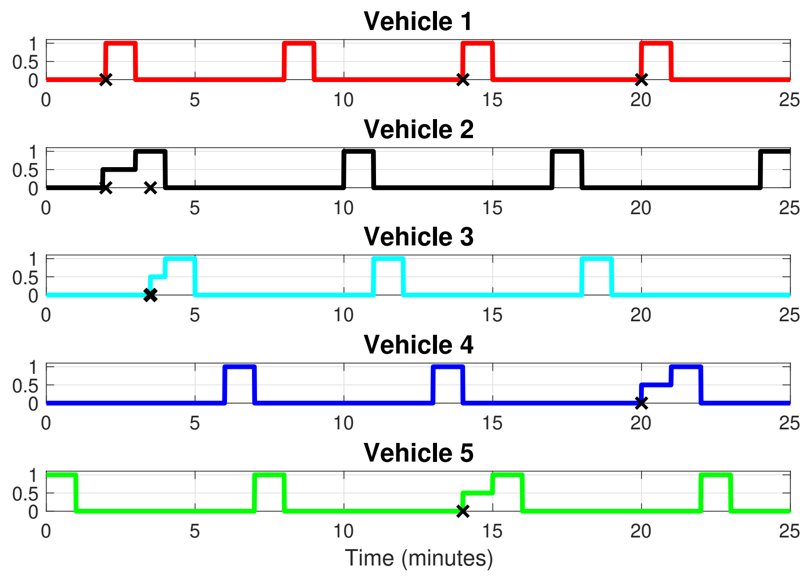


Figure 7.3: Intersection occupation for scheduling five vehicles. The y axis value 1 means that the vehicle is occupying the intersection, 0 means that the vehicle has not arrived at the intersection, and 0.5 means that the earliest arrival of a vehicle is delayed by a contention. The black crosses mark the time instant when a contention occurs.

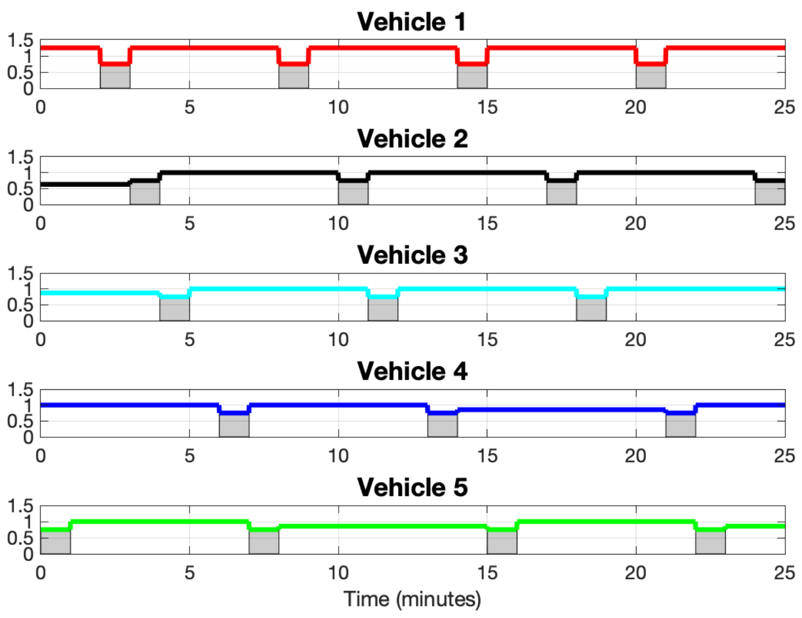


Figure 7.4: Optimal vehicle speed of scheduling five vehicles. The shaded areas mark the time interval when a vehicle is crossing the intersection.

7.6.2 Contention-resolving MPC VS HSF

In the previous simulated case, the optimal priority assignment is the same as the priority assignment under HFS. However, if we change the initial condition, the HFS will not be optimal and we will show that our optimal priority assignment can perform significantly better than HSF strategy.

Let the initial condition to be $x_1^0 = 1.75$, $x_2^0 = 3.75$, $x_3^0 = 4.75$, $x_4^0 = 5.75$ and $x_5^0 = 0$ miles. In this setup, vehicle 5 arrives at the intersection at time 0. The earliest arrival times of vehicles $1, 2, 3$ and 4 are 4, 3, 2 and 1 minutes, respectively. The contention-resolving MPC algorithm only takes 0.09 seconds to find the solution for this example. The total cost is 0.4662.

The decision tree constructed by contention-resolving MPC is shown in Figure 7.5. Six contentions occur in the time interval $[0, 25]$. Therefore, the total number of leaves in the fully constructed decision tree is $2^6 = 64$. And we can see that using the A-star inspired searching algorithm, contention-resolving MPC only needs to generate 24 leaves to find the optimal solution.

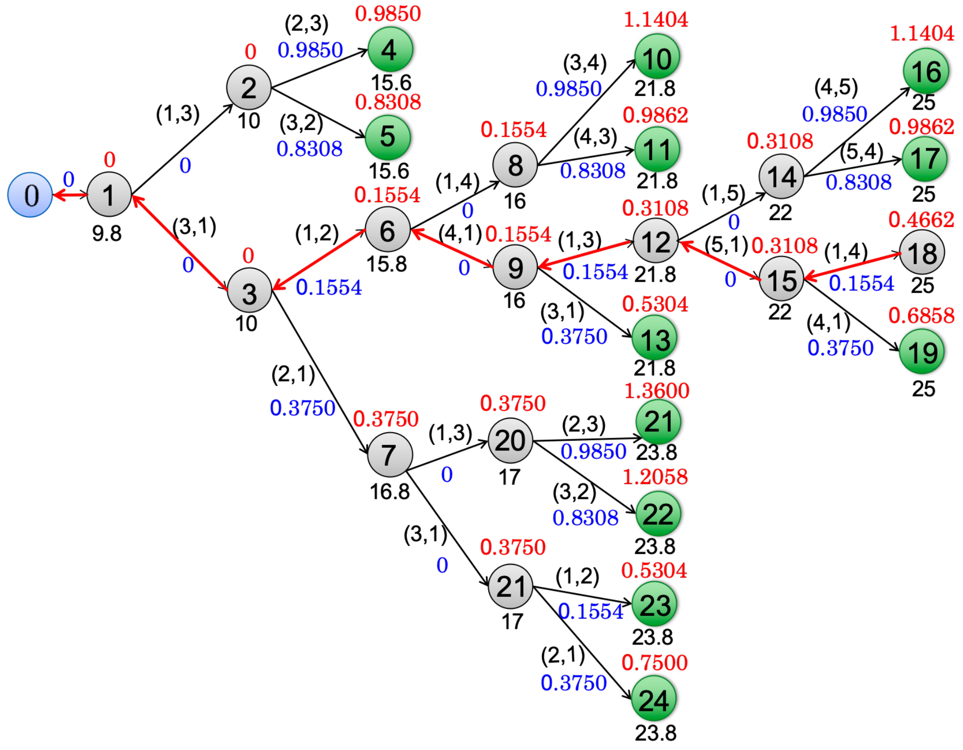


Figure 7.5: Decision tree constructed by contention-resolving MPC. Blue numbers represent branch costs $w_{l,j}$. The black numbers under leaves represent contention time instant t_l^c. The red numbers above leaves represent estimated total costs $\hat{C}_f(v)$. The red arrows represent the path with lowest cost.

The intersection occupation result where the vehicles are scheduled under the optimal priority assignment is shown in Figure 7.6 and the vehicle speed design is shown in Figure 7.7. The first contention occurs between vehicles 1 and 3 at time 9.8, marked by the black cross. The second contention occurs between vehicles 1 and 2 right after the first contention at time 10, marked by the purple cross. As we can see from Figure 7.6, although the maximal speed limit of vehicle 1 is larger than vehicle 3, the priority assignment computed by our method assigns vehicle 3 with higher priority (represented by the branch between leaf 1 and 3 in Figure 7.5). The intuitive reason for such solution is that vehicle 3 only needs an extra 0.2 minutes to pass the intersection when the first contention occurs, while vehicle 1 needs 1 minute to pass the intersection. Therefore, assigning vehicle 3 with higher priority leads to a smaller cost. Under such priority assignment, the optimal speed $u_1^*(t)$ of vehicle 1 is reduced to 1.2 miles per minute, which is shown within the time interval $[5, 10]$ in Figure 7.7. Vehicle 1 only needs to slightly sacrifice its maximal speed to resolve this contention while vehicle 3 travels with its maximal speed. Then at the next contention time 10, the second contention occurs between vehicles 1 and 2, which creates the possibility that vehicle 1 can be delayed twice. Our method can resolve this issue and determine that the optimal priority assignment is that vehicle 1 has higher priority than vehicle 2. Seen from 7.7, under this priority assignment, the optimal speed of vehicle 2 is $\frac{6}{7}$ miles per minute within time interval $[4, 11]$ minutes. Similar situations occur twice at 15.8 and 21.8 minutes, where vehicle 1 is assigned with a lower priority than the first contended vehicle and a higher priority than the second contended vehicle.

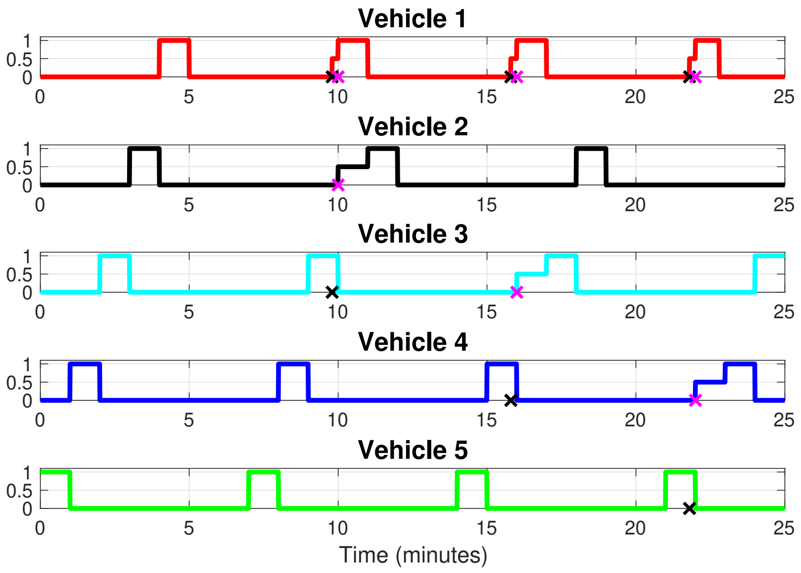


Figure 7.6: Intersection occupation for scheduling five vehicles. The y axis value 1 means that the vehicle is occupying the intersection, 0 means that the vehicle has not arrived at the intersection, and 0.5 means that the earliest arrival of a vehicle is delayed by a contention. The black crosses mark the time instant when a contention occurs.

For comparison, if we always assign higher priority to the vehicle with the highest speed limit, i.e. vehicle 1 always has highest priority following the HSF strategy, and use

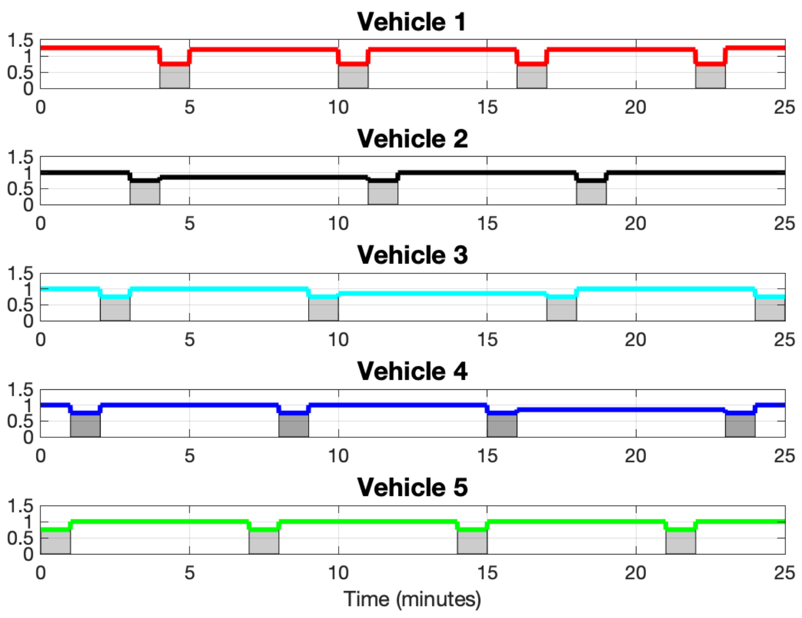


Figure 7.7: Optimal vehicle speed of scheduling five vehicles. The shaded areas mark the time interval when a vehicle is crossing the intersection.

regular MPC to design the vehicle speeds, then the cost would be 1.4235, which is 205% higher than our solution.

7.6.3 Numerical Results Without the CIA Assumption

In this subsection, we will discuss cases where the CIA assumption is relaxed and show a numerical example where the contention-resolving MPC can only find a sub-optimal solution.

Consider 2 vehicles traveling on the figure eight track. Let the speed limit of the first vehicle be $u_1^{max} = 1.5$ miles per minute (90 mph). Let the speed limit of the second vehicle be $u_2^{max} = 1$ mile per minute (60 mph). The intersection speed is $u_{\text{int}} = 0.75$ mile per minute (45 mph). The initial positions are $x_1^0 = 4.8$ and $x_2^0 = 0$ miles. All the other parameters are the same as previous simulations.

Figure 7.8 shows the intersection occupation results of vehicle 1 and vehicle 2 with and without CIA assumption. With the CIA assumption, we can see two contentions occur

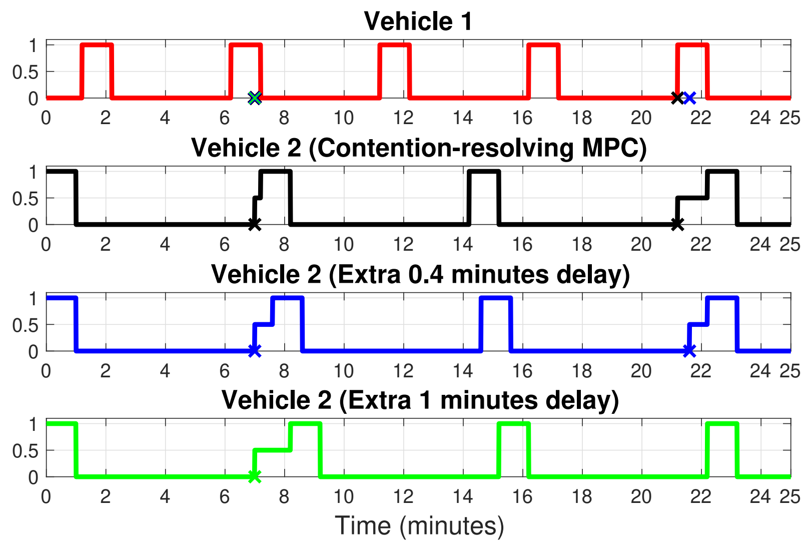


Figure 7.8: Intersection occupation for scheduling two vehicles. The First two sub-figures show the intersection occupation results of vehicle 1 and vehicle 2 computed by contention-resolving MPC under Assumption 3.2.2. The third sub-figure (from top to bottom) shows the intersection occupation time of vehicle 2 with an extra 0.4 minutes time delay. The forth sub-figure shows the intersection occupation time of vehicle 2 with an extra 1 minutes time delay.

between vehicle 1 and vehicle 2 within the time horizon $[0, 25]$. The first contention occurs at 7 minutes. To resolve this contention, vehicle 1 is assigned with higher priority and it leaves the intersection at 7.2 minutes. Therefore, vehicle 2 can only enter the intersection at or after time 7.2 minutes. Under Assumption 3.2.2, the speed of vehicle 2 is decreased to be $\frac{6}{6.2}$ miles per minute for the time window $[1, 7.2]$ such that vehicle 2 arrives at and enter the intersection at time 7.2 minutes. And it leaves the intersection at 8.2 minutes. Then vehicle 2 travels with its maximal speed and arrives at the intersection at time 14.2 without any contention with vehicle 1. The second contention between vehicles 1 and 2 occurs at time 21.2. Vehicle 1 is also assigned with higher priority than vehicle 2 to resolve this contention. Therefore, vehicle 1 enters the intersection at at time 21.2 and leaves the intersection at time 22.2. Vehicle 2 travels with a reduced speed $\frac{6}{7}$ miles per minute and arrives at the intersection at time 22.2 and leaves the intersection at time 23.2. The total

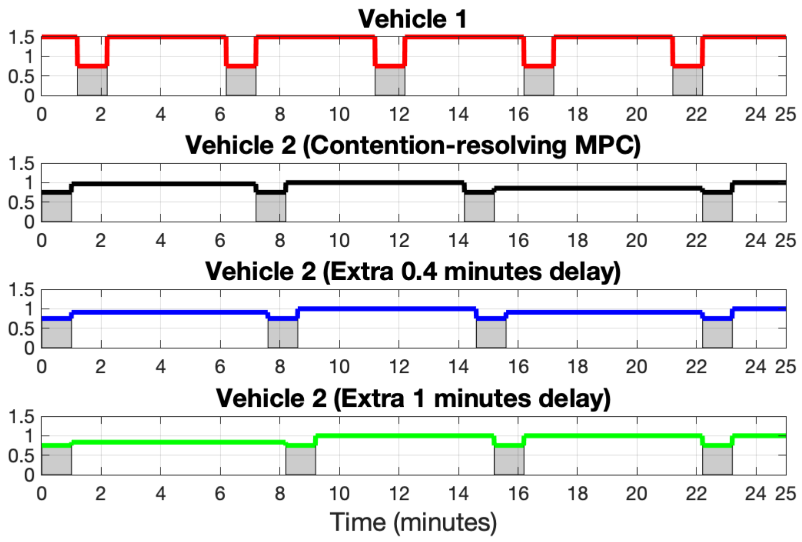


Figure 7.9: vehicle speed design of scheduling two vehicles.

cost under such scheduling and control co-design solution is

$$J_1 = \left(1 - \frac{6}{6.2}\right)^2 \cdot 6.2 + \left(1 - \frac{6}{7}\right)^2 \cdot 7 = 0.1493.$$

Case 1: if we relax the CIA assumption and let vehicle 2 be delayed for 0.6 minutes at the first contention (as shown by the third sub-figure in Figure 7.8), then vehicle 2 travels with a reduced speed $\frac{6}{6.6}$ miles per minute and arrives at the intersection at time 7.6, which is 0.4 minute after the time instant when vehicle 1 leaves the intersection. Since vehicle 2 is delayed for an extra 0.4 minutes compared to the case with the CIA assumption, the next earliest possible arrival time of vehicle 2 after time 7.6 is 14.6, which is also delayed for an extra 0.4 minutes compared to the case under the CIA assumption, shown by the sub-figure in the middle of Figure 7.8. At the earliest arrival time 14.6, vehicle 2 does not contend with vehicle 2 so it can travel with its maximal speed within time $[8.6, 14.6]$ and enter the intersection at time 14.6. The next earliest arrival time of vehicle 2 after time 14.6

is 22.6 where second contention occurs. Vehicle 1 is also assigned with higher priority than vehicle 2 to resolve this contention. Therefore, vehicle 1 enters the intersection at at time 21.2 and leaves the intersection at time 22.2. Vehicle 2 travels with a reduced speed $\frac{6}{6.6}$ miles per minute and arrives at the intersection at time 22.2 and leaves the intersection at time 23.2. The total cost under this schedule and vehicle speed control is

$$J_2 = \left(1 - \frac{6}{6.6}\right)^2 \cdot 6.6 + \left(1 - \frac{6}{6.6}\right)^2 \cdot 6.6 = 0.1091,$$

which is less than J_1. This numerical result show that if the CIA assumption is relaxed, contention-resolving MPC cannot find the global optimal solution. A solution which leads to smaller cost than the solution computed by contention-resolving MPC may exist.

Case 2: if we relax the CIA assumption and further delay the arrival of vehicle 2 at the first contention, as shown by the bottom sub-figure in Figure 7.8, then vehicle 2 travels with a reduced speed $\frac{6}{7.2}$ miles per minute and arrives at the intersection at time 8.2, which is 1 minute after the time instant when vehicle 1 leaves the intersection. Then the next two earliest possible arrival times of vehicle 2 after time 7.6 are 15.2 and 23.2 minutes. At both arrival times, vehicle 2 does not contend with vehicle 1 which reduces the number of contentions. It can travel with its maximal speed within time $[9.2, 25]$. The total cost under this schedule is

$$J_3 = \left(1 - \frac{6}{7.2}\right)^2 \cdot 7.2 = 0.2,$$

which is greater than J_1 and J_2. This example shows that further delaying the arrival of a vehicle not only affect the cost of the co-design optimization solution, it can also change the number of contentions in the future, which changes the structure of the decision tree. From (7.23), if $\Delta \geq \delta_i[k_i + k_0]$, i.e., the extra time delay is large enough such that the second contention will not occur, then we have $\delta_i[k_i + k_0] - \delta_i[k_i] - \Delta < 0$, which leads to $J_{\text{MPC}} < J(\Delta)$. Therefore, further delaying the arrival of a vehicle to avoid the second contention cannot reduce the cost computed by contention-resolving MPC.

CHAPTER 8
APPLICATION 3: HUMAN AND MULTI-ROBOT COLLABORATION SYSTEM

In this chapter, we analyze a human and multi-robot collaboration system and apply the contention-resolving MPC to optimally schedule the human attention when a human operator receives collaboration requests from multiple robots at the same time. The human attention scheduling problem is formulated as a binary optimization problem which aims to maximize the overall performance among all the robots, under the constraint that a human has limited attention capacity. We first present the optimal schedule for the human to determine when to collaborate with a robot if there is no contention occurring among robots' collaboration requests. We rigorously show that for the case where no contention occurs among robots, the optimal schedule for a robot to maximize its performance is to start the collaboration with a human operator once the collaboration request is generated. For the case where contentions occur, the optimal schedule for a robot i is to start the collaboration right after the time instant when all contended robots that are scheduled to collaborate before robot i complete their collaborations. This property ensures the *Condition of Immediate Access* or *CIA*, which we have shown to be a necessary condition for contention-resolving MPC to find the optimal solution in Chapter 7. We also developed discrete-time contention-resolving MPC to dynamically schedule the human attention and determine which robot the human should collaborate with first. The optimal schedule can then be determined using a sampling based approach. The effectiveness of our method is verified through simulations and compared with the highest trust first (or HTF) scheduling strategy.

8.1 Robot Performance and Human-to-robot Trust Models

In this chapter, we consider one human operator collaborating with N robots. In this human and multi-robot collaboration system setup, the human is considered as an expert so if the human is collaborating with a robot, then the human can help the robot to improve its performance on task execution. For a robot i where $i = 1, ..., N$, we first introduce a dynamic model describing its performance, which is given by the following model

$$\begin{aligned} P_i(k) = (1 - u_i(k)) \left[(1 - k_{i,R})P_i(k-1) + k_{i,R}P_{i,\min}\right] \\ + u_i(k) \left[(1 - k_{i,H})P_i(k-1) + k_{i,H}P_{i,\max}\right] \end{aligned} \tag{8.1}$$

where k denotes the discrete time step and i denotes the index of a robot. The parameters $P_{i,\min}$, $P_{i,\max}$ are the minimal and maximal values of the performance value of robot i. The control variable $u_i(k)$ only has two values, 0 or 1. If $u_i(k) = 1$, then the robot is in collaborative mode with the human operator. If $u_i(k) = 0$, then the robot is in autonomous mode without the collaboration with the human operator. The parameters $k_{i,R}$ and $k_{i,H}$ are coefficients for autonomous and collaborative mode, respectively, satisfying $0 < k_{i,H} < k_{i,R} < 1$. The robot performance model (8.1) guarantees that $P_i(k)$ is bounded between $[P_{i,\min}, P_{i,\max}]$, given that their initial performance value $P_i(k_0)$ is within $[P_{i,\min}, P_{i,\max}]$. The performance value $P_i(k)$ will decrease under the autonomous mode because it is a convex combination of $P_i(k–1)$ and $P_{i,\min}$. And $P_i(k)$ will increase under the collaborative mode because it is a convex combination of $P_i(k-1)$ and $P_{i,\max}$.

Then based on the robot's performance model, we introduce the concept of human-to-robot trust. We utilize the human-to-robot trust model in [62] to quantify how good the collaboration experience is for the human operator. The trust is modeled as

$$\mathcal{T}_i(k) = A_i\mathcal{T}_i(k-1) + B_iP_i(k) - C_iP_i(k-1) + D_iF_i(k) - E_iF_i(k-1) \tag{8.2}$$

where the function $\mathcal{T}_i(k)$ represents the trust level from the human operator to robot i at time k. It is determined by the previous trust level $\mathcal{T}_i(k-1)$, the robot performance measure $P_i(k)$ and $P_i(k-1)$, and the robot fault rate $F_i(k)$ and $F_i(k-1)$, which are random variables following the standard normal distribution. The parameters A_i, B_i, C_i, D_i and E_i are constant coefficients whose values depend on the human operator, robot i and the corresponding collaborative task. The trust level should be within a proper range so the human does not "under-trust" or "over-trust" a robot, i.e.

$$\mathcal{T}_{i,\min} \leq \mathcal{T}_i(k) \leq \mathcal{T}_{i,\max} \text{ for all } k \in [k_0, k_f] \tag{8.3}$$

where $\mathcal{T}_{i,\min} > 0$ and $\mathcal{T}_{i,\max} > 0$ are the lower and upper bounds of the trust level for robot i, respectively and the times k_0 and k_f are the starting and ending time of the scheduling time horizon. This will be one constraint which we consider in the problem formulation.

8.2 Human Attention Scheduling

For all N robots, each one needs to execute a sequence of tasks $\Gamma_i = \{\tau_{i,1}, \tau_{i,2}, ..., \tau_{i,n_i}, ...\}$ where i is the index of a robot and n_i is the index the task. We assume the tasks are all periodic for each robot and use the notation T_i to denote the period. Let $\alpha_i(n_i)$ denote the time that robot i starts to execute the n_ith task, which is also the time when robot i requests to collaborate with the human. For any index n_i, $C_i(n_i)$ is the collaboration time that robot i requires to collaborate with the human operator within the time window $[\alpha_i(n_i), \alpha_i(n_i)+T_i)$ satisfying $1 \leq C_i(n_i) < T_i$ for all i and n_i. And at each time $\alpha_i(n_i)$, the performance value $P_i(\alpha_i(n_i))$ of robot i is reinitialized to be $P_i^0(n_i) \in [P_{i,\min}, P_{i,\max}]$, because each task in task sequence Γ_i may be very different from each other. A collaboration completion time $\gamma_i(n_i)$ is the time step when robot i finishes collaborating with the human operator. Since the system is modeled in discrete time, the parameters α_i, C_i, T_i and γ_i are all integers.

Remark 7 *In our problem set up, it is not required that the human and robot collaboration* *needs to start at the moment $\alpha_i(n_i)$, but we will show in Section 8.2.2 that the collaboration starting at the moment $\alpha_i(n_i)$ is the optimal solution to maximize robot performance value if the human attention limitation is ignored, i.e., the contention constraint (8.4) is relaxed.*

Assumption 8.2.1 *For each task τ_{i,n_i}, once the collaboration starts between the human and robot i at time k, it will only ends at time $k + C_i(n_i)$.*

This assumption indicates that the collaboration between the human operator and a robot cannot be interrupted, which prevents frequent switches among the collaboration with different robots to save human's energy.

A contention time is defined to be a time when two or more robots request to collaborate with the human operator at the same time. Due to the human attention capacity limitation, we make the following assumption when a contention occurs:

Assumption 8.2.2 *At any given time, at most one robot can be in collaborative mode with the human operator and all the other robots are in autonomous mode, i.e.,*

$$\sum_{i=1}^{N} u_i(k) \leq 1 \textit{ for all } k. \tag{8.4}$$

Because of contentions, we introduce the delay variable $\delta_i(n_i) \geq 0$ so that

$$\gamma_i(n_i) = \alpha_i(n_i) + \delta_i(n_i) + C_i(n_i). \tag{8.5}$$

8.2.1 Formulation of Model Predictive Control

We formulate a human attention allocation problem to compute optimal scheduling $\mathbf{u}^*(k) = (u_1^*(k), ..., u_N^*(k))$ on a time interval $[k_0, k_f]$.

Given initial human-robot trust level $(\mathcal{T}_1(k_0), ..., \mathcal{T}_i(k_0), ..., \mathcal{T}_N(k_0))$ and initial robot performance value $(P_1(n_1), ..., P_i(n_i), ..., P_N(n_N))$ for all i and n_i, the optimal scheduling problem is to find values for the optimal $\mathbf{u}^*(k)$ by solving the optimization problem

$$\begin{aligned}
&\min_{\mathbf{u}(k)} \sum_{i=1}^{N} \sum_{k=k_0}^{k_f} [P_{i,\max} - P_i(k)] \text{ subject to } (8.1), (8.2), (8.3), (8.4), \\
&u_i(k) = 0, k \in [\alpha_i(n_i), \alpha_i(n_i) + \delta_i(n_i)(\mathbf{u}(k)) - 1], \\
&u_i(k) = 1, k \in [\alpha_i(n_i) + \delta_i(n_i)(\mathbf{u}(k)), \gamma_i(n_i)(\mathbf{u}(k))] \text{ and} \\
&u_i(k) = 0, k \in [\gamma_i(n_i)(\mathbf{u}(k)) + 1, \alpha_i(n_i + 1) - 1] \\
&\text{for all } n_i \text{ such that } k_0 \leq \alpha_i(n_i) \text{ and } \alpha_i(n_i + 1) \leq k_f.
\end{aligned} \tag{8.6}$$

where the notations $\delta_i(n_i)(\mathbf{u}(k))$ and $\gamma_i(n_i)(\mathbf{u}(k))$ represent that these time instants are implicit functions of $\mathbf{u}(k)$. The cost function aims to increase the robot performance as much as possible to reach the performance upper bounds. Equations (8.1) and (8.2) are system dynamic equations. Constraint (8.3) aims to maintain the trust level within the range. Equations (8.4) is the contention constraint where $u_i(k)$'s are coupled. Since $\mathbf{u}(k)$ is a vector of binary integers at each time k, the problem is binary optimization problem. It is a non-convex optimization problem that is difficult to solve.

8.2.2 Optimal Solution Without Considering Contention

We will first relax the trust level constraint (8.3) and the human attention limitation constraint (8.4) in the problem formulation (8.6) to find the optimal solution $\mathbf{u}(k)$ to maximize the overall robot performance value among the time horizon $[k_0, k_f]$. After relaxing the two constraints, the problem (8.6) can be decoupled and is equivalent to

$$\sum_{i=1}^{N}\sum_{n_i=1}^{N_i}\max_{\delta_i(n_i)}\sum_{k=\alpha_i(n_i)}^{\alpha_i(n_i+1)-1} P_i(k) \tag{8.7}$$

subject to (8.1) with $P_i^0(n_i)$ and

$$u_i(k)=\begin{cases}0, & k\in[\alpha_i(n_i),\alpha_i(n_i)+\delta_i(n_i)-1],\\ 1, & k\in[\alpha_i(n_i)+\delta_i(n_i),\alpha_i(n_i)+\delta_i(n_i)+C_i-1],\\ 0, & k\in[\alpha_i(n_i)+\delta_i(n_i)+C_i,\alpha_i(n_i+1)-1],\end{cases}$$

where N_i is the largest index of tasks in Γ_i satisfying $\alpha_i(N_i) < k_f$

Theorem 8.2.1 *(CIA condition) The optimal solution for problem (8.7) is* $\delta_i(n_i) = 0$ *for all* $1 \le n_i \le N_i$.

Proof. We first define the cost for robot i within the time window $[\alpha_i(n_i), \alpha_i(n_i+1)-1]$ to be

$$J_{i,n_i}(\delta_i(n_i)) = \sum_{k=\alpha_i(n_i)}^{\alpha_i(n_i+1)-1} P_i(k).$$

Then we will show that the derivative of $J_{i,n_i}(\delta_i(n_i))$ is less than 0, so $J_{i,n_i}(\delta_i(n_i))$ is strictly decreasing as $\delta_i(n_i)$ increases. For simplification, we will use P_i^0 to represent $P_i^0(n_i)$ in the following part of this proof.

During the time $k \in [\alpha_i(n_i), \alpha_i(n_i) + \delta_i(n_i)-1]$, we have $u_i(k) = 0$. The dynamic of robot i's performance value according to (8.1) is

$$P_i(k) = (1 - k_{i,R})P_i(k-1) + k_{i,R}P_{i,\min} \text{ with } u_i(k) = 0.$$

Then for any $k \in [\alpha_i(n_i), \alpha_i(n_i) + \delta_i(n_i)]$,

$$\begin{aligned} P_i(k) &= (1-k_{i,R})^{k-\alpha_i(n_i)} P_i^0 + k_{i,R} P_{i,\min} \sum_{\kappa=\alpha_i(n_i)}^{k-1} (1-k_{i,R})^{k-1-\kappa} \\ &= (1-k_{i,R})^{k-\alpha_i(n_i)} P_i^0 + P_{i,\min} \left[1-(1-k_{i,R})^{k-\alpha_i(n_i)}\right] \\ &= (1-k_{i,R})^{k-\alpha_i(n_i)} \left(P_i^0 - P_{i,\min}\right) + P_{i,\min}. \end{aligned} \tag{8.8}$$

The sum of costs among the time interval $[\alpha_i(n_i), \alpha_i(n_i) + \delta_i(n_i)]$ is

$$J_{i,n_i}^1(\delta_i(n_i)) = \sum_{k=\alpha_i(n_i)}^{\alpha_i(n_i)+\delta_i(n_i)} P_i(k) = P_{i,\min}\left[\delta_i(n_i)+1\right] + (P_i^0 - P_{i,\min}) \frac{1-(1-k_{i,R})^{\delta_i(n_i)+1}}{k_{i,R}}.$$

Let t_1 denote the time step $\alpha_i(n_i)+\delta_i(n_i)$ and P_i^1 denote $P_i(t_1)$ which can be computed as $P_i(t_1) = (1-k_{i,R})^{\delta_i(n_i)} (P_i^0 - P_{i,\min}) + P_{i,\min}$, which is the initial value for the time interval $k \in [\alpha_i(n_i) + \delta_i(n_i), \alpha_i(n_i) + \delta_i(n_i) + C_i(n_i) - 1]$. With $u_i(k) = 1$ for $k \in [\alpha_i(n_i) + \delta_i(n_i), \alpha_i(n_i) + \delta_i(n_i) + C_i(n_i) - 1]$, the dynamic of robot i's performance value is

$$P_i(k) = (1 - k_{i,H}) P_i(k-1) + k_{i,H} P_{i,\max}$$

. Then for any $k \in [\alpha_i(n_i) + \delta_i(n_i) + 1, \alpha_i(n_i) + \delta_i(n_i) + C_i(n_i)]$, we have

$$\begin{aligned} P_i(k) &= (1-k_{i,H})^{k-t_1} P_i^1 + k_{i,H} P_{i,\max} \sum_{\kappa=t_1}^{k-1} (1-k_{i,H})^{k-1-\kappa} \\ &= (1-k_{i,H})^{k-t_1} \left(P_i^1 - P_{i,\max}\right) + P_{i,\max} \end{aligned} \tag{8.9}$$

The costs among the time interval $[\alpha_i(n_i)+\delta_i(n_i)+1, \alpha_i(n_i)+\delta_i(n_i)+C_i(n_i)]$ is

$$J_{i,n_i}^2(\delta_i(n_i)) = \sum_{k=\alpha_i(n_i)+\delta_i(n_i)+1}^{\alpha_i(n_i)+\delta_i(n_i)+C_i(n_i)} P_i(k) = \frac{1-k_{i,H}}{k_{i,H}} \left[1-(1-k_{i,H})^{C_i(n_i)}\right] \left(P_i^1 - P_{i,\max}\right) + C_i(n_i) P_{i,\max}$$

Let t_2 denote the time step $\alpha_i(n_i)+\delta_i(n_i)+C_i(n_i)$ and P_i^2 denote

$$\begin{aligned} P_i(t_2) &= (1-k_{i,H})^{C_i(n_i)} \left(P_i^1 - P_{i,\max}\right) + P_{i,\max} \qquad (8.10)\\ &= (1-k_{i,H})^{C_i(n_i)}(1-k_{i,R})^{\delta_i(n_i)} \left(P_i^0 - P_{i,\min}\right) + (1-k_{i,H})^{C_i(n_i)}(P_{i,\min}-P_{i,\max}) + P_{i,\max}. \end{aligned}$$

For $k \in [\alpha_i(n_i)+\delta_i(n_i)+C_i(n_i)+1, \alpha_i(n_i+1)-1]$, we have $u_i(k)=0$, which leads to

$$P_i(k) = (1-k_{i,R})^{k-t_2} \left(P_i^2 - P_{i,\min}\right) + P_{i,\min}. \qquad (8.11)$$

The costs among the time interval $[\alpha_i(n_i)+\delta_i(n_i)+C_i(n_i)+1, \alpha_i(n_i+1)-1]$ is

$$\begin{aligned} J_{i,n_i}^3(\delta_i(n_i)) &= \sum_{k=\alpha_i(n_i)+\delta_i(n_i)+C_i(n_i)]+1}^{\alpha_i(n_i+1)-1} P_i(k)\\ &= \frac{1-k_{i,R}}{k_{i,R}} \left[1-(1-k_{i,R})^{t_3-\delta_i(n_i)}\right] \left(P_i^2 - P_{i,\min}\right) + (t_3 - \delta_i(n_i))\, P_{i,\min} \end{aligned}$$

where $t_3 = T_i - 1 - C_i(n_i)$. Therefore, the cost is

$$J_{i,n_i}(\delta_i(n_i)) = J_{i,n_i}^1(\delta_i(n_i)) + J_{i,n_i}^2(\delta_i(n_i)) + J_{i,n_i}^3(\delta_i(n_i)).$$

And if we denote $a_i = \frac{1-k_{i,R}}{k_{i,R}} > 0$, $b_i = \frac{1-k_{i,H}}{k_{i,H}} > 0$, $c_i = P_i^0 - P_{i,\min} \geq 0$, $d_i = (1-k_{i,R})^{C_i(n_i)} > 0$, $e_i = (1-k_{i,H})^{C_i(n_i)} > 0$, $x_i = \ln(1-k_{i,R}) < 0$, $y_i = (1-k_{i,R})^{\delta_i(n_i)} > 0$ and then take the derivative of $J_{i,n_i}(\delta_i(n_i))$ as

$$\frac{dJ_{i,n_i}(\delta_i(n_i))}{d\delta_i(n_i))} = \frac{dJ_{i,n_i}^1(\delta_i(n_i))}{d\delta_i(n_i))} + \frac{dJ_{i,n_i}^2(\delta_i(n_i))}{d\delta_i(n_i))} + \frac{dJ_{i,n_i}^3(\delta_i(n_i))}{d\delta_i(n_i))}.$$

We compute the first term as

$$\frac{dJ_{i,n_i}^1(\delta_i(n_i))}{d\delta_i(n_i))} = -\frac{P_i^0 - P_{i,\min}}{k_{i,R}} \ln(1-k_{i,R})(1-k_{i,R})^{\delta_i(n_i)+1} + P_{i,\min} = -a_i c_i x_i y_i + P_{i,\min}.$$

The second term can be computed as

$$\frac{dJ_{i,n_i}^2(\delta_i(n_i))}{d\delta_i(n_i))} = \frac{1-k_{i,H}}{k_{i,H}}(P_i^0 - P_{i,\min})\left[1-(1-k_{i,H})^{C_i(n_i)}\right]\ln(1-k_{i,R})(1-k_{i,R})^{\delta_i(n_i)}$$
$$= b_i c_i (1-e_i) x_i y_i.$$

The third term is computed as

$$\frac{dJ_{i,n_i}^3(\delta_i(n_i))}{d\delta_i(n_i))} = \frac{1-k_{i,R}}{k_{i,R}}\left\{\left[1-(1-k_{i,R})^{t_3-\delta_i(n_i)}\right]\frac{dP_i^2}{d\delta_i(n_i))} + \frac{d\left[1-(1-k_{i,R})^{t_3-\delta_i(n_i)}\right]}{d\delta_i(n_i)}\left(P_i^2 - P_{i,\min}\right)\right\} - P_{i,\min}$$

with $\frac{dP_i^2}{d\delta_i(n_i))} = (1-k_{i,H})^{C_i(n_i)}(P_i^0 - P_{i,\min})\ln(1-k_{i,R})(1-k_{i,R})^{\delta_i(n_i)} = c_i e_i x_i y_i$ and $\frac{d\left[1-(1-k_{i,R})^{t_3-\delta_i(n_i)}\right]}{d\delta_i(n_i)} = \ln(1-k_{i,R})(1-k_{i,R})^{t_3-\delta_i(n_i)} = (1-k_{i,R})^{t_3-\delta_i(n_i)}x_i$, which leads to $\frac{dJ_{i,n_i}^3(\delta_i(n_i))}{d\delta_i(n_i))} = a_i c_i e_i x_i y_i + a_i(e_i-1)x_i(1-k_{i,R})^{t_3-\delta_i(n_i)}(P_{i,\min} - P_{i,\max}) - P_{i,\min}$.

Then the derivative of $J_{i,n_i}(\delta_i(n_i))$ is

$$\frac{dJ_{i,n_i}(\delta_i(n_i))}{d\delta_i(n_i))} = -a_i c_i x_i y_i + P_{i,\min} + b_i c_i (1-e_i) x_i y_i + a_i c_i e_i x_i y_i$$
$$+ a_i(e_i-1)x_i(1-k_{i,R})^{t_3-\delta_i(n_i)}(P_{i,\min} - P_{i,\max}) - P_{i,\min}$$
$$= (b_i - a_i)(1-e_i)c_i x_i y_i + a_i(e_i-1)x_i(1-k_{i,R})^{t_3-\delta_i(n_i)}(P_{i,\min} - P_{i,\max}).$$

Since $k_{i,H} < k_{i,R}$, we have $b_i - a_i > 0$. And it is trivial that $d_i - 1 < 0$, therefore $(b_i - a_i)(1-e_i)c_i x_i y_i \leq 0$ and $a_i(e_i-1)x_i(1-k_{i,R})^{t_3-\delta_i(n_i)}(P_{i,\min} - P_{i,\max}) < 0$, from which we can conclude that $\frac{dJ_{i,n_i}(\delta_i(n_i))}{d\delta_i(n_i))} < 0$. Therefore, when $\delta_i(n_i) = 0$, $J_i(\delta_i(n_i)) = J_i^{\max}$, which leads to (8.12). □

Based on Theorem 8.2.1, when there is no contention among robots, the optimal solution for $u_i(k)$ is

$$u_i(k) = \begin{cases} 1, k \in [\alpha_i(n_i), \alpha_i(n_i){+}C_i(n_i){-}1], \\ 0, k \in [\alpha_i(n_i){+}C_i(n_i), \alpha_i(n_i{+}1){-}1], \end{cases} \tag{8.12}$$

for all n_i such that $k_0 \le \alpha_i(n_i)$ and $\alpha_i(n_i{+}1) \le k_f$. And it is trivial that

$$\gamma_i(n_i) = \alpha_i(n_i) + C_i(n_i). \tag{8.13}$$

However, if a contention occurs when robot i starts execute and requests to collaborate with the human, then constraints (8.4) cannot be relaxed and equation (8.13) will not hold because it may be delayed by the collaboration between the human and other robots, i.e. $\delta_i(n_i) \neq 0$. And from the proof of Theorem 8.2.1, we know that the derivative $\frac{dJ_{i,n_i}(\delta_i(n_i))}{d\delta_i(n_i))}$ is strictly less than 0, which means the larger time delay $\delta_i(n_i)$ will further increase the cost in (8.6). Therefore, we need a timing model to compute the minimal value which the time delay variable $\delta_i(n_i)$ can take without violating the contention constraint, which will be introduced in the next section. This property also ensures that the human attention scheduling problem is analogous to the classic real-time scheduling problems.

8.3 Analytical Timing Model for Human-and-robot Collaboration System

Here we use SMA to derive a discrete-time analytical timing model for the non-preemptive human and robots collaboration system.

At each time k on our time horizon $[k_0, k_f]$ of the optimization problem, we define our timing state variable $Z(k) = (D(k), R(k), O(k), \mathrm{ID}(k))$ as follows.

Definition 8.3.1 *The deadline variable is $D(k) = (d_1(k), ..., d_i(k), ..., d_N(k))$, where $d_i(k)$ is defined to be how long after time k the next generation time $\alpha_i(n_i)$ of task τ_{i,n_i} will be generated. The remaining time variable is $R(k) = (r_1(k), ..., r_i(k), ..., r_N(k))$, where*

$r_i(k)$ denotes the remaining collaboration time after time k that is needed to complete the collaboration of the most recently generated task τ_{i,n_i}. The delay variable is $O(k) = (o_1(k), ..., o_i(k), ..., o_N(k))$, where $o_i(k)$ is how long the collaboration completion of task τ_{i,n_i} has been delayed from its most recent request time $\alpha_i(n_i)$ to time k.

Definition 8.3.2 *The index variable is $\mathrm{ID}(k)$ is index of the robot that is collaborating with the human operator at time k, where $\mathrm{ID}(k) \neq 0$ implies that the human attention is occupied by one robot and $\mathrm{ID}(k) = 0$ implies that no robot is occupying the human attention at time k.*

To support the dynamic timing model, we redefine the collaboration time of a task as follows:

Definition 8.3.3 *For all i, $n_i \geq 0$, and $k \in [\alpha_i(n_i), \alpha_i(n_i+1)]$, we set $C_i(k) = C_i(n_i)$ for $k \in [\alpha_i(n_i), \alpha_i(n_i+1))$.*

The evolution rules for $Z(k)$ within a time interval $[k_0, k_f]$ can be expressed by mathematical equations. These equations lead to a timing model. It is an analytical model that is efficient to compute, and it supports the implementation of contention-resolving MPC.

8.3.1 Timing Model for Human Attention Scheduling

We express our evolution rules for $Z(k)$ on $[k_0, k_f]$. We divide $[k_0, k_f]$ into sub-intervals $[k_w, k_{w+1}]$ such that

$$\begin{aligned} k_{w+1} - k_w = {} & \mathrm{sgn}(\mathrm{ID}(k_w)) \min\{r_{\mathrm{ID}}(k_w), d_1(k_w), ..., d_N(k_w), k_f - k_w\} \\ & + (1 - \mathrm{sgn}(\mathrm{ID}(k_w))) \min\{d_1(k_w), ..., d_N(k_w), k_f - k_w\} \end{aligned} \tag{8.14}$$

for all w, where $r_{\mathrm{ID}}(k)$ is a simplified notation for the remaining time $r_{\mathrm{ID}(k)}(k)$ of timing state variable ID.

At time k_w, if $r_{\mathrm{ID}}(k_w-1) > 1$, which means the robot $\mathrm{ID}(k_w-1)$ that was occupying the human attention has not completed the collaboration at time k_w, then $\mathrm{ID}(k_w)$ is the same as $\mathrm{ID}(k_w-1)$ because the collaboration is non-preemptive. If $r_{\mathrm{ID}}(k_w-1) = 1$, which means the robot $\mathrm{ID}(k_w-1)$ completed the collaboration at time k_w, then $\mathrm{ID}(k_w)$ need to switch to the robot which is scheduled to collaborate with the human operator, i.e. the robot i with $u_i(k_w) = 1$. Combining these two cases, the evolution rule for the timing state variable ID can be expressed as

$$\mathrm{ID}(k_w) = \mathrm{ID}(k_w-1)\,\mathrm{sgn}(r_{\mathrm{ID}}(k_w-1)-1) + \underset{i}{\mathrm{argmax}}\,\{u_i(k_w)\}\big[1-\mathrm{sgn}(r_{\mathrm{ID}}(k_w-1)-1)\big] \tag{8.15}$$

where sgn is defined by $\mathrm{sgn}(q) = 1$ if $q > 0$ and $\mathrm{sgn}(q) = 0$ if $q = 0$. If the set $\Lambda(k_w)$ is empty, then $\mathrm{ID}(k_w) = 0$.

The values of the timing state variables d_i, r_i and o_i have jumps for some i. If the deadline variable of robot i satisfies $d_i(k_w-1) = 1$, then

$$d_i(k_w) = T_i, r_i(k_w) = C_i(k_w) \text{ and } o_i(k_w) = 0. \tag{8.16}$$

If $d_i(k_w-1) > 1$, then there are no jumps for the timing states for robot i and we have

$$\begin{aligned} &d_i(k_w) = d_i(k_w-1)-1, r_i(k_w) = r_i(k_w-1) - \mathbb{1}(\mathrm{ID}(k_w-1) = i),\\ &\text{and } o_i(k_w) = o_i(k_w-1) + \mathrm{sgn}(r_i(k_w-1)). \end{aligned} \tag{8.17}$$

where $\mathbb{1}(\cdot)$ is an indicator function which is defined to be 1 if the condition $\mathrm{ID}(k_w-1) = i$ holds and 0 otherwise. Combining the two cases depending on the different values of $d_i(k_w-1)$, the evolution rules of the timing state variables d_i, r_i and o_i at the times k_w can be summarized as

$$
\begin{aligned}
&d_i(k_w) = d_i(k_w-1) - 1 + \big[1 - \mathrm{sgn}(d_i(k_w-1)-1)\big] T_i, \qquad (8.18)\\
&r_i(k_w) = \mathrm{sgn}(d_i(k_w-1)-1)\big[r_i(k_w-1) - \mathbb{1}(\mathrm{ID}(k_w-1)=i)\big] + \big[1-\mathrm{sgn}(d_i(k_w-1)-1)\big] C_i(k_w)\\
&o_i(k_w) = \big[o_i(k_w-1) + \mathrm{sgn}(r_i(k_w-1))\big]\ \mathrm{sgn}(d_i(k_w-1)-1)
\end{aligned}
$$

During any time $k_w + \Delta k \in [k_w{+}1, k_{w+1}{-}1]$, the state $\mathrm{ID}(k_w + \Delta k)$ remains unchanged because $k_{w+1} - k_w \leq r_{\mathrm{ID}}(k_w)$. If $ID(k_w) \neq 0$, the evolution rules for control system $ID(k_w)$ are

$$
\begin{aligned}
&d_{\mathrm{ID}}(k_w + \Delta k) = d_{\mathrm{ID}}(k_w) - \Delta k, r_{\mathrm{ID}}(k_w + \Delta k) = r_{\mathrm{ID}}(k_w) - \Delta k,\\
&\text{and } o_{\mathrm{ID}}(t_w + \Delta t) = o_{\mathrm{ID}}(t_w) + \Delta t \qquad (8.19)
\end{aligned}
$$

where $d_{\mathrm{ID}}(k)$ and $o_{\mathrm{ID}}(k)$ are defined analogously to $r_{\mathrm{ID}}(k)$. And during this time window

$$u_{\mathrm{ID}}(k_w + \Delta k) = 1. \qquad (8.20)$$

For robot i where $i \neq ID(k_w)$, the evolution rules are

$$
\begin{aligned}
&d_i(k_w + \Delta k) = d_i(k_w) - \Delta k,\ r_i(k_w + \Delta k) = r_i(k_w)\\
&\text{and } o_i(k_w + \Delta k) = o_i(k_w) + \mathrm{sgn}(r_i(k_w))\Delta k. \qquad (8.21)
\end{aligned}
$$

During this time window, for robot i where $i \neq ID(k_w)$

$$u_i(k_w + \Delta k) = 0. \qquad (8.22)$$

Combining all of the evolution rules in (8.14)–(8.21) leads to the timing model of non-preemptive human attention scheduling, which provides the value of $Z(k)$ at each time k, given the initial state variable $Z(k_0)$, the vehicle timing parameters $C_i(n_i)$ and T_i for all i and n_i, and the value of $\mathbf{u}(k_0 \sim k)$, where $\mathbf{u}(k_0 \sim k)$ is our simplified notation for the decision variable for all robots at all time steps in the interval $[k_0, k]$.

Remark 8 *Notice that the only times when the value of timing state depend on the decision variable u_i is at the significant moments k_w. Therefore, we only need to determine the value of u_i at those times.*

8.4 Contention-Resolving MPC Algorithm

In this section, we consider the original problem formulation where constraints (8.3) and (8.4) are not relaxed. We convert the problem formulated by (8.6) into a path planning problem that can be solved iteratively. The conversion is based on the insight that value of the decision variable $\mathbf{u}$ only need to be decided at the significant moments when contention occurs.

8.4.1 Construction of Decision Tree

We use the timing model to determine when contentions occur by checking the following condition:

Proposition 8.4.1 *A contention starts at time k if and only if the following three conditions hold:*

$$\sum_{i=1}^{N} [1 - \mathrm{sgn}(C_i(k) - r_i(k))] \geq 2, \; r_{\mathrm{ID}}(k-1) \leq 1 \tag{8.23}$$

and $k = k_w$ for some i and some w where k_w is a significant moment computed by equation (8.14).

Proof. A collaboration request from robot i is waiting at a time k or is generated at time k if and only if $r_i(k) = C_i(k)$, i.e., $1 - \mathrm{sgn}(C_i(k) - r_i(k)) = 1$. Therefore,

$\sum_{i=1}^{N} [1 - \text{sgn}(C_i(k) - r_i(k))] \geq 2$ if and only if two or more robots are waiting to collaborate with the human operator at time k or generating requests at time k. Therefore, if a contention starts at time k, then k is one significant moment k_w for some i and w. And the robot that was collaborating with the human one time step before k will either finish the collaboration at time k, i.e., $r_{\text{ID}}(k-1) = 1$ or the human was not collaborating with any robot one time step before k, i.e., $r_{\text{ID}}(k-1) = 0$, so the condition $r_{\text{ID}}(k-1) \leq 1$ from (8.23) holds. Conversely, if the three conditions (8.23) are satisfied, then at time k, multiple robots are in contention to collaborate with the human which is a necessary condition, so a contention starts at time k. $\square$

Based on the contention times, we can construct a decision tree. Each possible value of $\mathbf{u}$ will produce a branch of a decision tree. An example of scheduling three robots for four consecutive contentions is shown in Figure 8.1. The upper part of the figure shows the constructed decision tree and the lower part shows the human attention occupation scheduled along the path with blue arrows. The infinite-time feasibility function of the discrete-time system represented by (5.4). The branch costs $w_{i,j}$ will be defined below.

8.4.2 Branch Cost

After constructing the decision tree, we define a cost for each branch. Along one branch (l, j) associated, since the decision variables $\mathbf{u}(k)$ are determined for all i and $k \in [k_l^c, k_j^c]$, we can calculate the significant moments $\gamma_i(n_i)$ for all i and n_i such that $k_l^c \leq \gamma_i(n_i) \leq k_j^c$ as follows:

$$
\begin{aligned}
&Z(k) = \mathbb{H}\Big(k; Z(k_l^c), (C_i(n_i), T_i)_{i=1,\ldots,N}, \mathbf{u}_m\Big) \text{ and} \\
&\gamma_i(n_i) = \alpha_i(n_i) + o_i(\alpha_i(n_i+1)-1) + \text{sgn}(r_i(\alpha_i(n_i+1-1)) \qquad (8.24)
\end{aligned}
$$

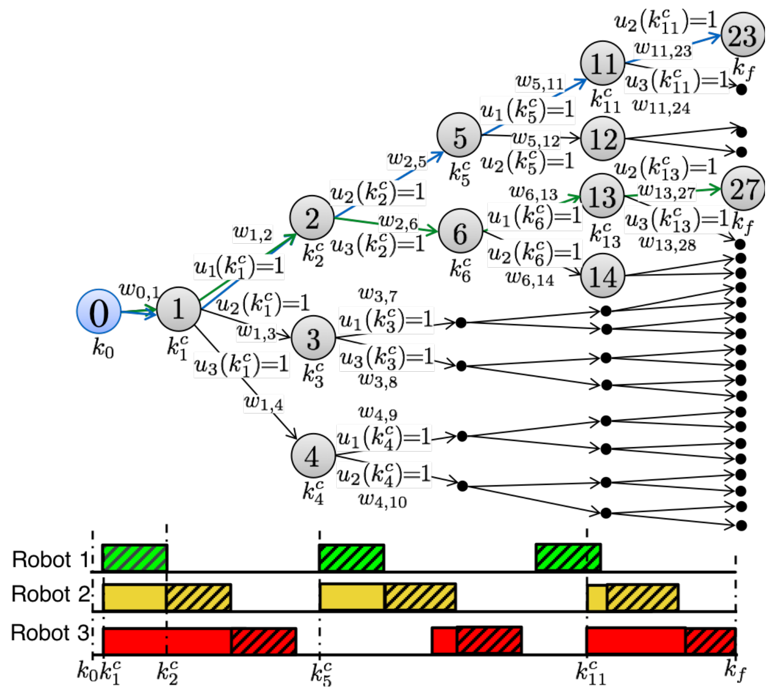


Figure 8.1: Decision tree for discrete-time contention-resolving MPC.

where $r_i(\alpha_i(n_i{+}1){-}1)$ and $o_i(\alpha_i(n_i{+}1){-}1)$ for each n_i are generated by the timing model except with a known $\mathbf{u}_m$. Then the branch cost $w_{l,j}$ is defined as

$$w_{l,j} = \sum_{i=1}^{N} w_{l,j}^i \tag{8.25}$$

where $w_{l,j}^i$ is the cost of robot i. For each i such that there is a completion time $\gamma_i(n_i+1) \in (t_l^c, t_j^c]$, let $\underline{n}_i$ be the smallest index n_i satisfying $\gamma_i(n_i+1) > k_l^c$ and $\overline{n}_i$ be the largest index n_i satisfying $\gamma_i(n_i+1) \leq k_j^c$. Then we set

$$w_{l,j}^i = \sum_{k=\gamma_i(\underline{n}_i)}^{\gamma_i(\overline{n}_i)-1} [P_{i,\max} - P_i(k)] \text{ with } \mathbf{u}(k) = \mathbf{u}_m(k) \text{ for } k \in [\gamma_i(\underline{n}_i), \gamma_i(\overline{n}_i) - 1]. \tag{8.26}$$

If no collaboration of robot i is completed within $[t_l^c, t_j^c]$, i.e. $\underline{n}_i > \overline{n}_i$, then we define $w_{l,j} = 0$. And if for any time $k \in [t_l^c, t_j^c]$, we have $\mathcal{T}_i(k) < \mathcal{T}_{i,\min}$ or $\mathcal{T}_i(k) > \mathcal{T}_{i,\max}$, then we define $w_{l,j} = +\infty$. The meaning of (8.26) is as follows. If $\gamma_i(n_i) \in (k_l^c, k_j^c]$, i.e., the n_ith collaboration of robot i is completed between the contention times k_l^c and k_j^c, then the cost of the n_ith task is included in the branch cost $w_{l,j}$. This branch cost formulation ensures that all costs included in one branch are determined and will not be changed by the decision variable $\mathbf{u}$ at or after time k_j^c. The cost of an uncompleted $(\overline{n}_i+1)$st collaboration will be included by the branches following the branch (l, j).

8.4.3 Search Algorithm

Based on the decision tree, the integer optimization problem in Section 8.2.1 can now be converted to the problem of finding a path from k_0 to k_f such that the whole cost along the path is lowest. In Chapter 5, we presented the contention-resolving MPC framework that leverages the A-star algorithm to search for an optimal path in the decision tree. We define the same stage cost $C_g(v_l)$ to be the same as Chapter 5. And the heuristic future cost $\hat{C}_h(v_l)$ to be

$$\hat{C}_h(v_l) = \sum_{i=1}^{N} \sum_{k=\gamma_i(\overline{n}_i)}^{k_f} P_i(k) \text{ subject to } (8.1),$$

$$u_i(k) = 1, k \in [\alpha_i(n_i), \alpha_i(n_i)+C_i(n_i)-1],\ u_i(k) = 0, k \in [\alpha_i(n_i)+C_i(n_i), \alpha_i(n_i+1)-1],$$

$$\text{for all } n_i \text{ such that } k_l^c \leq \alpha_i(n_i) \text{ and } \alpha_i(n_i+1) \leq k_f.$$

which is the cost without considering contention constraints and is less than or equal to the true future constraints. We have shown in Chapter 5 that the minimal cost path is guaranteed to be found with these defined costs. The search algorithm does not generate the whole decision tree. Instead, it efficiently generates a subtree without losing optimality.

8.5 Simulation Results

We simulate three robots collaborating with one human operator. The starting and ending time instants are $k_0 = 0$ and $k_f = 120$ respectively. The parameters for trust model are $A_i = 1$, $B_i = 0.605$, $C_i = 0.6$, $D_i = 0$ and $F_i = 0$ for all $i = 1, 2, 3$.

The initial values of trust level are

$$[\mathcal{T}_1(0), \mathcal{T}_2(0), \mathcal{T}_3(0)] = [1.93, 1.9, 1.98].$$

The lower bounds of the trust level are

$$[\mathcal{T}_{1,\min}, \mathcal{T}_{2,\min}, \mathcal{T}_{3,\min}] = [1.55, 1.65, 1.7].$$

The upper bounds of the trust level are

$$[\mathcal{T}_{1,\max}, \mathcal{T}_{2,\max}, \mathcal{T}_{3,\max}] = [2.15, 2.35, 2.1].$$

The initial values of performance are

$$[P_1^0(n_i), P_2^0(n_i), P_3^0(n_i)] = [0.7, 0.7, 0.7] \text{ for all } n_i.$$

The parameters for performance model are

$$[k_{1,R}, k_{2,R}, k_{3,R}] = [0.25, 0.25, 0.25] \text{ and } [k_{1,H}, k_{2,H}, k_{3,H}] = [0.1, 0.13, 0.15].$$

The lower and upper bounds are

$$[P_{1,\min}, P_{2,\min}, P_{3,\min}] = [0.6, 0.65, 0.65] \text{ and } [P_{1,\max}, P_{2,\max}, P_{3,\max}] = [0.75, 0.75, 0.75].$$

The timing parameters are

$$[C_1(n_i), C_2(n_i), C_3(n_i)] = [6, 6, 6] \text{ for all } n_i \text{ and } [T_1, T_2, T_3] = [20, 30, 40].$$

The human attention occupation result is shown in Figure 8.2. The robot performance is shown in Figure 8.3. Five contentions occur in the time interval $[0, 120]$. The cost under

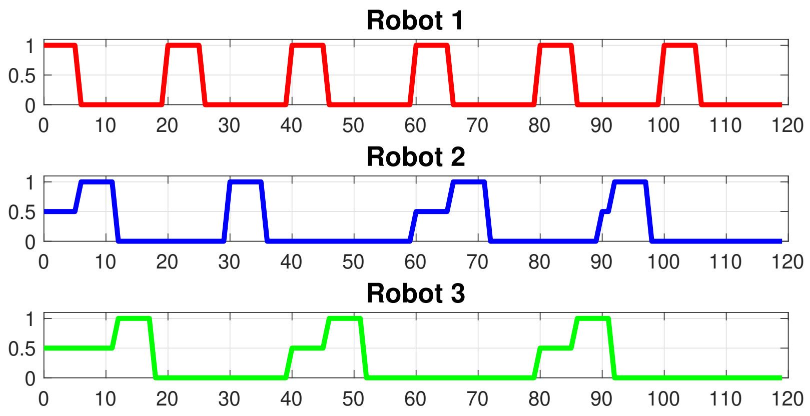


Figure 8.2: Human attention occupation for collaborating with three robots. The y axis value 1 means that the robot is collaborating witht he human, 0 means that the robot is not requesting the collaboration, and 0.5 means that the robot's collaboration request is delayed by a contention.

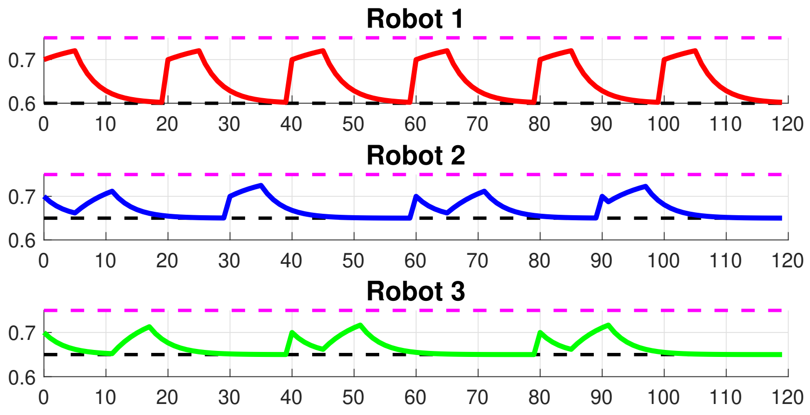


Figure 8.3: Performance values of three robots under the optimal schedule. The magenta dashed line represents $P_{i,\max}$ and the black dashed line represents $P_{i,\min}$.

optimal schedule is 31.4262, which is 11.99% less than the HTS scheduling strategy. While the example is simple, the simulation results show that our method performs better than the HTS.

CHAPTER 9
CONCLUSION AND FUTURE WORK

9.1 Conclusion

While model predictive control has gained popularity in process engineering and networked control systems, the previously available methods had difficulties coping with the co-design of optimal controls and priority assignments that occur in coupled control systems with shared resources. Resolving contentions in coupled control systems with shared resources is a challenging problem that is of compelling ongoing engineering interest. This thesis leads to new insights in scheduling and control co-design methods under contentions. The major contributions of this thesis are:

1. *The novel contention-resolving MPC frame work.* The contention-resolving MPC is a unique way to solve the special mixed integer optimization problem when dealing with the scheduling and control co-design. Enabled by the critical time instants when contention occurs, the contention-resolving MPC is able to decouple the priority and control design at the contention time instants and discretize the continuous planning time horizon into time intervals where the priority assignment remains to be fixed. The carefully designed decision tree structure and branch costs allow us to convert the coupled priority and vehicle speed control optimization problem into a path planning problem. And contention-resolving MPC can be proved to find the global optimal solution for the formulated scheduling and control co-design optimization problem under basic requirements and the condition of immediate access (or CIA) in real-time scheduling theory.

2. *Critical time instants derived by the timing states and SMA.* In this thesis, we have shown how to derive the classical Critical-time Instants Analysis through the math-

ematical equations instead of logical reasoning in [9], which give us a more systematical way to find the critical-time instants and analyze the schedulabilty in more complicated real-time system such as non-preemptive and aperiodic systems.

9.2 Future Work

In this section, we will discuss the future direction of the research presented in this thesis.

9.2.1 Critical time instants for Non-preemptive Systems

In real-world application, non-preemptive scheduling is important for a variety of reasons. For example, in many practical real-time scheduling problems such as I/O scheduling, properties of device hardware and software either make preemption impossible or prohibitively expensive. And the overhead of preemptive algorithms is more difficult to characterize and predict than that of non-preemptive algorithms. Since scheduling overhead is often ignored in scheduling models (including ours), an implementation of a non-preemptive scheduler will be closer to the formal model than an implementation of a preemptive scheduler.

Also, a recent trend in scheduling and control field is to use event-triggered system. Tasks in event-driven systems are aporadic because of random user inputs or non-periodic device interrupts. Events occur repeatedly, but the time interval between consecutive occurrences varies and can be arbitrarily large. Most existing research works describe sufficient conditions for scheduling non-preemptive and aperiodic tasks. The work in [110] gives necessary and sufficient conditions only for discrete-time system. For general aperiodic and non-preemptive tasks, when the worst case scenario will occurs and how to find the worst case condition is still an open question.

The timing states and siginificant moment analysis shows the potential to find the critical time and the worst-case conditions in general aperiodic and non-preemptive system. Once the worst case scenario can be found, the schedulability condition can also be developed.

9.2.2 Multiple-lane and Multiple-intersection Scheduling

In real-world intersections, more than two roads may intersect each other. In addition, each road can contain multiple lanes, each lane can host multiple vehicles, and each vehicle can perform multiple moves, e.g., a right turn, left turn, or lane shift. The intersection can be occupied by multiple vehicles at once, if the trajectories of the vehicles do not overlap. Therefore, the intersection area needs to be modeled as a shared resource that allows access by multiple customers. The work presented in this thesis has not addressed a complex contention relationship like this, which offers opportunity to advance the contention-resolving MPC.

Besides, we will investigate coordinated scheduling of multiple intersections. Locally optimal strategies for one intersection usually do not lead to an optimal strategy for multiple intersections. The challenge for scheduling multiple intersections is that the computational power for solving mixed integer programming problems will be extremely high due to the large number of decision variables, so we will pursue a distributed optimization approach. What information is shared among neighboring intersections by the distributed algorithms must be designed. Most existing work on distributed optimization does not apply to distributed mixed integer programming [114, 115, 116]. This may trigger a new research direction for distributed optimization.

REFERENCES

[1] J. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–162, 2007.

[2] G. C. Walsh, Y. Hong, and L. G. Bushnell, "Stability analysis of networked control systems," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 3, pp. 438–446, 2002.

[3] Z. Shi, N. Yao, and F. Zhang, "Scheduling feasibility of energy management in micro-grids based on significant moment analysis," pp. 431–449, 2017.

[4] J. Rios-Torres and A. A. Malikopoulos, "A survey on the coordination of connected and automated vehicles at intersections and merging at highway on-ramps," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1066–1077, 2017.

[5] X. Wang, Z. Shi, F. Zhang, and Y. Wang, "Mutual trust based scheduling for (semi)autonomous multi-agent systems," in *Proceedings of the 2015 American Control Conference (Chicago, IL, 1-3 July 2015)*, pp. 459–464.

[6] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-Time Systems*, vol. 28, no. 2-3, pp. 101–155, 2004.

[7] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney, "Task scheduling for control oriented requirements for cyber-physical systems," in *Real-Time Systems Symposium, 2008*, IEEE, 2008, pp. 47–56.

[8] X. Wang, Z. Shi, F. Zhang, and Y. Wang, "Dynamic real-time scheduling for human-agent collaboration systems based on mutual trust," *Cyber-Physical Systems*, vol. 1, no. 2-4, pp. 76–90, 2015.

[9] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[10] N. Yao, M. Malisoff, and F. Zhang, "Contention resolving optimal priority assignment for event-triggered model predictive controllers," in *Proceedings of the 2017 American Control Conference*, IEEE, 2017, pp. 2357–2362.

[11] J. Lee and B. Park, "Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 81–90, 2012.

[12] A. A. Malikopoulos, C. G. Cassandras, and Y. J. Zhang, "A decentralized energy-optimal control framework for connected automated vehicles at signal-free intersections," *Automatica*, vol. 93, pp. 244–256, 2018.

[13] Y. J. Zhang, A. A. Malikopoulos, and C. G. Cassandras, "Optimal control and coordination of connected and automated vehicles at urban traffic intersections," in *Proceedings of the American Control Conference*, IEEE, 2016, pp. 6227–6232.

[14] S. Engell and I. Harjunkoski, "Optimal operation: Scheduling, advanced control and their integration," *Computers & Chemical Engineering*, vol. 47, pp. 121–133, 2012.

[15] W. Chen, J. Yao, and L. Qiu, "Networked stabilization of multi-input systems over shared channels with scheduling/control co-design," *Automatica*, vol. 99, pp. 188–194, 2019.

[16] A. Farnam and R. M. Esfanjani, "Improved stabilization method for networked control systems with variable transmission delays and packet dropout," *ISA transactions*, vol. 53, no. 6, pp. 1746–1753, 2014.

[17] H. Gao, T. Chen, and J. Lam, "A new delay system approach to network-based control," *Automatica*, vol. 44, no. 1, pp. 39–52, 2008.

[18] C. Peng and T. C. Yang, "Event-triggered communication and H∞ control co-design for networked control systems," *Automatica*, vol. 49, no. 5, pp. 1326–1332, 2013.

[19] Z. Shi and F. Zhang, "Model predictive control under timing constraints induced by controller area networks," *Real-Time Systems*, pp. 1–32, 2015.

[20] C. Zhou, M. Du, and Q. Chen, "Co-design of dynamic scheduling and h-infinity control for networked control systems," *Applied Mathematics and Computation*, vol. 218, no. 21, pp. 10 767–10 775, 2012.

[21] M. M. B. Gaid, A. Cela, and Y. Hamam, "Optimal integrated control and scheduling of networked control systems with communication constraints: Application to a car suspension system," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 4, pp. 776–787, 2006.

[22] M. E. M. B. Gaid, A. S. Cela, and Y. Hamam, "Optimal real-time scheduling of control tasks with state feedback resource allocation," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 2, pp. 309–326, 2009.

[23] S. K. Mazumder, K. Acharya, and M. Tahir, "Joint optimization of control performance and network resource utilization in homogeneous power networks," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 5, pp. 1736–1745, 2009.

[24] L. Yao, W.-C. Chang, and R.-L. Yen, "An iterative deepening genetic algorithm for scheduling of direct load control," *IEEE Transactions on Power Systems*, vol. 20, no. 3, pp. 1414–1421, 2005.

[25] D. Roy, L. Zhang, W. Chang, D. Goswami, and S. Chakraborty, "Multi-objective co-optimization of flexray-based distributed control systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, IEEE, 2016, pp. 1–12.

[26] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[27] J. B. Rawlings and D. Q. Mayne, *Model predictive control: Theory and design*. Nob Hill Pub., 2009.

[28] L. Baskar, B. De Schutter, and H. Hellendoorn, "Model-based predictive traffic control for intelligent vehicles: Dynamic speed limits and dynamic lane allocation," in *Proceedings of the IEEE Intelligent Vehicles Symposium (Eindhoven, Netherlands, 4-6 June 2008)*, 2008, pp. 174–179.

[29] T. Bellemans, B. De Schutter, and B. De Moor, "Model predictive control for ramp metering of motorway traffic: A case study," *Control Engineering Practice*, vol. 14, no. 7, pp. 757–767, 2006.

[30] J. R. D. Frejo and E. F. Camacho, "Global versus local MPC algorithms in freeway traffic control with ramp metering and variable speed limits," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1556–1565, 2012.

[31] M. Lješnjanin, D. E. Quevedo, and D. Nešić, "Packetized mpc with dynamic scheduling constraints and bounded packet dropouts," *Automatica*, vol. 50, no. 3, pp. 784–797, 2014.

[32] R. Negenborn, B. De Schutter, and J. Hellendoorn, "Multi-agent model predictive control for transportation networks: Serial versus parallel schemes," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 3, pp. 353–366, 2008.

[33] L. Shi, S. Bart De, X. Yugeng, and H. Hans, "Fast model predictive control for urban road networks via MILP," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 846–856, 2011.

[34] A. Afram and F. Janabi-Sharifi, "Theory and applications of HVAC control systems–a review of model predictive control (MPC)," *Building and Environment*, vol. 72, pp. 343–355, 2014.

[35] C. R. Touretzky and M. Baldea, "Integrating scheduling and control for economic mpc of buildings with energy storage," *Journal of Process Control*, vol. 24, no. 8, pp. 1292–1300, 2014.

[36] Y. Zhao, Y. Lu, C. Yan, and S. Wang, "Mpc-based optimal scheduling of grid-connected low energy buildings with thermal energy storages," *Energy and Buildings*, vol. 86, pp. 415–426, 2015.

[37] Y. Chu and F. You, "Moving horizon approach of integrating scheduling and control for sequential batch processes," *AIChE Journal*, vol. 60, no. 5, pp. 1654–1671, 2014.

[38] G. Liu, J. Sun, and Y. Zhao, "Design, analysis and real-time implementation of networked predictive control systems," *Acta Automatica Sinica*, vol. 39, no. 11, pp. 1769–1777, 2013.

[39] G. Liu, Y. Xia, J. Chen, D. Rees, and W. Hu, "Networked predictive control of systems with random network delays in both forward and feedback channels," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 3, pp. 1282–1297, 2007.

[40] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[41] S. A. Fayazi and A. Vahidi, "Vehicle-in-the-loop (VIL) verification of a smart city intersection control scheme for autonomous vehicles," in *Proceedings of the IEEE Conference on Control Technology and Applications*, IEEE, 2017, pp. 1575–1580.

[42] F. Yan, M. Dridi, and A. El Moudni, "An autonomous vehicle sequencing problem at intersections: A genetic algorithm approach," *International Journal of Applied Mathematics and Computer Science*, vol. 23, no. 1, pp. 183–200, 2013.

[43] N. Yao and F. Zhang, "Resolving contentions for intelligent traffic intersections using optimal priority assignment and model predictive control," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*, IEEE, 2018, pp. 632–637.

[44] N. Yao, M. Malisoff, and F. Zhang, “Contention-resolving model predictive control for coordinating automated vehicles at a traffic intersection,” in *2019 IEEE 58th Conference on Decision and Control (CDC)*, IEEE, 2019, pp. 2233–2238.

[45] ——, “Contention-resolving model predictive control for coupled control systems with a shared resource,” *Automatica*, Submitted,

[46] N. Yao and F. Zhang, “Optimal real-time scheduling of human attention for a human and multi-robot collaboration system,” in *Proceedings of the 2020 American Control Conference*, IEEE, 2020, Accepted.

[47] J. Xu and D. L. Parnas, “Scheduling processes with release times, deadlines, precedence and exclusion relations,” *IEEE Transactions on software engineering*, vol. 16, no. 3, pp. 360–369, 1990.

[48] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of scheduling*. Courier Corporation, 2003.

[49] J. Goossens, S. Funk, and S. Baruah, “Priority-driven scheduling of periodic task systems on multiprocessors,” *Real-time systems*, vol. 25, no. 2-3, pp. 187–205, 2003.

[50] M. Bertogna and M. Cirinei, “Response-time analysis for globally scheduled symmetric multiprocessor platforms,” in *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, IEEE, 2007, pp. 149–160.

[51] S. K. Baruah and S. Chakraborty, “Schedulability analysis of non-preemptive recurring real-time tasks,” in *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, IEEE, 2006, 8–pp.

[52] V. Van Peteghem and M. Vanhoucke, “A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem,” *European Journal of Operational Research*, vol. 201, no. 2, pp. 409–418, 2010.

[53] M. Bertogna, G. Buttazzo, and G. Yao, “Improving feasibility of fixed priority tasks using non-preemptive regions,” in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, IEEE, 2011, pp. 251–260.

[54] R. B. Gmbh, “CAN specification (version 2.0),” 1991.

[55] G. C. Walsh and H. Ye, “Scheduling of networked control systems,” *IEEE control systems magazine*, vol. 21, no. 1, pp. 57–65, 2001.

[56] E. Bini, G. C. Buttazzo, and G. M. Buttazzo, "Rate monotonic analysis: The hyperbolic bound," *IEEE Transactions on Computers*, vol. 52, no. 7, pp. 933–942, 2003.

[57] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *RTSS*, vol. 89, 1989, pp. 166–171.

[58] T. F. Abdelzaher, V. Sharma, and C. Lu, "A utilization bound for aperiodic tasks and priority driven scheduling," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 334–350, 2004.

[59] S. K. Baruah, "The non-preemptive scheduling of periodic tasks upon multiprocessors," *Real-Time Systems*, vol. 32, no. 1-2, pp. 9–20, 2006.

[60] M. Marouf, L. George, and Y. Sorel, "Schedulability analysis for a combination of non-preemptive strict periodic tasks and preemptive sporadic tasks," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, IEEE, 2012, pp. 1–8.

[61] Z. Shi and F. Zhang, "Predicting time-delays under real-time scheduling for linear model predictive control," in *Proceedings of the 2013 International Conference on Computing, Networking and Communications*, IEEE, 2013, pp. 205–209.

[62] X. Wang, Z. Shi, F. Zhang, and Y. Wang, "Dynamic real-time scheduling for human-agent collaboration systems based on mutual trust," *Cyber-Physical Systems*, vol. 1, no. 2-4, pp. 76–90, 2015.

[63] K.-E. Arzén, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, IEEE, vol. 5, 2000, pp. 4865–4870.

[64] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–162, 2007.

[65] S. Longo, T. Su, G. Herrmann, and P. Barber, *Optimal and Robust Scheduling for Networked Control Systems*. Boca Raton, FL: CRC Press, 2013.

[66] M. Miskowicz, *Event-Based Control and Signal Processing*. Boca Raton, FL: CRC Press, 2015.

[67] K.-E. Årzén, A. Cervin, J. Eker, and L. Sha, "An introduction to control and scheduling co-design," in *Proceedings of the 39th IEEE Conference on Decision and Control (CDC 2000)*, 2000, pp. 4865–4870.

[68] D. Antunes, W. Heemels, and P. Tabuada, “Dynamic programming formulation of periodic event-triggered control: Performance guarantees and co-design,” in *Proceedings of the 51st IEEE Conference on Decision and Control (CDC 2012)*, IEEE, 2012, pp. 7212–7217.

[69] W. Zhang, M. S. Branicky, and S. M. Phillips, “Stability of networked control systems,” *IEEE control systems magazine*, vol. 21, no. 1, pp. 84–99, 2001.

[70] D. Goswami, A. Masrur, R. Schneider, C. J. Xue, and S. Chakraborty, “Multirate controller design for resource-and schedule-constrained automotive ecus,” in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2013, pp. 1123–1126.

[71] P. Marti, J. M. Fuertes, G. Fohler, and K. Ramamritham, “Jitter compensation for real-time control systems,” in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*, IEEE, 2001, pp. 39–48.

[72] E. L. Lawler and D. E. Wood, “Branch-and-bound methods: A survey,” *Operations research*, vol. 14, no. 4, pp. 699–719, 1966.

[73] L. Davis, “Handbook of genetic algorithms,” 1991.

[74] R. Hult, M. Zanon, S. Gras, and P. Falcone, “An miqp-based heuristic for optimal coordination of vehicles at intersections,” in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, 2018, pp. 2783–2790.

[75] R. Tachet, P. Santi, S. Sobolevsky, L. Reyes-Castro, E. Frazzoli, D. Helbing, and C. Ratti, “Revisiting street intersections using slot-based systems,” *PloS One*, vol. 11, no. 3, e0149607, 2016.

[76] K. Zhang, D. Zhang, A. de La Fortelle, X. Wu, and J. Gregoire, “State-driven priority scheduling mechanisms for driverless vehicles approaching intersections,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2487–2500, 2015.

[77] K. Zhang, A. Yang, H. Su, A. de La Fortelle, K. Miao, and Y. Yao, “Service-oriented cooperation models and mechanisms for heterogeneous driverless vehicles at continuous static critical sections,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 7, pp. 1867–1881, 2017.

[78] Y. Meng, L. Li, F.-Y. Wang, K. Li, and Z. Li, “Analysis of cooperative driving strategies for nonsignalized intersections,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 4, pp. 2900–2911, 2018.

[79] P. Jiao, H. Wang, and T. Sun, “Real-time arterial coordination control based on dynamic intersection turning fractions estimation using genetic algorithm,” *Mathematical Problems in Engineering*, vol. 2014, 2014.

[80] Q. Lu and K.-D. Kim, “A genetic algorithm approach for expedited crossing of emergency vehicles in connected and autonomous intersection traffic,” *Journal of Advanced Transportation*, vol. 2017, 2017.

[81] S. A. Fayazi and A. Vahidi, “Mixed-integer linear programming for optimal scheduling of autonomous vehicle intersection crossing,” *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 3, pp. 287–299, 2018.

[82] M. L. Cummings, J. P. How, A. Whitten, and O. Toupet, “The impact of human–automation collaboration in decentralized multiple unmanned vehicle control,” *Proceedings of the IEEE*, vol. 100, no. 3, pp. 660–671, 2011.

[83] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, “Human interaction with robot swarms: A survey,” *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 9–26, 2015.

[84] G. A. Miller, “The magical number seven, plus or minus two: Some limits on our capacity for processing information.,” *Psychological review*, vol. 63, no. 2, p. 81, 1956.

[85] N. Cowan, “The magical mystery four: How is working memory capacity limited, and why?” *Current directions in psychological science*, vol. 19, no. 1, pp. 51–57, 2010.

[86] N. Carr, *The shallows: What the Internet is doing to our brains*. WW Norton & Company, 2011.

[87] J. W. Crandall, M. L. Cummings, M. Della Penna, and P. M. De Jong, “Computing the effects of operator attention allocation in human control of multiple robots,” *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 41, no. 3, pp. 385–397, 2010.

[88] S.-Y. Chien, Y.-L. Lin, P.-J. Lee, S. Han, M. Lewis, and K. Sycara, “Attention allocation for human multi-robot control: Cognitive analysis based on behavior data and hidden states,” *International Journal of Human-Computer Studies*, vol. 117, pp. 30–44, 2018.

[89] J. W. Crandall, M. A. Goodrich, C. W. Nielsen, and D. R. Olsen Jr, “Validating human–robot interaction schemes in multitasking environments,” 2005.

[90] C. C. Murray and W. Park, "Incorporating human factor considerations in unmanned aerial vehicle routing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 4, pp. 860–874, 2012.

[91] D. W. Clarke, C. Mohtadi, and P. Tuffs, "Generalized predictive control—part ii extensions and interpretations," *Automatica*, vol. 23, no. 2, pp. 149–160, 1987.

[92] ——, "Generalized predictive control—part ii extensions and interpretations," *Automatica*, vol. 23, no. 2, pp. 149–160, 1987.

[93] K.-D. Kim and P. R. Kumar, "An mpc-based approach to provable system-wide safety and liveness of autonomous ground traffic," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3341–3356, 2014.

[94] X. Qian, J. Gregoire, A. De La Fortelle, and F. Moutarde, "Decentralized model predictive control for smooth coordination of automated vehicles at intersection," in *2015 European Control Conference (ECC)*, IEEE, 2015, pp. 3452–3458.

[95] Y. Nie, L. T. Biegler, and J. M. Wassick, "Integrated scheduling and dynamic optimization of batch processes using state equipment networks," *AIChE Journal*, vol. 58, no. 11, pp. 3416–3432, 2012.

[96] G. Folland, *Real Analysis*. New York, NY: Wiley and Sons, 1984, https://trid.trb.org/view/365890.

[97] M. Hirsch, S. Smale, and R. Devaney, *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. San Deigo, CA: Academic Press, 2004.

[98] Z.-P. Jiang and T.-F. Liu, "A survey of recent results in quantized and event-based nonlinear control," *International Journal of Automation and Computing*, vol. 12, no. 5, pp. 455–466, 2015.

[99] H. Yu and P. J. Antsaklis, "Event-triggered output feedback control for networked control systems using passivity: Achieving l2 stability in the presence of communication delays and signal quantization," *Automatica*, vol. 49, no. 1, pp. 30–38, 2013.

[100] W. H. Heemels, M. Donkers, and A. R. Teel, "Periodic event-triggered control for linear systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 4, pp. 847–861, 2012.

[101] T. Liu and Z.-P. Jiang, "Event-based control of nonlinear systems with partial state and output feedback," *Automatica*, vol. 53, pp. 10–22, 2015.

[102] N. Marchand, S. Durand, and J. F. G. Castellanos, "A general formula for event-based stabilization of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 58, no. 5, pp. 1332–1337, 2012.

[103] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the flexray communication protocol," *Real-time systems*, vol. 39, no. 1-3, pp. 205–235, 2008.

[104] J. K. Karlof, *Integer Programming: Theory and Practice*. CRC Press, 2006, ISBN: 978-0849319143.

[105] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover, 1998, ISBN: 978-0486402581.

[106] K. J. Astrom and B. M. Bernhardsson, "Comparison of Riemann and Lebesgue sampling for first order stochastic systems," in *Proceedings of the 41th IEEE Conference on Decision and Control*, 2002, pp. 2011–2016.

[107] D. Nesic, A. Teel, and D. Carnevale, "Explicit computation of the sampling period in emulation of controllers for nonlinear sampled-data systems," *IEEE Transactions on Automatic Control*, vol. 54, no. 3, pp. 619–624, 2009.

[108] L. George, N. Rivierre, and M. Spuri, "Preemptive and non-preemptive real-time uniprocessor scheduling," *HAL Preprints INRIA-00073732*, 2016.

[109] F. Zhang, Z. Shi, and S. Mukhopadhyay, "Robustness analysis for battery-supported cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 3, p. 69, 2013.

[110] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, IEEE, 1991, pp. 129–139.

[111] L. Wang, *Model Predictive Control System Design and Implementation using MATLAB®*. London, UK: Springer-Verlag London Limited, 2009.

[112] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[113] S. Sethi P. and G. L. Thompson, *Optimal Control Theory, Applications to Economics and Management Science, Second Edition*. Norwell, MA: Kluwer Academic Publishers, 2000.

[114] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[115] D. Jakovetić, J. Xavier, and J. M. Moura, "Fast distributed gradient methods," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1131–1146, 2014.

[116] S. Gong and L. Du, “Optimal location of advance warning for mandatory lane change near a two-lane highway off-ramp,” *Transportation research part B: methodological*, vol. 84, pp. 1–30, 2016.

VITA

Ningshi Yao was born and spent her early life in Chengdu, Sichuan Province, China. She attended Zhejiang University for her undergraduate study in Hangzhou, Zhejiang Province, China. She graduated with a B.S. in Control Science and Engineering. Following the graduation, she joined Georgia Tech to pursue her PhD in Electrical and Computer Engineering.