Automatica 122 (2020) 109219

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Contention-resolving model predictive control for coupled control systems with a shared resource[☆]

Ningshi Yao[a], Michael Malisoff[b], Fumin Zhang[a,*]

[a] School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30308, USA
[b] Department of Mathematics, Louisiana State University, Baton Rouge, LA, 70803, USA

ARTICLE INFO

Article history:
Received 29 April 2019
Received in revised form 28 March 2020
Accepted 17 July 2020
Available online 21 August 2020

Keywords:
Model predictive control
Mixed integer optimization
Event trigger
Networked control system

ABSTRACT

Priority-based scheduling strategies are often used to resolve contentions in resource constrained control systems. Such scheduling strategies inevitably introduce time delays into controls and may degrade the performance of control systems. Considering the coupling between priority assignment and control, this paper presents a 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 novel contention-resolving model predictive control method to dynamically assign priorities and compute an optimal control. The priority assignment can be determined using a sample-based approach without excessive demand on computing resources, and optimal controls can be computed iteratively following the order of the assigned priorities. We apply the proposed contention-resolving model predictive control to co-design scheduling and controls in networked control systems. We present simulation results to show the effectiveness of our proposed method.

1. Introduction

In modern industry, shared resources are widely used as 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 NCS), swarming robots and smart grids. For NCS, 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; see Hespanha, Naghshtabrizi, and Xu (2007) and Walsh, Hong, and Bushnell (2002). 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 (Shi, Yao, & Zhang, 2017).

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 (Sha et al., 2004; Zhang, Szwaykowska, Wolf, & Mooney, 2008). However, in many applications, a choice of the scheduling policy may affect performance significantly (Wang, Shi, Zhang, & Wang, 2015). For example, well-known scheduling policies, such as rate monotonic scheduling (or RMS) and earliest deadline first (or EDF) algorithms introduced in Liu and Layland (1973), 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 (Yao, Malisoff, & Zhang, 2017). The first-come-first-serve (FCFS) scheduling mechanism has been used to guarantee fairness; see Lee and Park (2012), Malikopoulos, Cassandras, and Zhang (2018) and Zhang, Malikopoulos, and Cassandras (2016).

☆ The research work is partially supported by 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. The material in this paper was partially presented at: the 2017 American Control Conference, May 24–26, 2017, Seattle, WA, USA. the 2nd IEEE Conference on Control Technology and Applications, August 21–24, 2018, Copenhagen, Denmark. This paper was recommended for publication in revised form by Associate Editor M. Lagoa under the direction of Editor Sophie Tarbouriech.

* Corresponding author.

E-mail addresses: nyao6@gatech.edu (N. Yao), malisoff@lsu.edu (M. Malisoff), fumin@gatech.edu (F. Zhang).

https://doi.org/10.1016/j.automatica.2020.109219
0005-1098/© 2020 Elsevier Ltd. All rights reserved.

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 (Engell & Harjunkoski, 2012). 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 Chen, Yao, and Qiu (2019), Farnam and Esfanjani (2014), Gao, Chen, and Lam (2008), Peng and Yang (2013), Shi and Zhang (2017) and Zhou, Du, and Chen (2012). 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 (Gaid, Cela, & Hamam, 2006, 2009; Mazumder, Acharya, & Tahir, 2009; Roy, Zhang, Chang, Goswami, & Chakraborty, 2016; Yao, Chang, & Yen, 2005) 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 iteratively, using a predefined cost function (which usually considers the overall performance and efficiency) while receding a finite optimization time horizon (Mayne, Rawlings, Rao, & Scokaert, 2000; Rawlings & Mayne, 2009). 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 (Baskar, De Schutter, & Hellendoorn, 2008; Bellemans, De Schutter, & De Moor, 2006; Frejo & Camacho, 2012; Lješnjanin, Quevedo, & Nešić, 2014; Negenborn, De Schutter, & Hellendoorn, 2008; Shi, Bart De, Yugeng, & Hans, 2011), energy storage systems (Afram & Janabi-Sharifi, 2014; Touretzky & Baldea, 2014; Zhao, Lu, Yan, & Wang, 2015) and chemical processes (Chu & You, 2014). 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 paper, 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 (Gaid et al., 2006; Liu, Sun, & Zhao, 2013; Liu, Xia, Chen, Rees, & Hu, 2007). 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 our previous work (Shi & Zhang, 2017) and establish analytical timing models for both preemptive 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 tree will contain 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. Different from our previous work (Yao et al., 2017), the cost function can handle cases where a control system's access to the shared resource is delayed multiple times.
3. *Co-design algorithm*. We provide a significant modification of the A-star algorithm from Hart, Nilsson, and Raphael (1968) 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 (Fayazi & Vahidi, 2017; Yan, Dridi, & El Moudni, 2013), 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.
4. *Practical application case study*. We apply contention-resolving MPC to networked control systems with shared communication media, under both preemptive and non-preemptive scheduling disciplines. 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 EDF or RMS 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.

The rest of this paper is organized as follows. In Section 2, we introduce the contention-resolving MPC formulation. In Section 3, we present the analytical timing models. In Section 4, we present the path planning problem converted from the priority assignment and MPC design problem. In Section 5, we present simulation results for applying contention-resolving MPC to NCS. Section 6 summarizes our work.

2. Problem formulation

The proposed contention-resolving MPC is a general theoretical framework to address resource allocation for coupled control systems. Consider N control systems that must access a shared resource. Assume that the ith control system for $i = 1, 2, \ldots, N$ is modeled in the form

$$\dot{x}_i(t) = f_i(x_i(t), u_i(t)), \quad y_i(t) = g_i(x_i(t)) \tag{1}$$

where x_i, u_i, and y_i represent the state vector, control, and output, respectively. Each control system is viewed as a *customer* that must be served to access the shared resource. Here and in the sequel, we make the following assumption about the shared resource and our dynamics, where we use measurability and essential boundedness in the Lebesgue measure sense of Folland (1984):

Assumption 2.1. The function f_i for $i = 1, \ldots, N$ in (1) is such that this holds for each i: For each measurable essentially bounded function u_i and each initial state $\bar{x}_i$ and each $T > 0$, the initial value problem for the dynamics f_i and the initial condition $x_i(0) = \bar{x}_i$ has a unique solution on $[0, T]$.

The preceding assumption is satisfied under standard Lipschitzness conditions, e.g., from Hirsch, Smale, and Devaney (2004, Chapter 7).

Assumption 2.2. At any given time, only one customer can occupy the shared resource.

Assumption 2.2 is valid in many real world applications. In the automotive industry, the vehicle communication buses such as the control area network (or CAN) (Robert Bosch GmbH, 1991) and FlexRay (Pop, Pop, Eles, Peng, & Andrei, 2008) 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.

The ith customer has a sequence of tasks, denoted by $\{\tau_i[1], \tau_i[2], \ldots, \tau_i[k], \ldots\}$, 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 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 amount of time for which the task $\tau_i[k]$ needs to occupy the resource is denoted by $C_i[k]$. The completion time instant when task $\tau_i[k]$ finishes the occupation of the shared resource is denoted as $\gamma_i[k]$. 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 2.3. 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, 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 $(\alpha_i, C_i, T_i)[k]$.

2.1. Priority-based scheduling

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

$$\gamma_i[k] = \alpha_i[k] + C_i[k]. \tag{2}$$

When multiple customers request the shared resource at the same time, a contention occurs and Eq. (2) will not hold. An example of three systems sharing one resource is shown in Fig. 1(a). 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, $\mathcal{P}(\{1, \ldots, N\})$ denotes the set of all permutations of $\{1, \ldots, N\}$.

Definition 2.1. A priority assignment is a tuple $\mathbf{P}(t) = (p_1(t), \ldots, p_i(t), \ldots, p_N(t)) \in \mathcal{P}(\{1, \ldots, N\})$, where $p_i(t)$ is the priority assigned to customer i at time t and such that for each i and j in $\{1, \ldots, N\}$, 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 2.4. 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 (Conway, Maxwell, & Miller, 2003). Based on Definition 2.1, each task has a unique priority number. Therefore, there exist no ties among the priority assignments when a contention occurs.

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. $\gamma_i[k] = \alpha_i[k]+C_i[k]+\delta_i[k]$.

Assumption 2.5. We assume that $\alpha_i[k]+C_i[k]+\delta_i[k] \leq \alpha_i[k+1]$ for all i and k.

The previous assumption can be interpreted to mean that no customer requests access to the shared resource until their previous task has completed using the shared resource. This assumption guarantees the schedulability of the system, meaning, all tasks are able to be completed before or at their deadlines.

Example 2.1. 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 Fig. 1(a). 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 Section 3, we will present a timing model which can accurately compute the time delays given a specific priority assignment.

2.2. Formulation of model predictive control

We formulate and solve a contention-resolving model predictive control problem to compute optimal priority assignments $\mathbf{P}^*(t) = (p_1^*(t), \ldots, p_N^*(t))$ and an optimal control command $\mathbf{u}^*(t) = (u_1^*(t), \ldots, u_N^*(t))$ 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 $\mathbf{x}(t_0) = (x_1(t_0), \ldots, x_N(t_0))$, initial controls $\mathbf{u}(t_0) = (u_1(t_0), \ldots, u_N(t_0))$, 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

$$\min_{\mathbf{P}(t),\mathbf{u}(t)} \sum_{i=1}^{N} V_i\big(x_i(t, \mathbf{P}(t_0 \sim t), u_i(t_0 \sim t)), u_i(t, \mathbf{P}(t_0 \sim t))\big) \tag{3}$$

over all $\mathbf{u}$ and $\mathbf{P}$ where the cost functions V_i for $i = 1, 2, \ldots, N$ 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 3.5 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

$$\begin{aligned} &V_i\big(x_i(t, \mathbf{P}(t_0 \sim t), u_i(t_0 \sim t)), u_i(t, \mathbf{P}(t_0 \sim t))\big) = \\ &\frac{1}{2}\int_{t_0}^{t_f} \Big(|x_i(t, \mathbf{P}(t_0 \sim t), u_i(t_0 \sim t)) - \bar{x}_i(t)|_{Q_i}^2 \\ &+ \ |u_i(t, \mathbf{P}(t_0 \sim t)) - \bar{u}_i(t)|_{R_i}^2\Big)\, \mathrm{d}t \\ &+ \ \rho|x_i(t_f, \mathbf{P}(t_0 \sim t_f), u_i(t_0 \sim t_f)) - \bar{x}_i(t_f)|_{K_i}^2, \end{aligned} \tag{4}$$

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 (Shi & Zhang, 2017).

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 3.5.

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 (Karlof, 2006). Mixed integer programming problems are usually solved by two categories of optimization methods. The first category is combinatorial optimization (Papadimitriou & Steiglitz, 1998), 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 (Lawler & Wood, 1966). 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 2.6. 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 (Astrom & Bernhardsson, 2002; Nesic, Teel, & Carnevale, 2009). 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

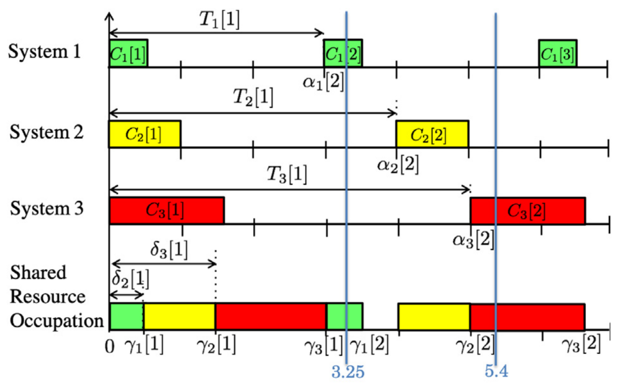
$$u_i(t) = u_i[k] \text{ for all } t \in [\gamma_i[k], \gamma_i[k+1]) \text{ and } k, \tag{5}$$

which defines the control u_i at all times when customer i can access the shared resource. As mentioned in Section 2.1, 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 2.2 and compute optimal priorities and optimal controls to compensate for the performance degradation caused by contentions and delays.

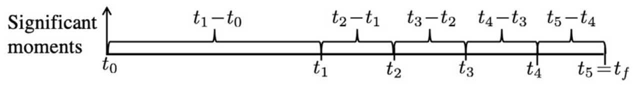
3. Significant moment analysis and timing model

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 (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 (George, Rivierre, & Spuri, 2016), priority-based scheduling can be classified into two categories, preemptive and non-preemptive scheduling. Therefore, in this paper, we model the



(a) Scheduled behavior.



(b) Significant moments t_w.

Fig. 1. Illustration of scheduling three systems. The upper three sub-figures in Fig. 1(a) show the task request times when contentions are not considered. The bottom sub-figure in Fig. 1(a) shows the resource occupation time after priorities are assigned to resolve the contention that occurs at time 0.

scheduling behavior of both preemptive and non-preemptive real-time systems.

In preemptive scheduling, 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 once the higher prioritized task is completed. In non-preemptive scheduling, 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 (Baruah & Chakraborty, 2006). In our previous work Shi and Zhang (2017) and Zhang, Shi, and Mukhopadhyay (2013), 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 both preemptive and non-preemptive scheduling.

3.1. 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 Zhang et al. (2013), where a task is a request for access to the shared resource:

Definition 3.1. The vector $D(t) = (d_1(t), \dots, d_i(t), \dots, d_N(t))$ 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.,

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

Definition 3.2. The vector $R(t) = (r_1(t), \dots, r_i(t), \dots, r_N(t))$ 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.,

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

Definition 3.3. The vector $O(t) = (o_1(t), \dots, o_i(t), \dots, o_N(t))$ 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.,

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

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

Definition 3.4. 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 $Z(t) = (D(t), R(t), O(t), \mathrm{ID}(t))$.

We use the example in Fig. 1 to further explain D, R and O.

Example 3.1. 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 3.1, 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 3.2, 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 $O(5.4) = \big(0.5, 1.0, 0.4\big)$.

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

Definition 3.5. 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

$$C_i(t) = C_i[k],\ T_i(t) = T_i[k] \text{ and } P_i(t) = P_i[k] \tag{6}$$

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.

3.2. Delay prediction using timing model

We use this notation to represent the timing model:

$$Z(t) = \mathbb{H}\Big(t; Z(t_0), \mathbf{S}, \mathbf{P}(t_0 \sim t)\Big), \tag{7}$$

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.

3.3. Timing model for preemptive network

The work (Shi & Zhang, 2013) established a dynamic timing model for the preemptive scheduling discipline. This section consists of a brief review of the timing model from Shi and Zhang (2013). 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

$$t_{w+1} - t_w = \min\left\{d_1(t_w), \ldots, d_N(t_w), t_f - t_w\right\}. \tag{8}$$

The following example illustrates the preceding notation:

Example 3.2. Consider the example in Fig. 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 length $t_1 - t_0 = \min\{d_1(0), d_2(0), d_3(0), 7 - 0\} = \min\{3, 4, 5, 7\} = 3$ 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 a 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

$$d_n(t_w) = d_n(t_w^-),\ r_n(t_w) = r_n(t_w^-),\ o_n(t_w) = o_n(t_w^-). \tag{9}$$

(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

$$d_i(t_w) = T_i(t_w),\ r_i(t_w) = C_i(t_w),\ o_i(t_w) = 0. \tag{10}$$

According to Eqs. (9) and (10), the evolution rules at the times t_w can be summarized as:

$$\begin{aligned} d_i(t_w) &= d_i(t_w^-) + \left(1 - \operatorname{sgn}(d_i(t_w^-))\right) T_i(t_w), \\ r_i(t_w) &= \operatorname{sgn}(d_i(t_w^-) + r_i(t_w^-))\, r_i(t_w^-) \\ &\quad + \left(1 - \operatorname{sgn}(r_i(t_w^-))\right)\left(1 - \operatorname{sgn}(d_i(t_w^-))\right) C_i(t_w), \\ o_i(t_w) &= o_i(t_w^-)\operatorname{sgn}(d_i(t_w^-)) \\ &\quad + o_i(t_w^-)\operatorname{sgn}(r_i(t_w^-))\left(1 - \operatorname{sgn}(d_i(t_w^-))\right), \end{aligned} \tag{11}$$

where sgn is defined by $\operatorname{sgn}(q) = 1$ if $q > 0$ and $\operatorname{sgn}(q) = 0$ 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

$$d_i(t_w + \Delta t) = d_i(t_w) - \Delta t. \tag{12}$$

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

$$\begin{aligned} &r_i(t_w + \Delta t) = \\ &\max\left\{0, r_i(t_w) - \max\left\{0, \Delta t - \sum_{q \in \mathrm{HP}_i(t_w)} r_q(t_w)\right\}\right\}. \end{aligned} \tag{13}$$

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

$$o_i(t_w + \Delta t) = o_i(t_w). \tag{14}$$

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

$$\begin{aligned} &o_i(t_w + \Delta t) = \\ &o_i(t_w) + \operatorname{sgn}(r_i(t_w))\ \min\left\{\Delta t, r_i(t_w) + \sum_{q \in \mathrm{HP}_i(t_w)} r_i(t_w)\right\} \end{aligned} \tag{15}$$

Combining all of the evolution rules in (11)–(15) leads to the timing model (7) of preemptive scheduling.

3.4. Timing model for non-preemptive network

The work (Shi & Zhang, 2017) 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. $1 - \operatorname{sgn}(\mathrm{ID}(t_w)) = 1$, then it is the same case as preemptive scheduling where $t_{w+1} - t_w = \min\left\{d_1(t_w), \ldots, d_N(t_w), t_f - t_w\right\}$.

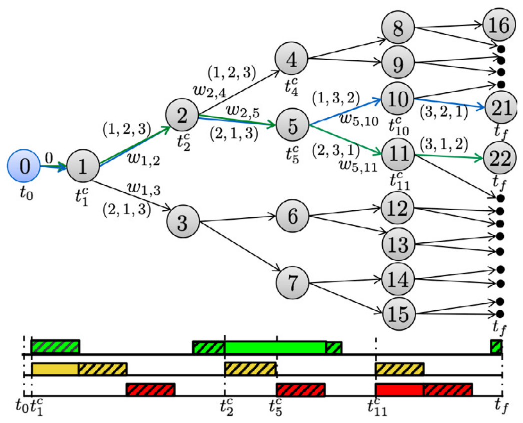


Fig. 2. Decision tree to solve the co-design problem for preemptive scheduling within a finite time window. The blue circle represents the root v_0, and gray circles and dots represent internal leaves. The decision tree is expanded in the direction of the arrows, which represent branches. The integers in brackets represent the priorities. The 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. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

If the shared resource is occupied by task $\mathrm{ID}(t_w)$ at time t_w, i.e. $\mathrm{sgn}(\mathrm{ID}(t_w)) = 1$, then we will require that $t_{w+1} - t_w \le \min\{d_1(t_w), \ldots, d_N(t_w), t_f - t_w\}$, and in addition, $t_{w+1} - t_w$ should be less than or equal to $r_{\mathrm{ID}}(t_w)$ so that the occupation completion time $t_w + r_{\mathrm{ID}}(t_w) \ge t_{w+1}$. Here $r_{\mathrm{ID}}(t)$ is a simplified notation for the remaining time $r_{\mathrm{ID}(t)}(t)$ of timing state variable ID at any t. Summarizing the above two cases, we have

$$\begin{aligned} &t_{w+1} - t_w = \qquad (16)\\ &\mathrm{sgn}(\mathrm{ID}(t_w)) \min\{r_{\mathrm{ID}}(t_w), d_1(t_w), \ldots, d_N(t_w), t_f - t_w\}\\ &+ (1 - \mathrm{sgn}(\mathrm{ID}(t_w))) \min\{d_1(t_w), \ldots, d_N(t_w), t_f - t_w\}\end{aligned}$$

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 (11). 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

$$\begin{aligned}\mathrm{ID}(t_w) &= \mathrm{ID}(t_w^-)\,\mathrm{sgn}(r_{\mathrm{ID}}(t_w^-))\\ &\quad + \left(\operatorname*{argmin}_{i\in\Lambda(t_w)} p_i(t_w)\right)\left(1 - \mathrm{sgn}(r_{\mathrm{ID}}(t_w^-))\right) \qquad (17)\end{aligned}$$

when $\Lambda(t_w) \ne \emptyset$, where $\Lambda(t_w) = \{i \in \{1, \ldots, N\} : r_i(t_w) = C_i(t_w)\}$ 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 $t_{w+1} - t_w \le r_{\mathrm{ID}}(t_w)$. If $\mathrm{ID}(t_w) \ne 0$, the evolution rules for control system $\mathrm{ID}(t_w)$ are

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

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

$$\begin{aligned}&d_i(t_w + \Delta t) = d_i(t_w) - \Delta t,\ \ r_i(t_w + \Delta t) = r_i(t_w)\\ &\text{and } o_i(t_w + \Delta t) = o_i(t_w) + \mathrm{sgn}(r_i(t_w))\Delta t. \qquad (18)\end{aligned}$$

Combining all of the evolution rules in (16)–(18) leads to the timing model (7) of non-preemptive scheduling.

3.5. 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

$$\begin{aligned}&\min_{\mathbf{P}(t),\mathbf{u}(t)} \sum_{i=1}^{N} V_i\big(x_i(t, \mathbf{P}(t_0 \sim t), u_i(t_0 \sim t)),\\ &\qquad u_i(t, \mathbf{P}(t_0 \sim t))\big); \qquad (19a)\\ &\text{s.t } Z(t) = \mathbb{H}\Big(t; Z(t_0), \mathbf{S}, \mathbf{P}(t_0 \sim t)\Big),\\ &\quad \delta_i[k] = o_i(\alpha_i[k+1]^-) - C_i[k],\\ &\quad \gamma_i[k] = \alpha_i[k] + C_i[k] + \delta_i[k] \text{ for } k = 1, \ldots, \overline{K}_i; \qquad (19b)\\ &\quad \dot{x}_i(t) = f_i(x_i(t), u_i(t)),\quad y_i(t) = g_i(x_i(t)),\\ &\quad \text{with } u_i(t) = u_i(t_0),\ t \in [\gamma_i[0], \gamma_i[1]) \text{ and}\\ &\quad u_i(t) = u_i[k] \ \text{ for all } t \in [\gamma_i[k], \gamma_i[k+1]),\\ &\qquad\quad k = 1, \ldots, \overline{K}_i; \qquad (19c)\\ &\quad u_i(t) \in \mathbb{U}_i; \qquad (19d)\\ &\quad \mathbf{P}(t) \in \mathcal{P}(\{1, \ldots, N\}); \qquad (19e)\end{aligned}$$

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. Eq. (19b) is the timing model to compute $\delta_i[k]$ which has been introduced in Sections 3.3 and 3.4. Eq. (19c) represents the system dynamics, which summarizes (1) and (5). Eq. (19d) represents the state and control constraints. Eq. (19e) dictates that the priority assignments are constrained to be in the set $\mathcal{P}(\{1, \ldots, N\})$ of all permutations of $\{1, 2, \ldots, N\}$ following Definition 2.1.

4. Solutions to the contention-resolving MPC

In this section, we propose a novel method to solve the mixed integer programming problem associated with contention-resolving MPC formulated in Section 2.2 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 Fig. 2. 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.

4.1. 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 4.1. *In preemptive scheduling, a contention starts at time t if and only if the following three conditions hold:*

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

Proof. Based on Definition 3.2, 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 4.2. *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:*

$$\sum_{i=1}^{N} [1 - \mathsf{sgn}(C_i(t_w) - r_i(t_w))] \geq 2,\ r_{\mathrm{ID}}(t_w^-) = 0 \tag{20}$$

where t_w is a significant moment computed by Eq. (16).

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 (20) holds. For sufficiency, if the two conditions (20) 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. □

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. Fig. 2 shows an example of decision tree. In the decision tree, each leaf represents a contention time satisfying Propositions 4.1 or 4.2. 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 4.1. 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 tree can be fully constructed.

4.2. 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 $\Lambda(t_l^c) = \{i \in \{1, \ldots, N\} : r_i(t_l^c) > 0\}$ for preemptive scheduling and $\Lambda(t_l^c) = \{i \in \{1, \ldots, N\} : r_i(t_l^c) = C_i(t_l^c)\}$ for non-preemptive scheduling. Also, M is the number of elements of $\Lambda(t_l^c)$. Let $\mathbf{P}_m$ denote the mth permutation in $\mathcal{P}(\{1, \ldots, M\})$, so $m \in \{1, 2, \ldots, M!\}$. For leaf l, we generate $M!$ branches from it. Each branch corresponds to a unique choice of the priority assignment in $\mathcal{P}(\{1, \ldots, M\})$. The mth 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 Fig. 2. 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.

4.3. Branch cost

After constructing the decision tree, 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 $t_l^c \leq \gamma_i[k] \leq t_j^c$,

$$Z(t) = \mathbb{H}\Big(t; Z(t_l^c), \mathbf{S}, \mathbf{P}_m\Big),$$
$$\text{and } \gamma_i[k] = \alpha_i[k] + o_i(\alpha_i[k+1]^-) \tag{21}$$

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 (19b). Then the branch cost $w_{l,j}$ is defined as

$$w_{l,j} = \sum_{i=1}^{N} w_{l,j}^i \tag{22}$$

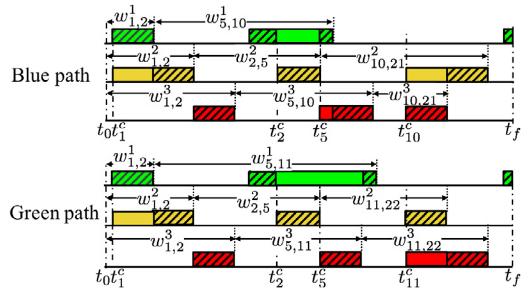


Fig. 3. Illustration of branch cost along a path. The ending time of the colored rectangles with diagonal lines represents the task completion time γ_i. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

where $w^i_{l,j}$ 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 $\gamma_i[k+1] \in (t^c_l, t^c_j]$, let $\underline{k}_i$ be the smallest index k satisfying $\gamma_i[k+1] > t^c_l$ and $\bar{k}_i$ be the largest index k satisfying $\gamma_i[k+1] \le t^c_j$. Then we set

$$w^i_{l,j} = \sum_{k=\underline{k}_i}^{\bar{k}_i} \min_{u_i[k]} V_i(u_i[k]; x_i(\alpha_i[k]), \gamma_i[k], \gamma_i[k+1]) \tag{23}$$

subject to (19c) and (19d).

The meaning of (23) 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 $(\bar{k}_i + 1)$st task will be included by the branches following the branch (l, j).

Fig. 3 shows an illustration of the defined branch cost for the blue path and green path in Fig. 2. 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 4.2. 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 4.3. The optimal control design is embedded in the branch cost calculation. To calculate $w^i_{n,j}$ in (23), we need to solve the optimization problem (23) by optimizing the control law $u_i(t)$. Since the priority along one branch is already known, we can use the MPC design methods from Rawlings and Mayne (2009) and Wang (2009) to solve (23) and compute $u^*_i(t)$. After solving (23) 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 (19a) 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 (19a). Among all the paths, the lowest cost path can be found by path planning algorithms (LaValle, 2006) 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 4.4, 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 (Hart et al., 1968) 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 4.6.

4.4. 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

$$C_f(v_l) = C_g(v_l) + C_h(v_l) \tag{24}$$

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 $\hat{C}_h(v_l) \le C_h(v_l)$ for all v_l, so the estimated cost $\hat{C}_f(v_l) = C_g(v_l) + \hat{C}_h(v_l)$ is equal 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

$$\hat{C}_h(v_l) = \min_{\mathbf{u}^h(t)} \sum_{i=1}^{N} V_i(u^h_i(t); x_i(t^c_l), u_i(t^c_l), t^c_l, t_f), \tag{25}$$

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

where $\mathbf{u}^h(t) = (u^h_1(t), \ldots, u^h_i(t), \ldots, u^h_N(t))$ is Lebesgue measurable and essentially bounded (as defined for instance in Folland, 1984), and t^c_l is the contention time instant corresponding to leaf v_l. Notice that the above optimization problem does not have the contention constraints from (19b).

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

Algorithm 1 Main Program

```
1: Data: $t_0$, $t_f$, $\lambda_i$ for $1 \le i \le N$, $\mathbf{x}(t_0)$, $\mathbf{u}(t_0)$, $Z(t_0)$
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 $t_l^c \le t_f$ 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 mth permutation $\mathbf{P}_m \in \mathcal{P}(\{1,\dots,M\})$ do
14:            $(v_{j+m}, t_{j+m}^c, w_{l,j+m})$=Expand($v_l$, $\mathbf{P}_m$, $t_l^c$);
15:            Add $v_{j+m}$ into frontier list and generated set;
16:            $C_g(v_{j+m}) = C_g(v_l) + w_{l,j+m}$;
17:            Solve (25) to obtain $\hat{C}_h(v_{j+m})$;
18:            $\hat{C}_f(v_{j+m}) = C_g(v_{j+m}) + \hat{C}_h(v_{j+m})$;
19:            $PT(v_{j+m}) = l$;
20:        Remove $v_l$ from frontier list;
```

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.

4.5. Contention-resolving MPC algorithm

Algorithms 1–3 present the pseudocode for our proposed algorithm based on the A-star algorithm to solve the optimization problem (3). Algorithm 1 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 1, 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 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 2 backtracks the path from the selected terminal leaf to v_0 when case (1) is satisfied in the search algorithm.

Algorithm 2 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 3 Expand

```
1: Data: $v_l$, $\mathbf{P}_m$, $t$
2: Find the next contention time under priority $\mathbf{P}_m$, and denote this contention time as $t_{j+m}^c$;
3: Solve the optimization formulated by (23) to obtain $u_i^*(t)$ and compute $w_{l,j+m}^i$ for each $i = 1,\dots,N$;
4: Compute $w_{l,j+m}$ using (22);
5: return $v_{j+m}$, $t_{j+m}^c$, $w_{l,j+m}$;
```

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 2 returns the optimal priority assignment $\mathbf{P}^*(t)$ for all $t \in [t_0, t_f]$ to the main program in Algorithm 1.

Algorithm 3 expands the selected leaf from the *frontier list* when case (2) is satisfied in the search algorithm. It utilizes Propositions 4.1 or 4.2 to determine the next contention time after a contention time t. Then it solves the optimization problem (23) to obtain the optimal control $u_i^*(t)$ and compute the branch cost $w_{l,j+m}$. Algorithm 3 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 1.

Fig. 4 is an illustration of the subtree constructed by our algorithm described above, using the same example as Fig. 2. Compared with the entire decision tree in Fig. 2, 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, reducing the computational workload for generating the entire tree which has totally 30 branches as shown in Fig. 2.

4.6. Proof of optimality

In this subsection, we prove that our algorithm finds the optimal solutions $\mathbf{P}^*(t)$ and $\mathbf{u}^*(t)$ which minimize (3). To this end, first note that since there are no state constraints in our optimization problems for each fixed choice of the priority assignments (except for the initial state and terminal state constraints), all of the optimization problems we solve have solutions, and this ensures recursive feasibility. Next, we show that the heuristic cost $\hat{C}_h(v_l)$ defined in Section 4.4 satisfies the requirements for the A-star algorithm.

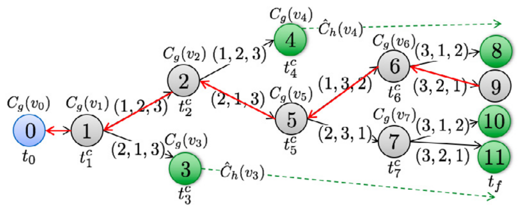


Fig. 4. Illustration of the subtree constructed by the proposed search algorithm. The blue circle represents the root v_0 and the red circle represents the terminal leaf. Green circles represent leaves in the *frontier list*. Solid black arrows represent branches generated by the algorithm and dashed green arrows represent the estimate cost $\hat{C}_h(v_l)$. The red arrows represent the path with lowest cost. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Proposition 4.3. *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 (25). The actual future cost $C_h(v_l)$ is obtained by solving the optimization problem defined by (3) given the initial condition $\mathbf{x}(t_l^c)$. Comparing (25) and (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) is constrained to be piecewise constant function that depends on the priorities, while $\mathbf{u}^h(t)$ in (25) 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) must be feasible but may not be an optimal solution for (25). Hence, $\hat{C}_h(v_l)$ is less than or equal to $C_h(v_l)$ for all v_l. □

Theorem 4.1. *Algorithm 1 finds an optimal solution $\mathbf{P}^*(t)$ and $\mathbf{u}^*(t)$ for the optimization problem (19a).*

Proof. From Hart et al. (1968, 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 4.3, the theorem follows.

Remark 4.4. Since $\mathcal{P}(\{1, \ldots, M\})$ contains all possible priority assignments, it also includes the priorities following RMS, EDF and FCFS rules. Therefore, the priorities assigned by the RMS, EDF or FCFS 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.

5. Simulation results

This section presents simulation results obtained by the proposed method implemented in MATLAB. We simulate an NCS consisting of four scalar systems. The first three are linear systems $\dot{x}_i(t) = a_i x_i(t) + u_i(t), i = 1, 2, 3$ with parameters $(a_1, a_2, a_3) = \left(1, \frac{6}{5}, \frac{4}{3}\right)$. The fourth system is nonlinear $\dot{x}_4(t) = x_4^2(t) + u_4(t)$. 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 $i = 1, \ldots, 4$. The output of each plant is the state $x_i(t)$. The time horizon $[t_0, t_f]$ for the simulation is from 0 to 6 s. The cost function is $V_i(x_i(0), 0) = \frac{1}{2}\int_0^6 \left\{x_i^2(t) + 0.0001u_i^2(t)\right\}dt + x_i^2(6)$ and the reference signal is $\lambda_i(t) = 0$ for all i and $t \in [0, 6]$. The task parameters are $(C_1[k], C_2[k], C_3[k], C_4[k]) = (0.3, 0.3, 0.2, 0.2)$ and $(T_1[k], T_2[k], T_3[k], T_4[k]) = (1, 1.25, 1.5, 2)$ in seconds. The four plants are all stabilizable from the initial condition if no contention exists.

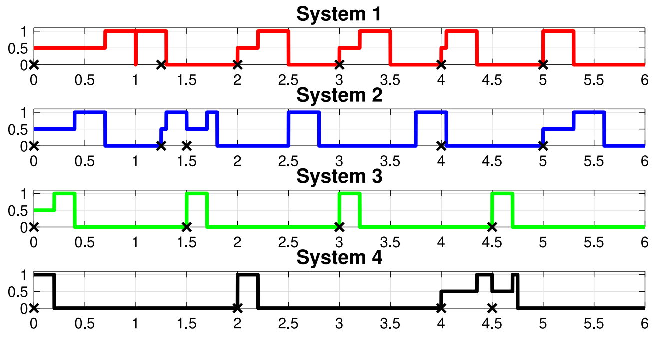


Fig. 5. Communication network occupation of scheduling four scalar systems under preemptive scheduling. The occupation value 1 means the system is occupying the network, 0 means the system does not require access to the network, and 0.5 means the system access request is delayed by contention. Black crosses mark times when a contention occurs.

5.1. 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 Fig. 5. 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 $t \in [0, 1.25)$ s, 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 s.

5.2. 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 Fig. 6. Five contentions occur in time window [0, 6], represented by the crosses in the figure. Similar to the preemptive scheduling case, for $t \in [0, 1)$ s, 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 s. 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 fourth and fifth contentions at times 4 and 5 s, the contention-resolving MPC assigns system 1 a higher priority, which is different from the first three contentions.

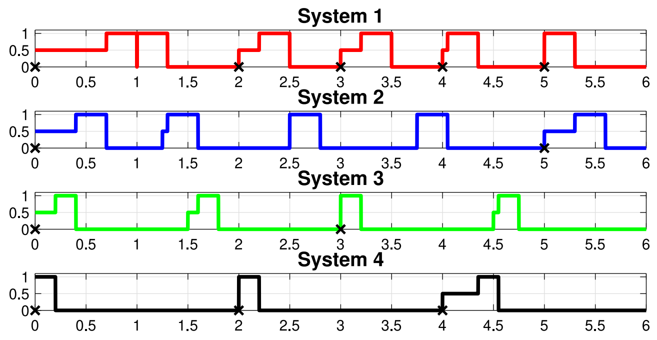


Fig. 6. Communication network occupation of scheduling four scalar systems under non-preemptive scheduling discipline. Black crosses mark times when a contention occurs.

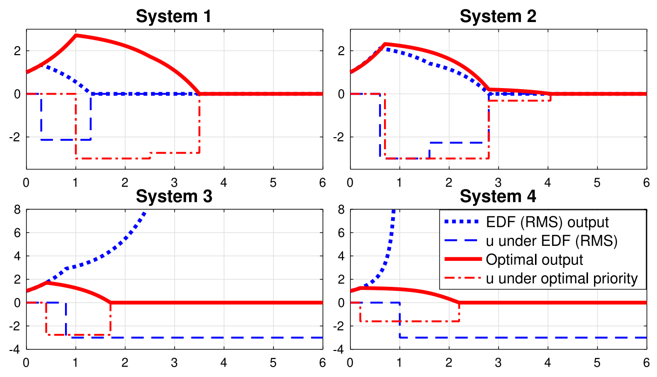


Fig. 7. Outputs of four scalar systems. The red solid lines show the output under optimal priority assignment, and the blue solid lines show the outputs under EDF. The outputs under RMS are the same as EDF. The dashed lines show the control u_i computed by the MPC in each case.

5.3. Control performance

The outputs of the four scalar systems under preemptive scheduling are presented in Fig. 7. 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, the nonlinear system 4 with the highest priority, and the most unstable linear system 3 with second highest priority from 0 to 1 s. The outputs of the four scalar systems under non-preemptive scheduling are the same as Fig. 7, 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.

6. Conclusions

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 paper leads to new insights in scheduling and control co-design methods under contentions. We presented a novel algorithm to design optimal priority assignments, which we applied to optimal control for networked control systems, considering both preemptive and non-preemptive scheduling disciplines. We showed how our algorithm is admissible for finding an optimal priority assignment associated with an optimal control computed by MPC. The simulation results demonstrated significant improvements using our proposed method, compared with the RMS and EDF scheduling strategies. In future work, we hope to develop robust contention-resolving MPC that can quantify the effects of perturbations that are caused by unpredictable events such as package drops in communication networks. Also, we hope to apply the contention-resolving MPC algorithm to an intersection management problem.

References

Afram, Abdul, & Janabi-Sharifi, Farrokh (2014). Theory and applications of HVAC control systems–A review of model predictive control (MPC). *Building and Environment*, *72*, 343–355.

Astrom, Karl J., & Bernhardsson, Bo M. (2002). Comparison of Riemann and Lebesgue sampling for first order stochastic systems. In *Proceedings of the 41st IEEE conference on decision and control* (pp. 2011–2016). http://dx.doi.org/10.1109/CDC.2002.1184824.

Baruah, Sanjoy K., & Chakraborty, Samarjit (2006). Schedulability analysis of non-preemptive recurring real-time tasks. In *Proceedings of the 20th international parallel and distributed processing symposium*. IEEE.

Baskar, Lakshmi, De Schutter, Bart, & Hellendoorn, Hans (2008). Model-based predictive traffic control for intelligent vehicles: Dynamic speed limits and dynamic lane allocation. In *Proceedings of the IEEE intelligent vehicles symposium* (pp. 174–179). Eindhoven, Netherlands. http://dx.doi.org/10.1109/IVS.2008.4621307.

Bellemans, T., De Schutter, B., & De Moor, B. (2006). Model predictive control for ramp metering of motorway traffic: A case study. *Control Engineering Practice*, *14*(7), 757–767.

Chen, Wei, Yao, Jing, & Qiu, Li (2019). Networked stabilization of multi-input systems over shared channels with scheduling/control co-design. *Automatica*, *99*, 188–194.

Chu, Yunfei, & You, Fengqi (2014). Moving horizon approach of integrating scheduling and control for sequential batch processes. *AIChE Journal*, *60*(5), 1654–1671.

Conway, Richard W., Maxwell, William L., & Miller, Louis W. (2003). *Theory of scheduling*. Chelmsford, MA: Courier Corporation.

Engell, Sebastian, & Harjunkoski, Iiro (2012). Optimal operation: Scheduling, advanced control and their integration. *Computers and Chemical Engineering*, *47*, 121–133.

Farnam, Arash, & Esfanjani, Reza Mahboobi (2014). Improved stabilization method for networked control systems with variable transmission delays and packet dropout. *ISA Transactions*, *53*(6), 1746–1753.

Fayazi, S. Alireza, & Vahidi, Ardalan (2017). 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* (pp. 1575–1580). IEEE.

Folland, G. (1984). *Real analysis*. New York, NY: Wiley and Sons.

Frejo, José Ramón Domínguez, & Camacho, Eduardo Fernández (2012). Global versus local MPC algorithms in freeway traffic control with ramp metering and variable speed limits. *IEEE Transactions on Intelligent Transportation Systems*, *13*(4), 1556–1565.

Gaid, M. E. Mongi Ben, Cela, Arben, & Hamam, Yskandar (2006). Optimal integrated control and scheduling of networked control systems with communication constraints: application to a car suspension system. *IEEE Transactions on Control Systems Technology*, *14*(4), 776–787.

Gaid, Mohamed El Mongi Ben, Cela, Arben S., & Hamam, Yskandar (2009). Optimal real-time scheduling of control tasks with state feedback resource allocation. *IEEE Transactions on Control Systems Technology*, *17*(2), 309–326.

Gao, Huijun, Chen, Tongwen, & Lam, James (2008). A new delay system approach to network-based control. *Automatica*, *44*(1), 39–52.

George, Laurent, Rivierre, Nicolas, & Spuri, Marco (2016). Preemptive and non-preemptive real-time uniprocessor scheduling. HAL Preprints INRIA-00073732.

Hart, Peter E., Nilsson, Nils J., & Raphael, Bertram (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Hespanha, João, Naghshtabrizi, Payam, & Xu, Yonggang (2007). A survey of recent results in networked control systems. *Proceedings of the IEEE*, *95*(1), 138–162.

Hirsch, Morris, Smale, Stephen, & Devaney, Robert (2004). *Differential equations, dynamical systems, and an introduction to chaos*. San Deigo, CA: Academic Press.

Karlof, John K. (2006). *Integer programming: Theory and practice*. Boca Raton, FL: CRC Press.

LaValle, Steven M. (2006). *Planning algorithms*. Cambridge, UK: Cambridge University Press.

Lawler, Eugene L., & Wood, David E. (1966). Branch-and-bound methods: A survey. *Operations Research*, *14*(4), 699–719.

Lee, Joyoung, & Park, Byungkyu (2012). Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment. *IEEE Transactions on Intelligent Transportation Systems*, *13*(1), 81–90.

Liu, Chung Laung, & Layland, James W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, *20*(1), 46–61.

Liu, Guoping, Sun, Jian, & Zhao, Yunbo (2013). Design, analysis and real-time implementation of networked predictive control systems. *Acta Automatica Sinica*, *39*(11), 1769–1777.

Liu, Guoping, Xia, Yuanqing, Chen, Jie, Rees, David, & Hu, Wenshan (2007). Networked predictive control of systems with random network delays in both forward and feedback channels. *IEEE Transactions on Industrial Electronics*, *54*(3), 1282–1297.

Lješnjanin, Merid, Quevedo, Daniel E., & Nešić, Dragan (2014). Packetized MPC with dynamic scheduling constraints and bounded packet dropouts. *Automatica*, *50*(3), 784–797.

Malikopoulos, Andreas A., Cassandras, Christos G., & Zhang, Yue J. (2018). A decentralized energy-optimal control framework for connected automated vehicles at signal-free intersections. *Automatica*, *93*, 244–256.

Mayne, David Q., Rawlings, James B., Rao, Christopher V., & Scokaert, Pierre O. M. (2000). Constrained model predictive control: Stability and optimality. *Automatica*, *36*(6), 789–814.

Mazumder, Sudip K., Acharya, Kaustuva, & Tahir, Muhammad (2009). Joint optimization of control performance and network resource utilization in homogeneous power networks. *IEEE Transactions on Industrial Electronics*, *56*(5), 1736–1745.

Negenborn, Rudy R., De Schutter, Bart, & Hellendoorn, Hans (2008). Multi-agent model predictive control for transportation networks: Serial versus parallel schemes. *Engineering Applications of Artificial Intelligence*, *21*(3), 353–366.

Nesic, Dragan, Teel, Andrew, & Carnevale, Daniele (2009). Explicit computation of the sampling period in emulation of controllers for nonlinear sampled-data systems. *IEEE Transactions on Automatic Control*, *54*(3), 619–624.

Papadimitriou, Christos H., & Steiglitz, Kenneth (1998). *Combinatorial optimization: Algorithms and complexity*. Mineola, NY: Dover.

Peng, Chen, & Yang, Tai Cheng (2013). Event-triggered communication and H∞ control co-design for networked control systems. *Automatica*, *49*(5), 1326–1332.

Pop, Traian, Pop, Paul, Eles, Petru, Peng, Zebo, & Andrei, Alexandru (2008). Timing analysis of the flexray communication protocol. *Real-Time Systems*, *39*(1–3), 205–235.

Rawlings, James Blake, & Mayne, David Q. (2009). *Model predictive control: Theory and design*. Madison, WI: Nob Hill Pub..

Robert Bosch GmbH (1991). *CAN specification (Version 2.0)*.

Roy, Debayan, Zhang, Licong, Chang, Wanli, Goswami, Dip, & Chakraborty, Samarjit (2016). Multi-objective co-optimization of FlexRay-based distributed control systems. In *2016 IEEE real-time and embedded technology and applications symposium* (pp. 1–12). IEEE.

Sha, Lui, Abdelzaher, Tarek, Årzén, Karl-Erik, Cervin, Anton, Baker, Theodore, Burns, Alan, et al. (2004). Real time scheduling theory: A historical perspective. *Real-Time Systems*, *28*(2–3), 101–155.

Shi, Lin, Bart De, Schutter, Yugeng, Xi, & Hans, Hellendoorn (2011). Fast model predictive control for urban road networks via MILP. *IEEE Transactions on Intelligent Transportation Systems*, *12*(3), 846–856.

Shi, Zhenwu, Yao, Ningshi, & Zhang, Fumin (2017). Scheduling feasibility of energy management in micro-grids based on significant moment analysis. In Houbing Song, Danda Rawat, Sabina Jeschke, & Christian Brecher (Eds.), *Cyber-physical systems* (pp. 431–449). New York, NY: Elsevier.

Shi, Zhenwu, & Zhang, Fumin (2013). Predicting time-delays under real-time scheduling for linear model predictive control. In *Proceedings of the 2013 international conference on computing, networking and communications* (pp. 205–209). IEEE.

Shi, Zhenwu, & Zhang, Fumin (2017). Model predictive control under timing constraints induced by controller area networks. *Real-Time Systems*, *53*(2), 196–227.

Touretzky, Cara R., & Baldea, Michael (2014). Integrating scheduling and control for economic MPC of buildings with energy storage. *Journal of Process Control*, *24*(8), 1292–1300.

Walsh, Gregory C., Hong, Ye, & Bushnell, Linda G. (2002). Stability analysis of networked control systems. *IEEE Transactions on Control Systems Technology*, *10*(3), 438–446.

Wang, Liuping (2009). *Model predictive control system design and implementation using MATLAB*. London, UK: Springer-Verlag London Limited.

Wang, Xiaotian, Shi, Zhenwu, Zhang, Fumin, & Wang, Yue (2015). Dynamic real-time scheduling for human-agent collaboration systems based on mutual trust. *Cyber-Physical Systems*, *1*(2–4), 76–90.

Yan, Fei, Dridi, Mahjoub, & El Moudni, Abdellah (2013). An autonomous vehicle sequencing problem at intersections: A genetic algorithm approach. *International Journal of Applied Mathematics and Computer Science*, *23*(1), 183–200.

Yao, Leehter, Chang, Wen-Chi, & Yen, Rong-Liang (2005). An iterative deepening genetic algorithm for scheduling of direct load control. *IEEE Transactions on Power Systems*, *20*(3), 1414–1421.

Yao, Ningshi, Malisoff, Michael, & Zhang, Fumin (2017). Contention resolving optimal priority assignment for event-triggered model predictive controllers. In *Proceedings of the 2017 American control conference* (pp. 2357–2362). IEEE.

Zhang, Yue J., Malikopoulos, Andreas A., & Cassandras, Christos G. (2016). Optimal control and coordination of connected and automated vehicles at urban traffic intersections. In *Proceedings of the American control conference* (pp. 6227–6232). IEEE.

Zhang, Fumin, Shi, Zhenwu, & Mukhopadhyay, Shayok (2013). Robustness analysis for battery-supported cyber-physical systems. *ACM Transactions on Embedded Computing Systems*, *12*(3), 69.

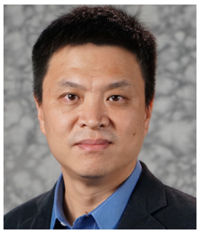
Zhang, Fumin, Szwaykowska, Klementyna, Wolf, Wayne, & Mooney, Vincent (2008). Task scheduling for control oriented requirements for cyber-physical systems. In *Real-time systems symposium, 2008* (pp. 47–56). IEEE.

Zhao, Yang, Lu, Yuehong, Yan, Chengchu, & Wang, Shengwei (2015). MPC-based optimal scheduling of grid-connected low energy buildings with thermal energy storages. *Energy and Buildings*, *86*, 415–426.

Zhou, Chuan, Du, Mingli, & Chen, Qingwei (2012). Co-design of dynamic scheduling and H-infinity control for networked control systems. *Applied Mathematics and Computation*, *218*(21), 10767–10775.

Ningshi Yao received the B.S. degree from Zhejiang University, Hangzhou, China, in 2014. She has been pursuing a Ph.D. degree from the School of Electrical and Computer Engineering at Georgia Institute of Technology, Atlanta, USA, since 2014. Her research interests include control theory and human robot interaction.

Michael Malisoff is the Roy Paul Daniels Professor #3 in the Louisiana State University College of Science. He earned his Ph.D. in Mathematics in 2000 from Rutgers University. He received the First Place Student Best Paper Award at the 1999 IEEE Conference on Decision and Control and 7 NSF research grants at PI. He is an associate editor of Asian Journal of Control, Discrete and Continuous Dynamical Systems Series B, European Journal of Control, and SIAM Journal on Control and Optimization.



Fumin Zhang is a professor in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. He received a Ph.D. degree in 2004 from the University of Maryland (College Park) in Electrical Engineering, and held a postdoctoral position in Princeton University from 2004 to 2007. His research interests include mobile sensor networks, maritime robotics, control systems, and theoretical foundations for cyber–physical systems. He received the NSF CAREER Award in 2009 and the ONR Young Investigator Program Award in 2010.