Real-Time Systems (2022) 58:275–312
https://doi.org/10.1007/s11241-022-09388-5

Design optimization for real-time systems with sustainable schedulability analysis

Yecheng Zhao[1] · **Runzhi Zhou**[1] · **Haibo Zeng**[1]

Accepted: 20 June 2022 / Published online: 16 August 2022

Abstract

The design of modern real-time systems not only needs to guarantee their timing correctness, but also involves other critical metrics such as control quality and energy consumption. As real-time systems become increasingly complex, there is an urgent need for efficient optimization techniques that can handle large-scale systems. However, the complexity of schedulability analysis often makes it difficult to be directly incorporated in standard optimization frameworks, and inefficient to be checked against a large number of candidate solutions. In this paper, we propose a novel optimization framework for the design of real-time systems. It leverages the sustainability of schedulability analysis that is applicable for a large class of real-time systems. It builds a counterexample-guided iterative procedure to efficiently learn from an unschedulable solution and rule out many similar ones. Compared to the state-of-the-art, the proposed framework may be ten times faster while providing solutions with the same quality. This work is a journal extension to the conference paper published at RTSS 2020, which adds new discussions for techniques that improve the algorithm scalability, as well as a set of new experiments to better evaluate the proposed framework.

Keywords Design optimization · Sustainable schedulability analysis

✉ Yecheng Zhao
zyecheng@vt.edu

Runzhi Zhou
zrz@seas.upenn.edu

Haibo Zeng
hbzeng@vt.edu

[1] Virginia Polytechnic Institute and State University Blacksburg, Virginia, USA

1 Introduction

Design optimization techniques are becoming vital and urgent for a number of application domains for real-time systems. For example, the automotive industry is extremely cost sensitive yet its products are highly safety critical (Ebert and Favaro 2017). Unmanned aerial vehicles powered by batteries must carefully plan and operate according to their tight energy budget (Hassanalian and Abdelkefi 2017).

There has been a rich set of research on the development of timing and schedulability analysis over the past years. However, many of the analysis techniques are either impossible, or too complex and inefficient, to be used in well-established optimization frameworks (i.e., mathematical programming) (Zhao et al. 2018). As a result, the existing practice for design optimization often has to rely on ad-hoc approaches. This typically comes with the loss of solution quality as well as limited applicability. Hence, an efficient and general framework that bridges the gap between schedulability analysis and optimization is critical to the future success of real-time systems design.

In this paper, we seek to address this urgent need and propose a general framework for optimizing real-time systems. We leverage the concept of sustainable schedulability analysis that is recommended as a good engineering practice for real-time systems (Baruah and Burns 2006). Specifically, if a task system is schedulable under a sustainable schedulability analysis, then it should remain to be schedulable with, for example, decreased worst case execution time (WCET), or increased period.

The design of our framework centers around the concept of Maximal Unschedulable Assignment (MUA) to variables that are sustainable in the schedulability analysis, including task WCET and period. It avoids the direct formulation of schedulability region, but uses MUAs to provide an efficient abstraction of the schedulability constraints. It develops a counterexample (i.e., unschedulable solutions) guided paradigm to learn from an unschedulable solution and rule out many similar unschedulable ones. It also builds an MUA-driven branching algorithm that takes advantage of the special structure in the optimization problem.

Our framework is generally applicable to a broad range of optimization problems for real-time systems. We use two case studies to demonstrate the benefit of our framework. The first is the optimization of energy consumption on platforms with dynamic voltage and frequency scaling (DVFS), where task WCETs change based on the selected CPU frequency. The second is the selection of task periods to optimize control quality under schedulability constraints (Mancuso et al. 2014). Compared to the state-of-the-art, the proposed framework may be ten times faster while providing solutions with the same quality.

Extended version This paper is an extension to the conference publication (Zhao et al. 2020) appeared at RTSS 2020. Comparing to the conference version, this paper adds the following new contents:

1. In Sect. 5.2, we describe a size limit K on the number of nodes to keep when expanding the MUA-driven branching tree in each iteration. This addresses the

issue of exponential growth of tree size for large problems. We discuss how to select the value K to balance between optimization quality and algorithm scalability.

2. In Sect. 5.4, we perform a more in-depth analysis of the MUA conversion procedure. We discussed an improved conversion algorithm in Algorithm 2. The new algorithm treats each decision variable in a way that is more sensitive to the objective function. We add two examples, Examples 5 and 6, to demonstrate how the new algorithm improves convergence rate over Algorithm 1 for separable and monotonic objective functions.
3. In Section 7.1, we perform a number of new experiments for the case study of energy consumption optimization. Specifically, we add new experiment results in Fig. 6 that compares the framework using Algorithm 1 and the one with Algorithm 2. We also add new experiment results in Table 2 and Table 3 that compares the runtime and solution quality for different settings of K.
4. In Sect. 7.2, we perform a number of new experiments for the case study of control performance optimization. In Fig. 9 we compare Algorithm 1 and the newly proposed Algorithm 2. We also perform new experiments that compare the runtime and sub-optimality for different settings of K. The results are given in the newly added Figs. 10 and 11. Finally, we perform another experiment where task priority assignment is also a design variable. We evaluate the framework in terms of runtime and solution quality improvement. The results are given in the two new figures, Figs. 12 and 13.

Paper organization The rest of the paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the system model and the problem setting that fits our framework. Section 4 describes the concept of Maximal Unschedulable Assignment (MUA). Section 5 develops the framework based on MUA. Section 6 discusses the applicability and limitations. Section 7 shows the experiments to demonstrate the advantage of our framework. Finally, Sect. 9 concludes the paper.

2 Related work

There is a rich literature for the optimization of real-time systems. Generally speaking, the current approaches can be classified into four categories: (i) meta heuristics such as simulated annealing (e.g., (Tindell et al. 1992; Bate and Emberson 2006)) and genetic algorithms (e.g., (Hamann et al. 2004; Shin and Sunwoo 2007)); (ii) problem specific heuristics (e.g., (Saksena and Wang 2000; Han et al. 2013; Wang et al. 2016)); (iii) adoption of existing optimization frameworks such as branch-and-bound (BnB) (e.g., (Wang and Saksena 1999; Al-Bayati et al. 2015)), Mixed Integer Linear Programming (MILP) (e.g., (Zeng and Di Natale 2012; Gu et al. 2016)), and convex programming (e.g., (Huang et al. 2014)); (iv) customized optimization frameworks that are tuned for specific design variables in real-time systems, such as the optimization of task priority (Zhao and Zeng 2017, 2017, 2018, 2018, 2019), task period (Zhao et al. 2018), and task offset (Bansal et al. 2018). The first two

categories either do not have any guarantee on solution quality, or suffer from scalability issues and may have difficulty to handle large industrial designs. For the third category, besides the possible scalability issues, it also requires that the schedulability analysis can be formulated in the chosen framework, which may not always be possible. The current approaches in the fourth category are also limited in their applicability. For example, unlike the proposed framework in this paper, none of them (Zhao et al. 2018; Zhao and Zeng 2017, 2017, 2018; Bansal et al. 2018; Zhao and Zeng 2018, 2019) considers task WCETs as design variables.

The problem of task period selection has been studied in the literature (Seto et al. 1998; Bini and Di Natale 2005). In particular, Bini and Di Natale (2005) propose a problem specific branch-and-bound technique for optimizing task rate, but it describes the exact schedulability region in the domain of task rates, hence it may be applicable to problems with optimization objectives that depend only on task periods (not on other variables such as task response times). Mancuso et al. (2014) develop a branch-and-bound algorithm, where a linear lower bound is adopted as an approximation to task response time. In Shin and Sunwoo (2007), a genetic algorithm is used for the problem to minimize the sum of end-to-end delays in networked control systems. Davare et al. consider the period optimization to minimize end-to-end latencies for a set of paths, and formulate it in mixed integer geometric programming (MIGP) framework (Davare et al. 2007). Differently, Zhao et al. (2018) propose a customized procedure specialized for the minimization of end-to-end latencies, which is several orders of magnitude faster than (Davare et al. 2007). Our approach is also to develop a customized framework. However, it not only is more generally applicable than (Zhao et al. 2018), but also may run 100× faster.

There are various approaches proposed to address the problem of optimizing energy consumption for systems with DVFS, see a related review in Bambagini et al. (2016). However, they are all focusing on one particular scheduling model and associated schedulability analysis. For example, Huang et al. (2014) consider mixed-criticality systems scheduled with Earliest Deadline First with Virtual Deadline, which allows to formulate the problem as a convex program. Instead, our framework is generally applicable to any systems as long as the schedulability analysis is sustainable.

In summary, compared to the existing approaches, our framework is applicable to a larger class of optimization problems in real-time systems. It does not pertain to a particular scheduling model or schedulability analysis, but can be used for any systems with sustainable schedulability analysis. It applies to the optimization of various decision variables including task WCET, period, deadline, or priority assignment. Finally, it may still be much faster than the state-of-the-art, as demonstrated in the experimental results.

3 System model

In this paper, we consider a general setting of real-time systems for which the associated schedulability analysis is sustainable (Baruah and Burns 2006). It contains m tasks indexed from 1 to m. The design optimization of a real-time system is to

select the appropriate values for design variables such that (a) a given cost function is minimized, and (b) system schedulability is satisfied. Mathematically it can be expressed as follows

$$\begin{aligned} &\min\ F(\mathbf{X}) \\ s.t.\ &\text{system schedulability} \\ &x_j \in [x_j^l, x_j^u], \forall j = 1, ..., n \end{aligned} \tag{1}$$

where $\mathbf{X} = (x_1, ...x_n)$ is the vector of variables. Here we first focus on the case that $\mathbf{X}$ may include the response time R_i, WCET C_i, deadline D_i and/or period T_i for any task τ_i. In Sect. 6 we will discuss how to handle the case where priority assignment is also part of the decision variables. Each variable x_j in $\mathbf{X}$ takes integer values within a bounded range $[x_j^l, x_j^u]$ (i.e., the design variables have finite resolutions). We do not impose any particular form of schedulability analysis, as long as it is sustainable (Baruah and Burns 2006).

Sustainability is proposed as a guideline for the development of schedulability analysis techniques in real-time systems (Baruah and Burns 2006). Specifically, a schedulability analysis is defined as sustainable if any schedulable task system remains schedulable with (i) decreased WCET C_i; (ii) larger period T_i; (iii) larger deadline D_i for any task τ_i, among others.[1]

Here we discuss how to leverage the sustainability with respect to task deadlines to handle the case that response time R_i appears in the objective function. In this case, we not only need to make sure that $R_i \leq D_i$ (which can be satisfied by any R_i that is no larger than D_i), but also a precise value of R_i in order to compare different schedulable solutions. In this case, we replace R_i with a virtual deadline $\hat{D}_i$ (Zhao and Zeng 2017), which can be interpreted as a safe estimation on R_i (hence $R_i \leq \hat{D}_i \leq D_i$) when the system is schedulable. It is easy to see any schedulability analysis that is sustainable with respect to the deadline D_i is also sustainable with respect to $\hat{D}_i$.

For systems with sustainable schedulability analysis, without loss of generality, they satisfy the following property.

Property 1 *The system schedulability constraints can be written as* $G(\mathbf{X}) \leq 0$, *where each function in* $G(\mathbf{X})$ *is* **monotonically non-increasing** *with respect to each variable* x_j *in* $\mathbf{X}$. *Hence, if a smaller assignment to* x_j (*e.g., the period, virtual deadline, or the additive inverse* $-C_j$ *of WCET* C_j) *makes the system schedulable, then a larger assignment to* x_j *also does* (*assuming all other variables remain unchanged*).

Note that some variables such as WCET C_i may be opposite to the above property (smaller C_i corresponds to easier schedulability). In this case, we can simply perform a variable conversion (i.e., replacing C_i with $C_i' = -C_i$) to make it conform to the assumption.

[1] For simplicity we call such a schedulability analysis sustainable, but it is termed as self-sustainable analysis in Baker and Baruah (2009), see a detailed discussion therein.

3.1 Examples on sustainable schedulability analysis

Sustainability is a general property that applies to many schedulability analysis techniques in real-time systems. For example, for the classical Liu-Layland task model scheduled with fixed priority (Liu and Layland 1973), the response time based schedulability analysis (Audsley et al. 1991) is sustainable

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \leq D_i \tag{2}$$

A more complicated example is mixed-criticality systems scheduled with Adaptive Mixed-Criticality (AMC) scheduling policy (Baruah et al. 2011). The proposed AMC-rtb and AMC-max schedulability analyses (Baruah et al. 2011) are both sustainable (Guo et al. 2017). The two analyses mainly differ in the estimation of interferences from higher priority tasks during the criticality change.

AMC-rtb analysis calculates the response time of a HI-ciriticality task τ_i during the criticality change as

$$\begin{aligned} R_i(HI) = C_i(HI) + \sum_{\forall j \in hpH(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) + \\ \sum_{\forall j \in hpL(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \end{aligned} \tag{3}$$

Here $C_i(HI)$ represents the WCET of τ_i in HI criticality mode. $R_i(LO)$ is the response time of τ_i in LO criticality mode given by Eq. (2). $hpH(i)$ and $hpL(i)$ represent the set of HI- and LO-criticality tasks of higher priority than τ_i, respectively.

Intuitively, AMC-rtb assumes that a HI-criticality task always executes in HI-criticality mode and LO-criticality task may execute up to $R_i(LO)$. AMC-max improves upon AMC-rtb by considering different specific time instants of criticality change and dividing the workload of higher priority HI-criticality tasks into LO-mode and HI-mode. Specifically, given a time instant s of criticality change, AMC-max computes the WCRT of τ_i as follows

$$\begin{aligned} R_i(HI, s) = C_i(HI) + \sum_{\forall j \in hpL(i)} \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO) + \\ \sum_{\forall j \in hpH(i)} M(j, s, R_i(HI, s)) C_j(HI) + \\ \sum_{\forall j \in hpH(i)} \left(\left\lceil \frac{t}{T_j} \right\rceil - M(j, s, R_i(HI, s)) \right) C_j(LO) \end{aligned} \tag{4}$$

where $M(j, s, t)$ represents the maximum number of instances of τ_j that are released as HI-criticality instances during the time interval $[s, t]$, which is expressed as

$$M(j,s,t) = \min\left\{\left\lceil\frac{t-s-(T_j-D_j)}{T_j}\right\rceil+1, \left\lceil\frac{t}{T_j}\right\rceil\right\} \tag{5}$$

The WCRT of τ_i during the criticality change can be computed by examining all possible time instants s of criticality change

$$R_i(HI) = \max_{\forall s} R_i(HI,s) \tag{6}$$

It is shown to be sufficient to only consider those s in the interval $[0, R_i(LO))$ (Baruah et al. 2011).

3.2 Example optimization problems

We now provide two examples that fit our framework.

3.2.1 Optimizing control quality

The first problem is to optimize control performance for a set of periodic tasks scheduled on a uniprocessor (Mancuso et al. 2014). The objective function, which represents the control cost, are approximated as a weighted sum of task period T_i and response time R_i for each task τ_i (Mancuso et al. 2014)

$$F(\mathbf{X}) = \sum_{i=1}^{m} \alpha_i T_i + \beta_i R_i \tag{7}$$

where α_i and β_i are given constant weights. Note that in the above objective function R_i can be replaced by a (virtual) deadline D_i. The reason is that for every optimal solution for (7), there is a solution with the same objective value for 8 by simply setting $D_i = R_i$.

$$F(\mathbf{X}) = \sum_{i=1}^{m} \alpha_i T_i + \beta_i D_i \tag{8}$$

Our framework works for any scheduling policy as long as its schedulability analysis is sustainable. In the experiments (Sect. 7.2) we assume the same as those in Mancuso et al. (2014), i.e., the schedulability analysis in Eq. (2).

3.2.2 Energy minimization with DVFS

Platforms with DVFS capabilities allow to adjust the CPU clock rate to save energy. Higher clock rate gives smaller WCET, which generally helps schedulability. However, this comes with an increased energy consumption. The goal is to determine the clock rate f_i for executing each task τ_i such that the system is schedulable while the total energy is minimized.

Specifically, suppose τ_i has an execution time C_i^b measured at a base clock rate f^b, then its execution time at another clock rate f_i can be estimated as $C_i = C_i^b \times \frac{f^b}{f_i}$.

Thus $f_i = f^b \times \frac{C_i^b}{C_i}$. We normalize f^b to be 1 and consider that C_i^b is given, which makes C_i a decision variable in the optimization. We adopt the energy consumption objective formulated in Huang et al. (2014):

$$F(\mathbf{X}) = \sum_{\forall \tau_i} \frac{C_i^b f^b}{T_i} \cdot \beta \cdot (f_i)^{\alpha-1} = \sum_{\forall \tau_i} \frac{1}{T_i} \cdot \beta \cdot \frac{(C_i^b)^\alpha}{(C_i)^{\alpha-1}} \tag{9}$$

where β is a circuit-dependent constant. A common assumption for α is 3 (Nelson et al. 2011; Pagani and Chen 2014). Like the previous case, we do not impose any constraint on the scheduling policy as long as the associated schedulability analysis is sustainable. In the experiments (Sect. 7.1), we use Eq. (2) for a large number of random systems. For an industrial case study, we assume mixed-criticality systems scheduled with AMC, and adopt the AMC-rtb and AMC-max schedulability analyses (Baruah et al. 2011), both of which are sustainable (Guo et al. 2017).

4 Maximal unschedulable assignment

We now introduce the concept of Maximal Unschedulable Assignment (MUA).

Definition 1 An assignment

$$\mathcal{X} = (v_1, ...v_n) \tag{10}$$

is a valuation of each variable $x_i = v_i$ in $\mathbf{X}$. An assignment $\mathcal{X} = (v_1, ...v_n)$ is said to dominate another assignment $\mathcal{X}' = (v'_1, ...v'_n)$, denoted as $\mathcal{X} \succeq \mathcal{X}'$, if $\mathcal{X}$ is componentwise no smaller than $\mathcal{X}'$, i.e., $v_i \geq v'_i, \forall i$.

Definition 2 An assignment $\mathcal{X}$ is said to be unschedulable if it violates the schedulability constraints. $\mathcal{X}$ is a maximal unschedulable assignment (MUA) if (a) $\mathcal{X}$ is unschedulable and (b) there is no other unschedulable assignment that dominates $\mathcal{X}$.

We remark that the concept of MUAs is well-defined, since by Property 1 of schedulability constraints with respect to the variables in $\mathbf{X}$, any assignment $\mathcal{X}'$ that is dominated by an MUA $\mathcal{X}$ must also be unschedulable. Also, any two MUAs cannot dominate each other, otherwise one of them is not an MUA. Note that we assume variables with integer values (or in general discrete variables), hence the MUAs always exist.

Example 1 Consider a hypothetical optimization problem formulated in (11) where || denotes the logic OR operation. The decision variables are $\mathbf{X} = (x_1, x_2)$, which take values in the range [0, 9]. The feasibility region are formed by the disjunction of two linear constraints, which is shown in Fig. 1 as the green shadowed area.

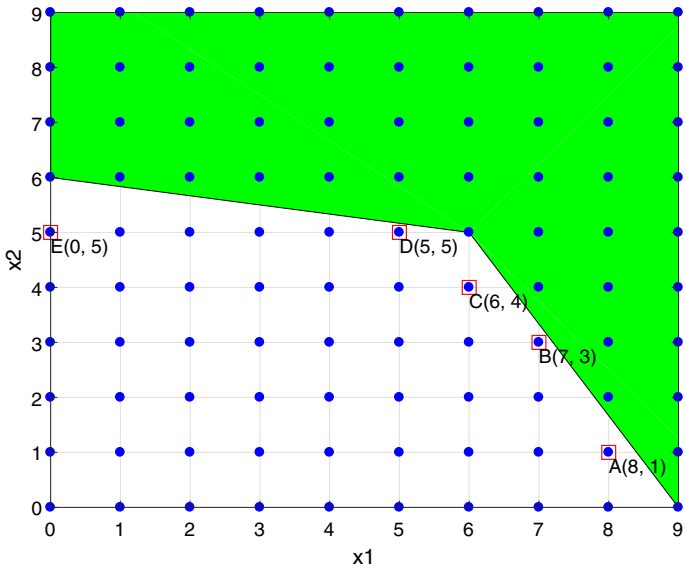


Fig. 1 The MUAs of the problem in Eq. (11)

$$s.t. \begin{array}{l} \min\ F(\mathbf{X}) = x_1 + x_2 \\ \left\| \begin{array}{l} x_1 + 6x_2 \geq 36 \\ 5x_1 + 3x_2 \geq 45 \end{array} \right. \\ 0 \leq x_1, x_2 \leq 9 \end{array} \tag{11}$$

In the figure, the five points $A = (8, 1)$, $B = (7, 3)$, $C = (6, 4)$, $D = (5, 5)$, $E = (0, 5)$ marked in the figure are all unschedulable assignments as they lie outside of the schedulability region. E is not an MUA, since D dominates E. Meanwhile, A, B, C and D are all MUAs since they are unschedulable assignments and there exists no other unschedulable assignment that dominates any of them.

By Property 1, any assignment dominated by an unschedulable assignment is also unschedulable. Therefore, an MUA $\mathcal{X} = (v_1, ...v_n)$ implies the following constraint that must be satisfied by any schedulable solution

$$\neg \left\{ \begin{array}{l} x_1 \leq v_1 \\ \ldots \\ x_n \leq v_n \end{array} \right. \Leftrightarrow \left\| \begin{array}{l} x_1 \geq v_1 + 1 \\ \ldots \\ x_n \geq v_n + 1 \end{array} \right. \tag{12}$$

where { denotes the logic AND operation. We call (12) the *implied constraint* by $\mathcal{X}$. Note that no matter how complicated the schedulability analysis is, the MUA-implied constraints will always take the form in (12), hence they are an abstract interpretation of the schedulability constraints. The higher the values in $\mathcal{X}$, the

stronger the implied constraint. In this sense, constraints implied by MUAs are the "strongest" type of constraints for schedulability. In Example 1, the implied constraints by MUAs $A = (8, 1)$, $B = (7, 3)$, $C = (6, 4)$, $D = (5, 5)$ together represent the exact schedulability region.

5 Optimization framework

We now present the optimization framework that builds upon the concept of MUA. By the sustainability of schedulability analysis (hence Property 1), once we find an unschedulable solution, we can generalize it to MUAs to simultaneously rule out many similar unschedulable solutions. This leads to the following key idea for the design of our framework: we use MUA-implied constraints as an efficient abstraction (as opposed to a direct formulation) of the schedulability region, and employ an iterative procedure to gradually learn those MUAs that are critical for determining the optimal solution (Sect. 5.1). Furthermore, we maintain an MUA-driven branching structure to allow incremental update of the branching tree and efficient solution to each leaf problem (Sect. 5.2).

5.1 MUA-guided iterative procedure

In real-time systems, the complexity of the schedulability analysis may prevent us from leveraging existing optimization frameworks. Consider the example of a mixed-criticality system scheduled according to adaptive mixed-criticality scheduling (Baruah et al. 2011). The system operates in two modes: low-criticality (LO) or high-criticality (HI). Correspondingly, the tasks in the system are categorized as either low-criticality(LO) or high-criticality(HI). HI tasks are characterized by two worst-case execution times, denoted $C(HI)$ and $C(LO)$), in HI and LO modes respectively. $C(HI)$ is usually much higher than $C(LO)$. When in LO mode, all tasks are scheduled and need to meet the deadline requirements. In HI mode, LO tasks are dropped and only HI tasks need to meet the deadline. Schedulability analysis for such system needs to consider the worst case scenario, taking into account that LO to HI criticality mode change may happen anytime. The AMC-max (Baruah et al. 2011) schedulability analysis, as detailed in Eq. (6), is the most accurate analysis for fixed-priority AMC scheduling. It considers all possible time instants s of criticality change up to the the task's worst-case response time in LO mode. However, the analysis is difficult to use in traditional mathematical programming frameworks. This is because the LO mode worst-case response time is typically also a function of the design parameters and therefore the range of s is not known a priori when formulating the mathematical programming model. As systems become more sophisticated and the associated schedulability analysis gets more delicate, often times a direct formulation in existing optimization frameworks is more difficulty if possible at all.

We consider a different approach that leverages the sustainability of schedulability analysis, a property that was first advocated by Baruah and Burns (2006).

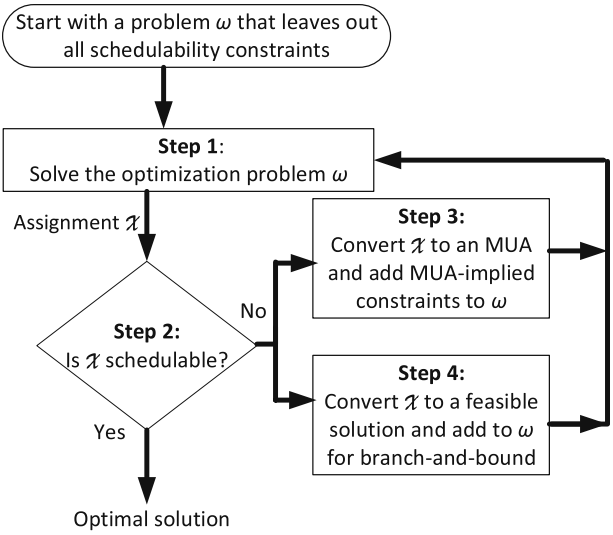


Fig. 2 MUA-guided optimization framework

Our key observation is that the sustainability of schedulability analysis (hence Property 1) allows to generalize from one unschedulable solution to MUAs, which can simultaneously rule out many similar unschedulable solutions. This is leveraged to develop the iterative optimization framework guided by counterexamples (i.e., unschedulable solutions), as illustrated in Fig. 2. Initially, it starts with an optimization problem ω that leaves out all the schedulability constraints. It then enters an iterative procedure that contains four steps. The first step is to solve ω using an MUA-driven branching algorithm. Here ω maintains to be a relaxed version of the original problem in Eq. (1), since it only includes the implied constraints from a subset of all MUAs (and hence only part of the schedulability constraints). The second step is to use the associated (sustainable) schedulability analysis to check if the returned solution $\mathcal{X}$ from Step 1 is schedulable or not. If yes, then $\mathcal{X}$ must also be an optimal solution to the original problem (Theorem 1). Otherwise, it performs two operations, Steps 3 and 4, that can be executed *in parallel*. In Step 3, it converts $\mathcal{X}$ to an MUA and adds the implied constraints (12) to ω. In this way, the counterexample (i.e., the unschedulable solution $\mathcal{X}$ returned in Step 1) is generalized as much as possible, such that many similar unschedulable solutions can be ruled out together. In Step 4, it converts $\mathcal{X}$ to a feasible solution and adds it to ω, which allows both efficient branch-and-bound and early termination of the algorithm. These steps will be explained from Sect. 5.3 to Sect. 5.5, after we present the MUA-driven tree structure (Sect. 5.2).

We now discuss two important properties of the framework, as stated in the following theorem.

Theorem 1 *The algorithm in Fig. 2 guarantees to terminate. If Step* 1 *in each iteration is solved optimally w.r.t. the added constraints, the algorithm guarantees to return an optimal solution upon termination.*

Proof During each iteration, the procedure has to find a solution $\mathcal{X}$ that satisfies the constraints implied by all the previously added MUAs. This means that if $\mathcal{X}$ is still unschedulable, then it must correspond to MUAs that are different from the known ones. Since the total number of MUAs is clearly finite and bounded by $\Omega(\Pi_{\forall i}(x_i^u - x_i^l + 1))$, the algorithm guarantees to terminate.

The implied constraint (12) only cuts away unschedulable decision space. Since the problem starts with no MUAs in ω, at any point during the optimization, the feasibility region defined by the added MUA-implied constraints maintains to be an over-approximation of the exact feasibility region. Hence the optimization problem ω in Step 1 has the same objective function but a larger feasibility region than the original problem (1). This implies that upon termination, where the algorithm finds a schedulable solution, the solution must also be optimal to (1).

Example 2 As an example, we apply the framework to the problem (11). Here we focus on how the framework prunes the unschedulable solutions, and ignore the step of finding feasible solutions.

Iteration 1 The algorithm initially ignores all schedulability constraints. Optimizing $F(\mathbf{X})$ gives the assignment $\mathcal{X} = (x_1, x_2) = (0, 0)$. Since $\mathcal{X}$ is clearly unschedulable, the algorithm proceeds to convert $\mathcal{X}$ into an MUA. Depending on the strategy for MUA computation (which will be discussed in Sect. 5.3), the algorithm may obtain any one of points $A = (8, 1)$, $B = (7, 3)$, $C = (6, 4)$, or $D = (5, 5)$ in Fig. 1. Suppose A is returned as the converted MUA.

Iteration 2 The feasibility region is updated to Fig. 3b, where the area colored in yellow represents the region cut away by the constraints implied by the MUA A. Optimizing $F(\mathbf{X})$ gives the assignment $\mathcal{X} = (0, 2)$. Since $\mathcal{X}$ is not schedulable, the algorithm proceeds to convert $\mathcal{X}$ into an MUA. Suppose $B = (7, 3)$ is returned as the converted MUA.

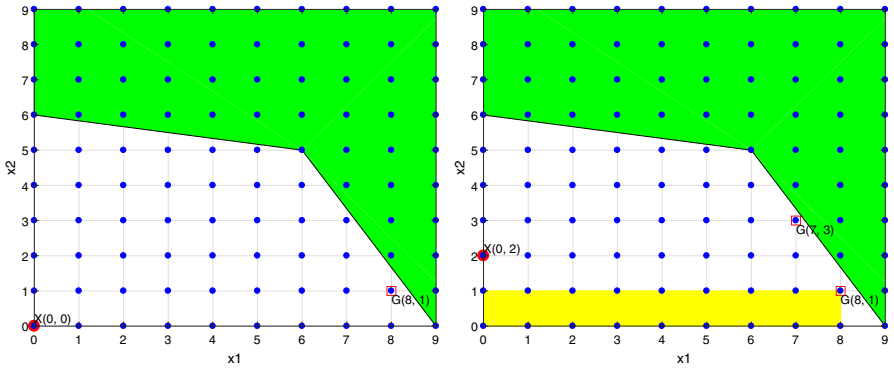
Iteration 3 The feasibility region is updated to Fig. 3c. Optimizing $F(\mathbf{X})$ returns the assignment $\mathcal{X} = (0, 4)$. Suppose $C = (6, 4)$ is returned as the computed MUA.

Iteration 4 The feasibility region is updated to Fig. 3d. Optimizing $F(\mathbf{X})$ returns the assignment $\mathcal{X} = (0, 5)$. Now the only possible MUA that can be computed is $D = (5, 5)$.

Iteration 5 The feasibility region is updated to Fig. 3e. Optimizing $F(\mathbf{X})$ gives the assignment $\mathcal{X} = (0, 6)$. This assignment is now schedulable and the algorithm terminates with an optimal solution $(x_1, x_2) = (0, 6)$.

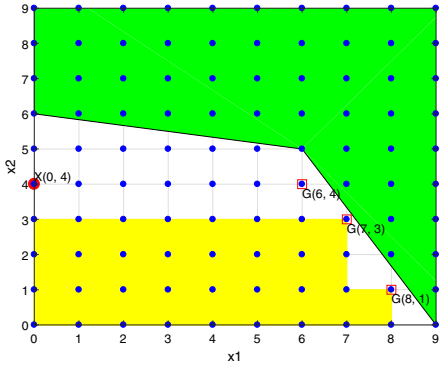
5.2 MUA-driven branching tree

As in Figure 2, the above procedure requires to solve an instance of ω during each iteration, which may only be slightly different from the previous iterations since the

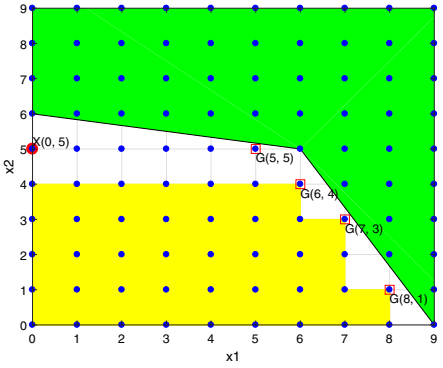


(a) Iteration 1

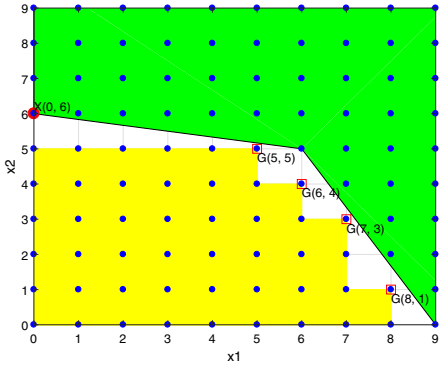
(b) Iteration 2



(c) Iteration 3



(d) Iteration 4



(e) Iteration 5

Fig. 3 The iterative procedure applied to the problem in Eq. (11). The yellow shadowed area is the pruned unschedulability at the end of the iteration

procedure only adds a handful of new MUAs to ω. Directly calling mathematical programming solvers to solve ω is not necessarily efficient since many solvers do

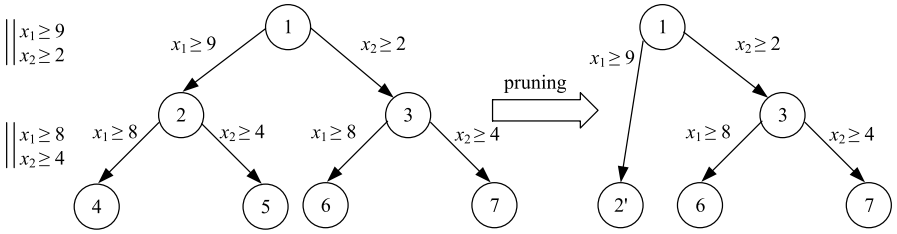


Fig. 4 Branching tree at the end of Iteration 2 of Example 2: original tree (left); after pruning (right)

not support incremental solving, i.e., they have to solve each instance of ω from scratch.[2]

Hence we build a branching tree structure to represent the MUA-implied constraints. It allows incremental updating of the tree, as well as efficiently solving each subproblem at the leaf nodes. Initially (i.e., before entering the iteration in Fig. 2) the tree only contains one (root) node. Each time a new MUA $\mathcal{X}$ is added to the problem ω, we add a new layer in the tree to represent the disjunction (i.e., logic OR) of the constraints implied by $\mathcal{X}$, where each branch (edge) in the new layer is a constraint in the disjunction. Each node $\mathcal{N}$ in the tree represents the set of conjunctive (i.e., logic AND) constraints along the path from the root node to this node $\mathcal{N}$. Since the constraint on each branch takes the particular form $x_j \geq u_j$ for some value u_j, the constraint represented by a node $\mathcal{N}$ can be simplified as

$$\begin{cases} x_1 \geq u_1 \\ \dots \\ x_n \geq u_n \end{cases} \tag{13}$$

For simplicity, we denote the node as $\mathcal{N} = [u_1, ..., u_n]$.

Some nodes in the tree are redundant, in the sense that the corresponding constraints may have already been satisfied by constraints along the path from the root. These nodes can be pruned to make the tree structure more compact.

Example 3 Consider the branching tree after Iteration 2 in Example 2, i.e., after adding the constraints implied by MUAs $A = (8, 1)$ and $B = (7, 3)$. The left hand side of Fig. 4 shows the resulted tree. The constraint corresponding to A is $x_1 \geq 9 \vee x_2 \geq 2$, and the constraint corresponding to B is $x_1 \geq 8 \vee x_2 \geq 4$, where $\vee$ denotes the logic OR operation. Hence, the root node (node 1) is first branched to two children, nodes 2 and 3, where the branch to node 2 represents the constraint $x_1 \geq 9$, and the branch to node 3 represents the constraint $x_2 \geq 2$. When adding the constraints for MUA $B = (7, 3)$, each of nodes 2 and 3 is branched to two children. Node 6, for example,

[2] The only known exception is the MILP solver CPLEX (International Business Machines Corporation xxxx), which provides an interface for building customized branching tree. But the price is that it can no longer run in parallel on multiple cores, which actually makes the whole procedure slower.

represents the constraints along the path from the root node, i.e., $x_2 \geq 2 \wedge x_1 \geq 8$, where $\wedge$ denotes the logic AND operation.

When adding the constraint $x_1 \geq 8 \vee x_2 \geq 4$ implied by MUA B to node 2, this constraint is already satisfied by the constraint $x_1 \geq 9$ represented by node 2, i.e., $(x_1 \geq 9) \wedge (x_1 \geq 8 \vee x_2 \geq 4) = x_1 \geq 9$. The right hand side of Fig. 4 shows the tree structure after pruning the redundant nodes.

The special form of MUA-implied constraints make it easy to check the redundancy of nodes.

Theorem 2 *Assume $\mathcal{N} = [u_1, ..., u_n]$ is an existing node in the tree. If an MUA $\mathcal{X} = (v_1, ..., v_n)$ to be added satisfies that $\exists i : u_i \geq v_i + 1$, then the constraints implied by $\mathcal{X}$ are redundant.*

Proof This is easy to see by checking the represented constraints of $\mathcal{N}$ in (13) and those of MUA $\mathcal{X}$ in (12).

$$\begin{cases} x_1 \geq u_1 \\ \dots \\ x_n \geq u_n \end{cases} \Rightarrow x_i \geq u_i \Rightarrow x_i \geq v_i + 1 \Rightarrow \begin{Vmatrix} x_1 \geq v_1 + 1 \\ \dots \\ x_n \geq v_n + 1 \end{Vmatrix}$$

Pruning redundant nodes avoids exploring redundant sub-trees and reduces search space. However, the number of nodes in each layer may still grow exponentially as more MUAs are being added. For large size problems that require many MUAs to define the optimal solution, it is impractical to examine and solve all possible leaf nodes in each iteration. To alleviate the problem, a size limit K can be set on the number of nodes to keep when adding a new MUA (and hence a new layer of nodes) to the tree. Specifically, each node in the layer represents an optimization problem with the same objective function and a constraint of form (13). We solve the problem and obtain the objective value. We then sort these nodes in ascending order according to their optimized objective values and only keep at most K best nodes. In other words, we only keep at most K nodes that has smaller objective values than the rest. In the next iteration, the framework only branches into new nodes from these K nodes. Since an MUA-implied constraint can have at most n disjuncted inequalities where n is the number of variables, the number of new nodes being branched into in each iteration is at most Kn. Likewise, among these Kn nodes, another K nodes will be selected for branching in the next iteration. Therefore, in each iteration, the framework solves at most Kn instances of node problems.

With a size limit K, the sub-problem in each iteration is only solved sub-optimally. In return, the algorithm gains better scalability. In most cases, a larger value on K gives better solution. But extremely large K does not necessarily brings more benefit on solution quality comparing to smaller one, but may drastically increases run-time. The best setting of K may depend on the nature of the problem. In Sect. 7, we evaluate how K affect the solution quality and scalability of the optimization algorithm.

In the following we explain how the steps in Fig. 2 are handled. For Step 2, it can use any schedulability analysis as long as it is sustainable.

5.3 Step 1: Solving ω

For Step 1, it requires to solve the following optimization problem at each leaf node $\mathcal{N} = [u_1, ..., u_n]$ in the tree:

$$\begin{aligned} &\min_{x_j \in \mathbf{X}} \ F(\mathbf{X}) \\ s.t. \ &\text{constraints of form (13)} \\ &x_j \in [x_j^l, x_j^u], \forall j = 1, ..., n \end{aligned} \tag{14}$$

The constraints in (14) are of a particularly simple form: there are no coupled constraints for any pair of variables x_i and x_j. This means that the overall optimization can be done by optimizing each variable in sequence (Boyd and Vandenberghe 2004). In other words, (14) can be transformed to

$$\begin{aligned} &\min_{x_1} \ \min_{x_2} ... \min_{x_n} \ F(\mathbf{X}) \\ s.t. \ &\text{constraints of form (13)} \\ &x_j \in [x_j^l, x_j^u], \forall j = 1, ..., n \end{aligned} \tag{15}$$

For example, if $F(\mathbf{X})$ is convex with respect to the variables, then solving (15) amounts to solving a series of single-variable convex programs, which is a lot easier than those with coupled constraints.

In addition, even if $F(\mathbf{X})$ is not convex (hence in general difficult to solve (Boyd and Vandenberghe 2004)), we may still be able to provide a simple solution to it. In real-time systems, often times there exists some tradeoff between schedulability and the metrics in the objective function. For example, when the CPU frequency is increased, the task WCET is lower, but the energy consumption will be higher. Similarly, increasing the task periods will make the task system easier to schedule, but the control quality and stability will be worsened. In component-based design, modularity (i.e., the number of exposed interfaces) is a critical metric, but better modularity may lead to larger code size and longer WCET (Tripakis and Lublinerman 2018). We leverage this observation to derive the following corollary, which is a direct consequence from Eq. (15). In fact, Corollary 3 applies to both problems in the experiments.

Corollary 3 *If $F(\mathbf{X})$ is monotonically non-decreasing with respect to each of the variables in $\mathbf{X}$, then the optimal solution to (14) is $x_1 = u_1, ..., x_n = u_n$.*

This corollary is intuitive since the objective function will push each variable x_i to its lowest possible value u_i.

5.4 Step 3: Converting an Unschedulable Assignment to MUA

In this section, we discuss algorithms for converting an unschedulable assignment $\mathcal{X}$ into an MUA $\mathcal{U}$. It utilizes Property 1, i.e., the schedulability analysis is sustainable with respect to the variables, to maximally increase each entry in $\mathcal{X}$ while maintaining its unschedulability. The procedure is summarized in Algorithm 1. It uses binary search to sequentially find the maximal value that each entry in $\mathcal{U}$ can be increased to while maintaining unschedulability. The algorithm requires $O(n \log x^{\max})$ number of schedulability analysis where $x^{\max} = \max\{x_1^u, ..., x_n^u\}$.

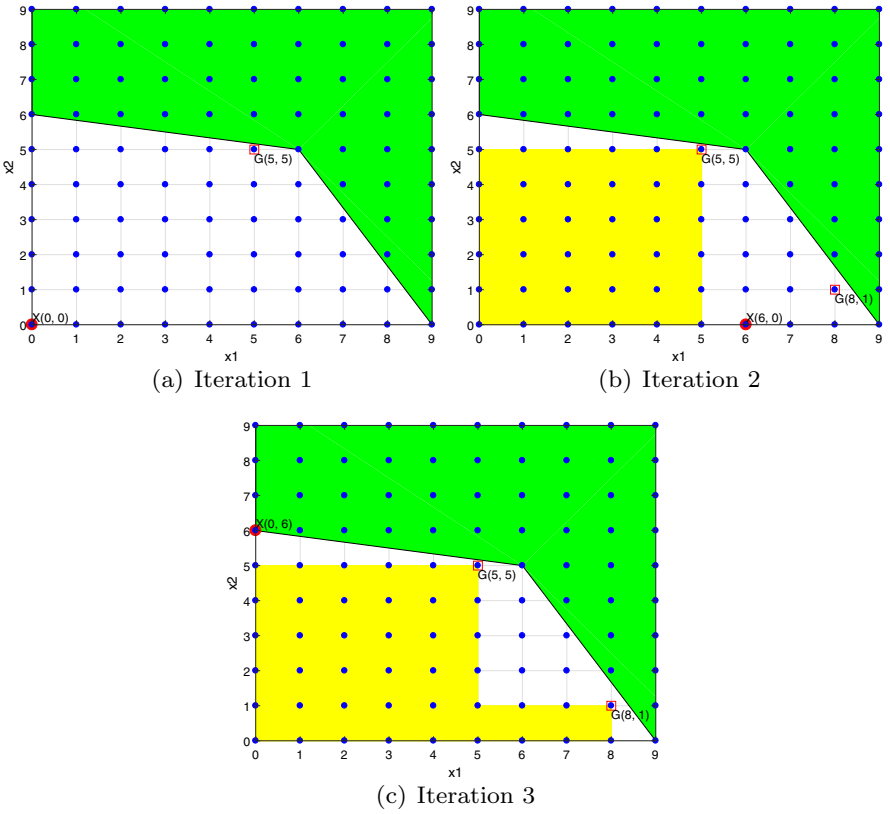
Algorithm 1 Conversion to MUA

```
1: function ConvertToMUA(Unschedulable Assignment 𝒳)
2:     𝒰 = (u_1, ..., u_n) = 𝒳
3:     for each entry u_i in 𝒰 do
4:         Use binary search to maximally increase u_i while keeping the system unschedulable
5:     end for
6:     return 𝒰
7: end function
```

This step is critical in the efficiency of the overall algorithm. Unlike typical counterexample guided algorithms, once we find an unschedulable solution, in the next iteration we rule out not only this solution but also many similar ones. Here the concept of MUA is critical: it is essentially a generalization from one unschedulable solution to many, which is a key in allowing a fast convergence rate of the framework.

Example 4 Using Algorithm 1 on Example 1 yields the exact trace of iterations shown in Fig. 3 in Example 2. For instance, in the first iteration, when a solution $\mathcal{X} = (0, 0)$ is returned, Algorithm 1 first increases u_1 to 8 since that is the largest value of x_1 that still makes the system unschedulable. It then increases u_2 from 0 to 1 to get the MUA $\mathcal{U} = (8, 1)$.

Although simple and straightforward, the returned MUA may not give the best convergence rate w.r.t the objective function. Specifically, the procedure in Algorithm 1 tends to give a larger increase on entries that are visited earlier, and a smaller increase on entries that are visited later. This is mainly because each time an entry is increased, the assignment becomes closer to being feasible, which leaves less room for variables visited later in the order to increase. This can negatively impact convergence rate if the objective function is sensitive to these variables that are visited later. For example, in iteration 1 of Example 2, since x_1 is visited first in Algorithm 1, the computed MUA $\mathcal{U} = (8, 1)$ has a large increase of 8 on x_1, but only a very small increase of 1 on x_2. However, the objective function $F(\mathbf{X}) = x_1 + x_2$ is equally sensitive to both x_1 and x_2. As a result, optimizing problem ω in the following iteration chooses to satisfy $x_2 \geq 2$ and gives solution $\mathcal{X} = (0, 2)$ that only increases the objective value by 2. An intuitive improvement



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3

Fig. 5 Iterations of optimizing Example 2 using Algorithm 2 for MUA conversion

in this case is to simultaneously increase all entries first before sequentially increasing each one. This gives an improved MUA conversion procedure in Algorithm 2.

Algorithm 2 Improved MUA conversion algorithm

1: **function** ConvertToMUAImproved(Unschedulable assignments $\mathcal{X}$)
2: $\mathcal{U} = (u_1, ..., u_n) = \mathcal{X}$, $d = 0$
3: Use binary search to maximally increase d while keeping $\mathcal{U}' = (u_1 + d, ..., u_n + d)$ unschedulable
4: Update $\mathcal{U} = (u_1 + d, ..., u_n + d)$
5: Apply Algorithm 1 to convert $\mathcal{U}$ into an MUA
6: **return** $\mathcal{U}'$
7: **end function**

The new algorithm first increases all entries simultaneously by the same amount d as much as possible, and then applies Algorithm 1 to convert it into a valid MUA. This gives a more balanced increase for each variable.

Example 5 We now revisit Example 2 and apply Algorithm 2 for MUA conversion. The trace of iterations is shown in Fig. 5.

Iteration 1 Similar to iteration 1 in Example 2 (that uses Algorithm 1), solution $\mathcal{X} = (0, 0)$ is returned for computing MUA. Since all entries are simultaneously increased first, Algorithm 2 computes MUA $\mathcal{U}_1 = (5, 5)$ instead of (8, 1).

Iteration 2 With $\mathcal{U}_1 = (5, 5)$, the feasibility region is updated to Fig. 3b. There are two solutions of equal optimal objective value: (6, 0) and (0, 6). The latter is the true optimal solution. Depending on the implementation for solving ω, either of them may be returned. If (0, 6) is returned, the framework terminates already. If (6, 0) is returned, Algorithm 2 will compute MUA $\mathcal{U}_2 = (8, 1)$ (it first increases both entries simultaneously to (7, 1), then applies Algorithm 1, which gives (8, 1)).

Iteration 3 With $\mathcal{U}_1 = (5, 5)$ and $\mathcal{U}_2 = (8, 1)$, the feasibility region becomes Fig. 3c. Solving the problem returns the true optimal solution $\mathcal{X} = (0, 6)$. The framework terminates.

Comparing with Algorithm 1, the use of Algorithm 2 reduces the total number of iterations from 5 to 3. In particular, after the first iteration, the objective value drastically increases from 0 to 6, while in the original Example 2, it increases only to 2. This illustrates that a proper strategy of computing MUAs can have a significant impact on the convergence speed. Intuitively, the goal is to find a proper way to increase entries such that the resulting MUA-implied constraint gives the largest increase on the objective value.

To achieve this, an important factor to consider is how sensitive the objective function is to each variable. Algorithm 2 simultaneously increase all entries by the same amount, and thus essentially assumes that the objective function is equally sensitive to each variable. This however, is generally not true. For example, if the objective function in (11) is changed to $F(\mathbf{X}) = x_1 + 8x_2$, then a better strategy is to increase x_1 by 8 times the amount x_2 is increased, i.e. changing line 3 of Algorithm 2 to $\mathcal{U}' = (u_1 + 8d, u_2 + d)$. If we revisit the example in this case, it is not difficult to see that this will lead the algorithm to compute MUA $\mathcal{U}_1 = (8, 1)$ in the first iteration. In the second iteration, the framework will get the true optimal solution (9, 0) and terminates. However, for the original scheme in Algorithm 2, the framework will need to go through the same 3 iterations in Fig. 5 before obtaining the optimal solution.

Finding the best MUA conversion that generally works well can be challenging. We discuss a specific but rather common scenario, where the objective function satisfies the following two properties.

- The objective function is separable in each variable and can be written as a sum of functions, i.e. $F(\mathbf{X}) = f_1(x_1) + f_2(x_2) + ...f_n(x_n)$
- Each function term $f_i(x_i)$ is invertible and non-decreasing w.r.t the increasing of x_i.

In this scenario, Algorithm 2 can be adapted to work generally well. The intuition is that for objective functions satisfying the above conditions, we can perform a variable conversion and reformulate the problem such that the objective

function is a linear sum of variables. Specifically, for each $f_i(x_i)$ we introduce a new decision variable y_i such that $x_i = f_i^{-1}(y_i)$, where f_i^{-1} is the inverse function of f_i. The objective function can then be re-written as $F(\mathbf{Y}) = y_1 + y_2 + ...y_n$. The second property suggests that increasing the value of y_i can only increase the value of x_i. This ensures that schedulability is also sustainable w.r.t. variable y_i. Therefore, the framework can be applied to solve the reformulated problem. Since $F(\mathbf{Y})$ is now equally sensitive to each variable, Algorithm 2 can be used to compute MUA that best benefits convergence speed. However, it should be noted that the values y_i can take are not necessarily integer and depends on $f_i(x_i)$. The value needs to be adjusted each time it is increased, i.e. rounding to the closest valid value.

Example 6 Consider a problem modified from (11) where the objective function is changed to $F(\mathbf{X}) = x_1 + 8x_2$

$$\begin{aligned} &\min\ F(\mathbf{X}) = x_1 + 8x_2 \\ s.t.\ &\left\| \begin{matrix} x_1 + 6x_2 \geq 36 \\ 5x_1 + 3x_2 \geq 45 \end{matrix} \right. \\ &0 \leq x_1, x_2 \leq 9, x_1 \in \mathbb{Z}, x_2 \in \mathbb{Z} \end{aligned} \tag{16}$$

Introduce variable $y_1 = x_1$ and $y_2 = 8x_2$ and re-write the above problem as

$$\begin{aligned} &\min\ F(\mathbf{Y}) = y_1 + y_2 \\ s.t.\ &\left\| \begin{matrix} y_1 + \frac{3}{4}y_2 \geq 36 \\ 5y_1 + \frac{3}{8}y_2 \geq 45 \end{matrix} \right. \\ &0 \leq y_1 \leq 9, y_1 \in \mathbb{Z} \\ &0 \leq y_2 \leq 72, y_2 \in \mathbb{Z}, \quad y_2 \mod 8 \equiv 0, \end{aligned} \tag{17}$$

Note that the values y_2 can take are only integer multiples of 8, since x_2 takes integer values. We now apply the proposed framework with Algorithm 2 to the above problem.

Iteration 1 All constraints are ignored, solving the problem returns the solution $(y_1, y_2) = (0, 0)$. In Algorithm 2, since both entries are simultaneously increased by the same amount as much as possible, the returned MUA is $\mathcal{U}_1 = (y_1, y_2) = (8, 8)$. Note that this corresponds to an assignment of (8, 1) in terms of the original variable (x_1, x_2).

Iteration 2 $\mathcal{U}_1 = (y_1, y_2) = (8, 8)$, the problem is updated to

$$\begin{aligned} &\min\ F(\mathbf{Y}) = y_1 + y_2 \\ s.t.\ &\left\| \begin{matrix} y_1 \geq 8 + 1 \\ y_2 \geq 8 + 1 \end{matrix} \right. \Rightarrow \left\| \begin{matrix} y_1 \geq 9 \\ y_2 \geq 16 \end{matrix} \right. \end{aligned} \tag{18}$$

Solving the updated problem returns the true optimal solution $(y_1, y_2) = (9, 0)$. Note that this corresponds to the optimal solution $(x_1, x_2) = (9, 0)$ of the original problem.

5.5 Step 4: finding feasible solutions

The branching tree structure (as in Fig. 4) allows to implement a typical branch-and-bound algorithm, i.e., to use the known best solution $\mathcal{X}$ to cut the branches that are certainly no better than $\mathcal{X}$ and hence are surely suboptimal. Getting a feasible solution also allows returning useful results for designers even if they are not necessarily the global optimal solution. This facilitates the possible early termination of the overall procedure, otherwise it will not be able to get any feasible solution until the optimal solution is found.

Our main idea is to make use of the assignment returned in Step 1 and convert it heuristically into a good-quality schedulable assignment. The conversion can be performed concurrently with the rest of the optimization process (Steps 1 and 3). Specifically, in Step 2, if the assignment $\mathcal{X} = (v_1, ..., v_n)$ is unschedulable, we simply scale each variable x_i by a factor $a \in [0, 1]$, until $\mathcal{X}$ becomes schedulable

$$x_i = v_i + a(x_i^u - v_i) \tag{19}$$

When $a = 1$, x_i takes the upper-bound value x_i^u. We use binary search to find the minimum a such that the assignment $\mathcal{X}$ becomes schedulable after scaling.

6 Applicability and efficiency

In this section, we discuss the applicability and expected efficiency of the proposed techniques.

Applicability to optimizing priority assignment As mentioned earlier, the framework is applicable to optimizing task response time, period, WCETs or deadline, as long as the schedulability analysis is sustainable with respect to these variables. Regarding task priority assignment as part of the decision variables, this comes with two cases: (a) the objective function is independent from the task priorities; (b) it is sensitive to task priority assignments, such as the memory consumption in the software implementation that preserves the semantics of the synchronous reactive models (Zeng et al. 2011).

For case (a), we can leverage Audsley's algorithm (Audsley 2001) that is applicable to many task models and scheduling schemes (Davis et al. 2016): if there exists a schedulable priority assignment, Audsley's algorithm will be able to find it. Hence, we can leave out the variables of priority assignment in the problem ω, but instead incorporate the optimization of priority assignment in the procedure for checking schedulability (e.g., Step 2 in Fig. 2, and Line 4 in Algorithm 1): After fixing the values of all other variables, the existence of a schedulable priority assignment now can be efficiently checked by Audsley's algorithm.

For case (b), it is necessary to explicitly include those binary variables for task priority orders in the optimization problem ω in the framework of Fig. 2. In this case, we can leverage the concept of unschedulability core (Zhao and Zeng 2017). Intuitively, it is an irreducible representation of the reason why a given total priority

order is unschedulable, in the sense that relaxing any order will make the system schedulable. We leave the details of this discussion to future work.

Efficiency The proposed technique fits the best for problems that have the following characteristics:

1. The schedulability analysis is complex.
2. The rest of the problem is relatively simple.

For the first characteristic, the exact schedulability analysis is often NP-complete, even for the basic settings, e.g., periodic task with fixed priority scheduling (Ekberg and Yi 2017), or EDF scheduling with arbitrary deadlines (Ekberg and Yi 2015). In this case, even if the problem may be formulated in some standard mathematical programming framework, our approach might still be better, since it uses MUAs to abstract away the details of schedulability analysis, and only performs such an analysis on a small number of design choices guided by the objective function. Of course, there are cases that the schedulability analysis is particularly simple, such as the condition for tasks with implicit deadlines scheduled by EDF (Liu and Layland 1973), or the utilization based bound for EDF-VD (Huang et al. 2014). In this case, although our framework is still applicable, it is faster to directly handle the schedulability condition.

For the second characteristic, this is satisfied when the objective function has some friendly properties (such as monotonicity or convexity with respect to the variables), and the constraints only include the system schedulability. If the problem includes some additional constraints other than schedulability, then the problem at each leaf node of the branching tree (see Fig. 4) is not necessarily easy to solve, since there might be coupled constraints among the variables. In this case, it can be more efficient to use other appropriate solvers to directly solve ω at Step 1 of the framework.

7 Experiment result

We now use two problems to demonstrate the advantage of our framework. The first is the minimization of energy consumption, the second is the optimization of control quality.

7.1 Optimizing energy consumption with DVFS

In this experiment, we consider the energy consumption model in Eq. (9). The decision variables are task execution times, which can be adjusted by tuning the processor frequency. Note that the objective function meets the two requirements discussed in Sect. 5.4. Specifically, the objective function is separable in each decision variable and can be written as a sum of monotonic and invertible functions:

$$F(\mathbf{X}) = \sum_{\forall \tau_i} f_i(C_i)$$
$$f_i(C_i) = \frac{1}{T_i} \cdot \beta \cdot \frac{(C_i^b)^\alpha}{(C_i)^{\alpha-1}} \tag{20}$$

Thus we can perform a variable conversion to reformulate the problem as (21), where Algorithm 2 can be applied for computing MUAs with better convergence rate.

$$\begin{aligned} &\min F(\mathbf{Y}) = \textstyle\sum_{\forall \tau_i} y_i \\ &s.t.\ \ \tau_i \text{ schedulable with } C_i = f_i^{-1}(y_i), \forall \tau_i \end{aligned} \tag{21}$$

We use both random systems as well as an industrial case study to compare different approaches. To demonstrate that our framework is applicable to various scheduling models and schedulability analysis techniques, we assume the periodic task model and Eq. (2) in the experiments on random systems, and use AMC-rtb and AMC-max (Eqs. 3–6) in the experiments on the industrial case study. The relative simplicity of Eq. (2) compared to AMC-rtb and AMC-max also allows us to perform experiments on a large number of random systems.

Random Systems For random systems, we compare the following methods:

- MIGP: A mixed-integer geometric programming formulation, solved by the geometric programming solver gpposy (Mutapcic et al. 2006) with the BnB (bmi) solver in YALMIP (Löfberg 2004).
- MUA-MILP: The proposed iterative procedure in Fig. 2, but the subproblem ω with MUA-implied constraints is formulated as an MILP and solved using CPLEX (International Business Machines Corporation xxxx).
- MUA-incremental: The proposed technique depicted in Fig. 2 with MUA-driven branching tree for incremental update, where each problem at the leaf node is solved using Corollary 3. Note that the method uses Algorithm 2 for MUA conversion and the result is the same one presented in the original conference submission (Zhao et al. 2020). The algorithm was not discussed however due to space limitation.
- MUA-incremental-naive-conversion: The same as MUA-incremental, but MUA conversion uses Algorithm 1 instead of Algorithm 2.
- Minimum-single-speed: A simple heuristic that uses binary search to find the minimum single speed at which all tasks become schedulable.

To avoid excessive waiting, we adopt the strategy disucssed in Sect. 5.2 in and set $K = 10000$ to limit the number of nodes in each iteration. Also, the time limits of MUA-MILP and MUA-incremental are set to 600 seconds for each task system. The BnB (bmi) solver in YALMIP does not have a time limit setting and only allows to set a limit on the number of iterations. Therefore, we set a maximum iteration limit of 2000. This gives roughly a similar time limit for MIGP as those of the other methods.

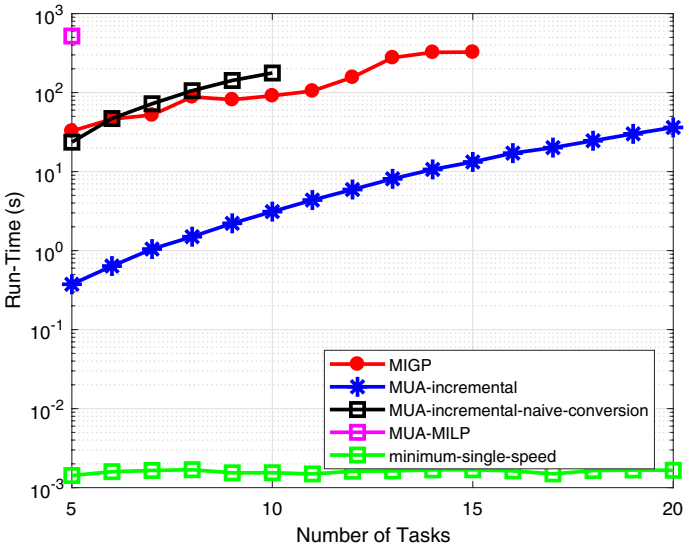


Fig. 6 Runtime of minimizing energy consumption on random systems

The task sets are generated synthetically as follows. For each task set, we first randomly select a system utilization in the range [0.5, 0.9]. We then generate a period T_i for each task according to log-uniform distribution in the range [100, 100000], and a utilization U_i for each task using UUnifast algorithm (Davis and Burns 2009). The corresponding WCET $C_i^b = T_i \cdot U_i$ is treated as the execution time at the base clock rate. The range of decision variable C_i is taken as $[C_i^b, 2C_i^b]$. This means that the clock rate can be decreased as low as half the base clock rate. The deadline D_i of each task τ_i is generated randomly in the range $[C_i^b, T_i]$. Priorities are assigned according to the deadline monotonic policy.

Figure 6 illustrates the average runtime over 1000 random systems for each m, the number of tasks in the system. The runtime of Minimum-single-speed is very short (a few milliseconds), as it is simply a binary search on a single period value. MUA-incremental is about one to two orders of magnitude faster than MIGP. The capping of MIGP that occurs for systems with 14 or 15 tasks is mainly due to a large number of cases reaching the iteration limit. Meanwhile, MUA-incremental is able to finish all the instances in the time limit. On the other hand, MUA-incremental-naive-conversion is comparable to MIGP and much slower than MUA-incremental, which demonstrates the benefit of Algorithm 2 in providing better convergence rate. As for MUA-MILP, it is very slow such that even for systems with 5 tasks, most of the cases are timed out. This is because the MILP solver CPLEX is unable to perform efficient incremental solving of the problems. Again, it demonstrates the benefit of designing a branching algorithm that takes advantage of the MUA-guided framework, as in the case of MUA-incremental.

Since MUA-MILP is unable to finish for most of the cases, we only compare the quality of solutions from MUA-incremental, MIGP and Minimum-single-speed

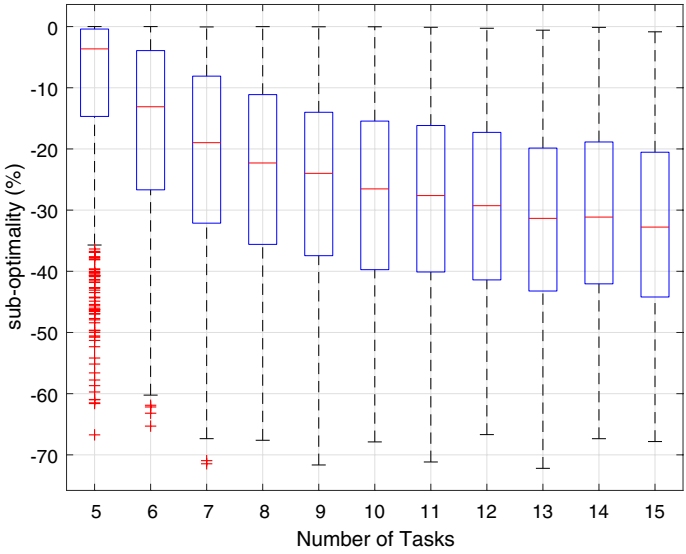


Fig. 7 Relative gap of MIGP compared to MUA-incremental

in terms of relative gap. For each random system, we define the relative gap of MIGP (or Minimum-single-speed) with respect to MUA-incremental as the relative difference in the objective values. Since MUA-incremental always provides a better objective value than the other two, we define the relative gap of MIGP (resp. Minimum-single-speed) as

$$s = \frac{p_A - p_B}{p_B} \times 100\% \tag{22}$$

where p_A is the objective value of MUA-incremental, and p_B represents that of MIGP (resp. Minimum-single-speed).

Figure 7 shows the whisker box plot of the distribution of the relative gap for MIGP compared to MUA-incremental. On average MUA-incremental finds 3% to 30% better solutions (i.e., with less energy consumption) than MIGP within the time limit. Likewise, Fig. 8 shows the distribution for Minimum-single-speed. As in the figure, MUA-incremental is on average 3% to 10% better than Minimum-single-speed.

Flight management system We next evaluate the techniques on an avionics case study consisting of a subset of Flight Management System application (Huang et al. 2014). The system contains 11 tasks of different criticality, which implement functions such as localization and flight planning. Each task is abstracted into an implicit deadline sporadic task characterized by a minimum inter-arrival time, a range of execution time that is typical in practice, and a criticality level. 7 tasks are of HI-criticality and the other 4 are of LO-criticality. The parameter configuration of the case study is summarized in Table 1.

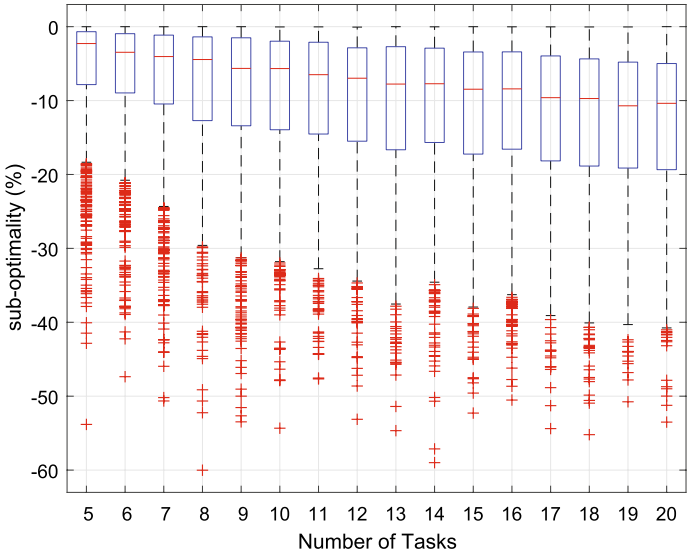


Fig. 8 Relative gap of Minimum-single-speed compared to MUA-incremental

Table 1 Flight management system case study Huang et al. (2014)

τ	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
T	5000	200	1000	1600	100	1000
$C(LO)$	[5, 40]	[5, 40]	[5, 40]	[5, 40]	[5, 40]	[5, 40]
HI/LO	HI	HI	HI	HI	HI	HI
τ	τ_7	τ_8	τ_9	τ_{10}	τ_{11}	
T	1000	1000	1000	1000	1000	
$C(LO)$	[5, 40]	[50, 400]	[50, 400]	[50, 400]	[50, 400]	
HI/LO	HI	LO	LO	LO	LO	

We consider fixed-priority uniprocessor scheduling according to Adaptive-Mixed-Criticality (AMC) for these tasks. For schedulability analysis, AMC-max has much higher computational complexity comparing to AMC-rtb, but it is more accurate and may help find better quality solutions when used in optimization. In the following, we consider the problem of minimizing LO-criticality energy consumption (Huang et al. 2014) given by Eq. (9). The range of the LO-criticality WCET $C_i(LO)$ of each task τ_i is determined as follows. We first take the upper-bound C_i^u of the execution time range given in the case study. Then we consider $C_i(LO)$ to be freely adjustable in the interval $[\frac{C_i^u}{4}, 2C_i^u]$ by CPU clock rate adjustment. For HI-criticality task, $C_i(HI)$ is obtained by scaling $C_i(LO)$ by a fixed criticality factor γ, i.e. $C_i(HI) = \gamma C_i(LO)$. In the experiments, we vary γ to take three possible values 3, 4, 5. The optimization problem is to find a $C_i(LO)$ for each task that minimizes Eq. (9).

We compare the following four methods:

Table 2 Results on flight management system by optimizing Task WCETs

Method	$\gamma = 3$		$\gamma = 4$		$\gamma = 5$	
	Time	Objective	Time	Objective	Time	Objective
MUA-MILP-AMCrtb	$\geq 48h$	N/A	$\geq 48h$	N/A	$\geq 48h$	N/A
MIGP-AMCrtb	$\geq 48h$	6.216e+007	$\geq 48h$	8.379e+007	$\geq 48h$	1.076e+008
MUA-incremental-AMCrtb-K5	0.05s	4.691e+007	0.04s	6.220e+007	0.03s	8.459e+007
MUA-incremental-AMCmax-K5	0.04s	4.691e+007	0.05s	6.220e+007	0.03s	8.459e+007
MUA-incremental-AMCrtb-K500	2.40s	4.129e+007	2.04s	5.840e+007	1.76s	7.732e+007
MUA-incremental-AMCmax-K500	3.18s	4.129e+007	2.64s	5.840e+007	2.14s	7.732e+007
MUA-incremental-AMCrtb-K50000	873.91s	4.108e+007	582.06s	5.803e+007	409.76s	7.723e+007
MUA-incremental-AMCmax-K50000	903.10s	4.108e+007	619.86s	5.803e+007	406.59s	7.723e+007

- MIGP-AMCrtb: Formulating the problem with AMC-rtb analysis as a mixed integer geometric program, and use the gpposy and YALMIP solvers to solve it. The value of K, i.e., the maximum number of nodes in each iteration is set to 50000.
- MUA-MILP-AMCrtb: The iterative framework in Fig. 2, where the problem ω is directly solved as an MILP program, and the schedulability analysis uses AMC-rtb.
- MUA-incremental-AMCrtb-K[K]: The framework in Fig. 2, where problems ω are solved with the MUA-driven branching tree, and the schedulability test is AMC-rtb. The suffix "-K[K]" represents the value of K, and we consider 3 different values 5, 500 and 50000.
- MUA-incremental-AMCmax-K[K]: The same as MUA-incremental-AMCrtb, except that the schedulability analysis uses AMC-max.

We let each approach run for 48 hours. Note that we do not consider AMC-max for MIGP as the analysis is too complicated to formulate in the MIGP framework. Likewise, we do not apply AMC-max to MUA-MILP, since it is already very slow under AMC-rtb: it cannot find any feasible solutions in 48 hours.

We first assume priority assignment is given by rate monotonic (ties are broken by criticality level). The results are summarized in Table 2. For $\gamma = 3$, MIGP-AMCrtb gets stuck in a suboptimal solution (about 51% worse than the optimal) early and cannot find any better solution even after 48 hours. MUA-MILP-AMCrtb is even worse, as it cannot find any feasible solution in 48 hours. Comparably, MUA-incremental performs much better: regardless of whether AMC-max or AMC-rtb is used, it finds the best solution in about 15 minutes for $K = 50000$, or about 200 times faster than MUA-MILP-AMCrtb and MIGP-AMCrtb. It can also be observed that the settings of $K = 500, 500000$ have an obviously better quality in solution compared with $k = 5$. While the solution with $K = 50000$ is marginally better than

Table 3 Results on Flight Management System by Co-optimizing Task priority assignments and WCETs

Method	$\gamma = 3$		$\gamma = 4$		$\gamma = 5$	
	Time	Objective	Time	Objective	Time	Objective
MUA-MILP-AMCrtb	$\geq 48h$	N/A	$\geq 48h$	N/A	$\geq 48h$	N/A
MUA-incremental-AMCrtb-K5	0.14s	2.666e+007	0.09s	3.056e+007	0.09s	3.762e+007
MUA-incremental-AMC-max-K5	0.16s	2.666e+007	0.16s	2.666e+007	0.15s	2.666e+007
MUA-incremental-AMCrtb-K500	12.16s	2.629e+007	10.20s	2.835e+007	6.55s	3.439e+007
MUA-incremental-AMC-max-K500	16.71s	2.630e+007	15.85s	2.629e+007	14.13s	2.629e+007
MUA-incremental-AMCrtb-K50000	2356.61s	2.626e+007	1204.74s	2.832e+007	517.68s	3.415e+007
MUA-incremental-AMC-max-K50000	2734.41s	2.626e+007	2260.71s	2.626e+007	2569.84s	2.626e+007

that with $K = 500$ give the same solution quality, the runtime is much longer for $K = 50000$. This suggests that while increasing the value of K may help find better solution, setting it to be excessively large brings marginal benefit but may significantly increase runtime.

The cases of $\gamma = 4$ and $\gamma = 5$ are similar. Note that the use of the more accurate AMC-max analysis did not improve the quality of the solution. This is mainly due to the rate-monotonic priority assignment. For this case, LO-criticality tasks are mostly lower in priorities than HI-criticality tasks (except τ_1, τ_4 which have quite loose deadlines). As a result, most HI-criticality tasks only suffer interference from other HI-criticality tasks. Therefore, the worst-case scenario for the response time calculation occurs when the system is entirely in the HI-criticality mode, where AMC-max and AMC-rtb give the same response time. On the other hand, the small difference in the runtimes of MUA-incremental-AMCmax and MUA-incremental-AMCrtb demonstrates that the efficiency of our approach is relatively insensitive to the complexity of the schedulability analysis.

We next consider the setting where priority assignment is not given and need to be co-optimized with WCETs. We omit MIGP-AMCrtb from this experiment as it is no longer applicable. Note that for methods based on the proposed optimization framework (MUA-incremental-AMCrtb-K[K] and MUA-incremental-AMCmax-K[K]), including priority assignment in the design space is rather simple: since both AMC-rtb and AMC-max are both compatible with Audsley's algorithm (Guo et al. 2017), in the framework we just use a schedulability analysis procedure that incorporates both the response time calculation (either AMC-rtb or AMC-max) and Audsley's algorithm for finding a feasible priority assignment.

The results are summarized in Table 3. A number of observations can be made. First, much better solutions are found when priority assignment is treated as a decision variable. The overall objective values reduce by as much as more than half comparing to the results in Table 2. This shows the benefit of the proposed optimization

framework: it is able to handle the co-optimization of various variables. Second, the difference between AMC-rtb and AMC-max analyses is now more noticeable. For example, when the criticality factor $\gamma = 5$, the use of AMC-max provides a solution that is 23% better than that of AMC-rtb. This demonstrates the benefit of using a more accurate analysis for optimization. However, such a benefit can only be achieved when the optimization algorithm is capable of accommodating the analysis. This is difficult for MIGP as AMC-max is too complicated to formulate in geometric programming framework. Our framework, on the other hand, can incorporate any schedulability analysis as long as it is sustainable. Third, the runtime noticeably increases comparing to Table 2. This is mainly because the schedulability analysis now incorporates Audsley's algorithm, which has a substantially higher computation complexity than just a plain response time calculation. Lastly, the difference in solution quality for different settings of K is even smaller than that in the previous experiment. The only noticeable difference is when $\gamma = 5$ and between MUA-incremental-AMCrtb-K5 and MUA-incremental-AMCrtb-K500. This is mostly because schedulability constraints become easier to satisfy when priority assignment can be co-optimized, especially for settings that use the AMC-max analysis. This suggests that the value of K can also be set by considering how constrained the problem is.

7.2 Control performance

In this experiment, we consider the problem of optimizing control performance for a set of periodic tasks scheduled on a uniprocessor. The problem was originally introduced in Mancuso et al. (2014), where the objective is formulated in Eq. (8). Note that the objective function also satisfies the two properties discussed in Sect. 5.4, and thus can benefit from Algorithm 2.

Mancuso et al. (2014) proposed to relax the response time analysis (2) by removing the ceiling operator (hence it uses $\frac{R_i}{T_j}$ to estimate the number of interferences from task τ_j on task τ_i). Although the relaxation was claimed to be a close approximation to the exact response time in practice, it is possible that the relaxed analysis may give unsafe solutions. In fact, in the randomly generated systems below, the solution returned by Mancuso et al. (2014) is always unschedulable with the exact analysis in Eq. (2). Thus, in the comparison of other methods below, we use the exact analysis (2).

We evaluate on randomly generated synthetic task sets. The parameters are generated using a similar setting as Mancuso et al. (2014). Specifically, in Equation (7), α_i is randomly generated in the range [1, 1000], β_i is randomly generated in the range [1, 10000], and the WCET of each task C_i is randomly selected from [1, 100]. The upper-bound for the task period T_i is set to be 5 times the sum of all task WCETs, i.e., $T_i^u = 5 \sum_{j=1}^{n} C_j$. This sets 20% as the lower bound on system utilization. Task priority are assigned with the rate monotonic policy.

We compare the following three methods:

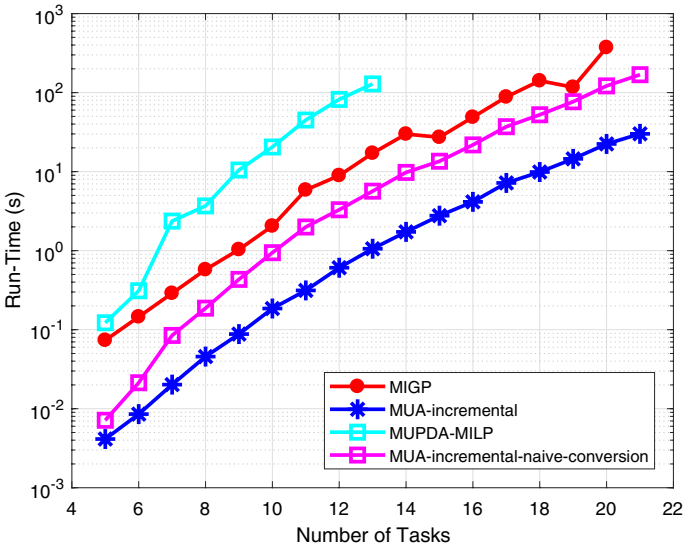


Fig. 9 Runtime of optimizing control performance

- MIGP: MIGP formulation proposed in Davare et al. (2007) for optimizing task periods. We use geometric programming solver gpposy (Mutapcic et al. 2006) with the BnB (bmi) solver in YALMIP (Löfberg 2004) for solving MIGP problems.
- MUPDA-MILP: The proposed optimization framework in Zhao et al. (2018), which is an iterative procedure that leverages CPLEX solver (International Business Machines Corporation xxxx) to solve a series of MILP problems.
- MUA-incremental: The proposed optimization framework in Fig. 2, where the problem ω is solved using the MUA-driven branching tree. Note that the method uses Algorithm 2 for MUA conversion. The same result is presented in the original conference paper (Zhao et al. 2020). The algorithm however, was not discussed due to space limitation.
- MUA-incremental-naive-conversion: The same as MUA-incremental but using Algorithm 1 for MUA conversion.

Like in Sect. 7.1, we set a maximum iteration limit of 2000 for MIGP, and the time limits of the other two methods are set to 600 seconds for each problem instance. The size limit for MUA-incremental is set to $K = 10000$.

Figure 9 plots the runtime of all the optimization methods MIGP, MUPDA-MILP and MUA-incremental. While all methods give the same optimal solution when finishing within the time limit, MUPDA-MILP has the worst scalability. As discussed in Zhao et al. (2018), this confirms that it is often inefficient in solving problems with objective that are sensitive to many decision variables, since it takes many iterations to terminate, and each MILP problem has to be solved from scratch (instead of incrementally as in our approach). Meanwhile, MUA-incremental runs about

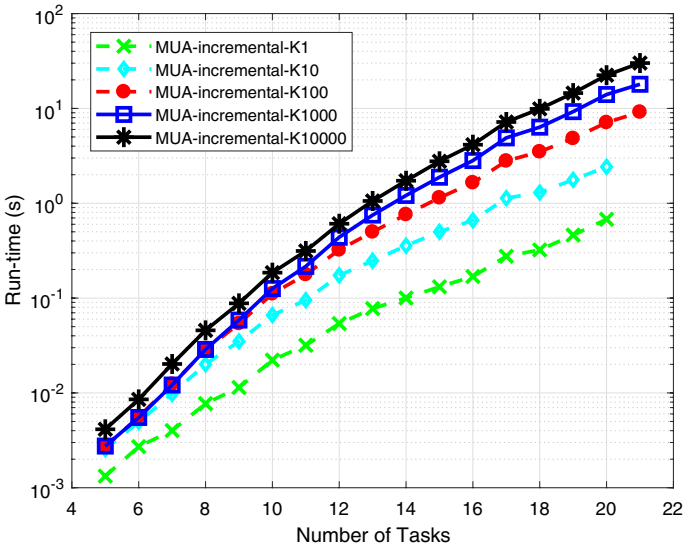


Fig. 10 Runtime by different settings of K

10 times faster than MIGP. This again demonstrates the advantage of our proposed framework that judiciously combines the MUA-guided iterative procedure with the incremental update of the branching tree. MUA-incremental is also several times faster than MUA-incremental-naive-conversion, which confirms that Algorithm 2 can improve the convergence speed of the optimization framework.

We next study the effect of K on algorithm efficiency and quality of solution. We try four different values $K = 1, 10, 100, 1000$. The corresponding configurations are denoted as MUA-incremental-K1, MUA-incremental-K10, MUA-incremental-K100 and MUA-incremental-K1000 respectively. Figure 10 shows the average runtime by different configurations on K. As in the figure, increasing the value of K increases the run-time: From $k = 1$ to $k = 10000$, the run time increase by more than 2 orders of magnitude.

We now compare the solution quality for different settings of K. We use MIGP as a reference, since when solve to optimality, it returns the globally optimal solution. Specifically, let p_1, p_2 denote the objective value by MIGP and MUA-incremental-K[K] respectively, then the sub-optimality of MUA-incremental-K[K] is evaluated as $s = \frac{\max\{p_2 - p_1, 0\}}{p_1}$. Note MIGP may not be able to finish within the time limit and give a worse solution than MUA-incremental-K[K], and s essentially only considers the cases where MIGP outperforms MUA-incremental-K[K]. Figure 11 gives the whisker box plot of sub-optimality for different K. When $K = 1$, the sub-optimality of MUA-incremental-K1 can be as large as 7%. As K increases, the maximum sub-optimality becomes smaller. For $K = 10000$, we only observe 3 cases where the framework obtains worse solution than MIGP, and the worst sub-optimality is only 0.08%.

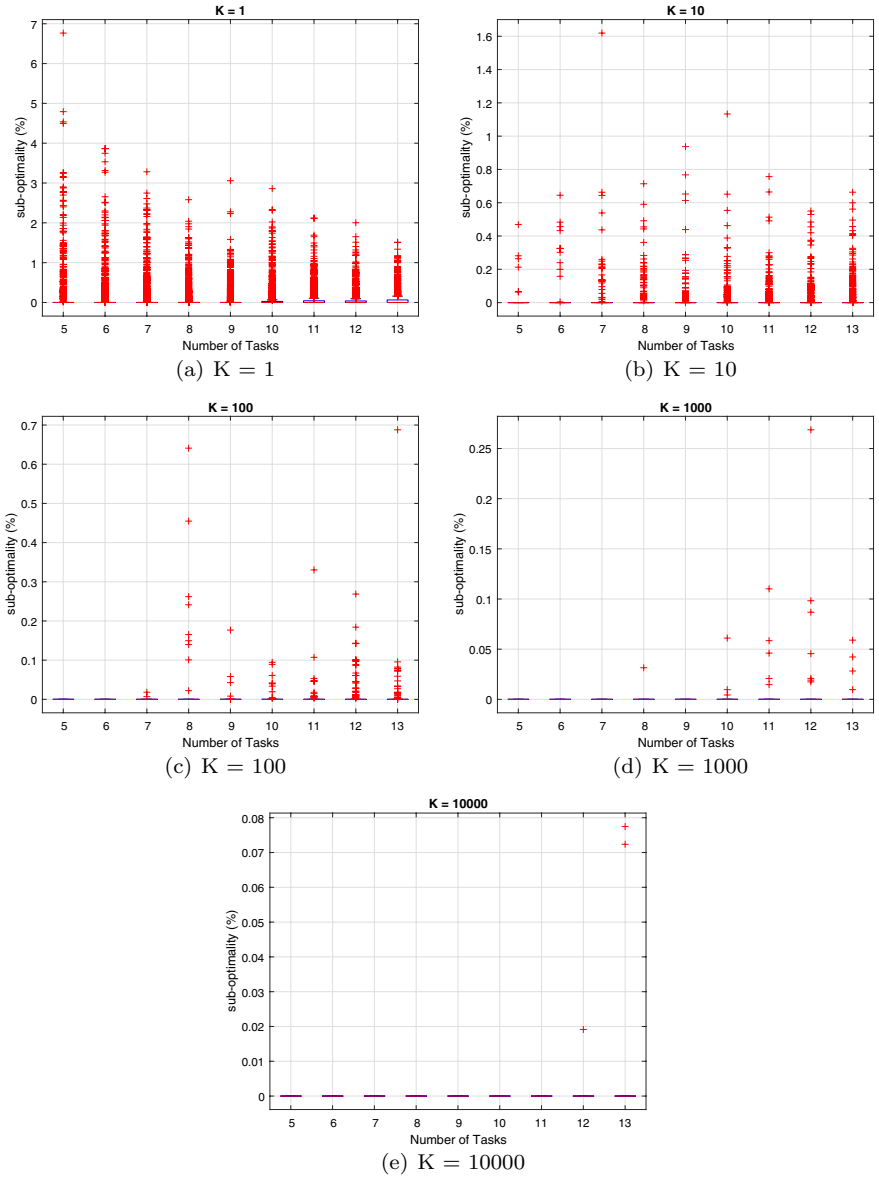


Fig. 11 Sub-optimality for different settings of K

In the next experiment, we remove the assumption that priority assignment is given and allows it to be co-optimized instead. For this problem, it is optimal by just combining MUA-incremental with the deadline monotonic priority assignment policy, where task deadlines are returned by solving step 1. This is optimal since we assume tasks have constrained deadlines and the response time analysis follows Equation (2).. This approach is denoted as MUA-incremental-PA. Since

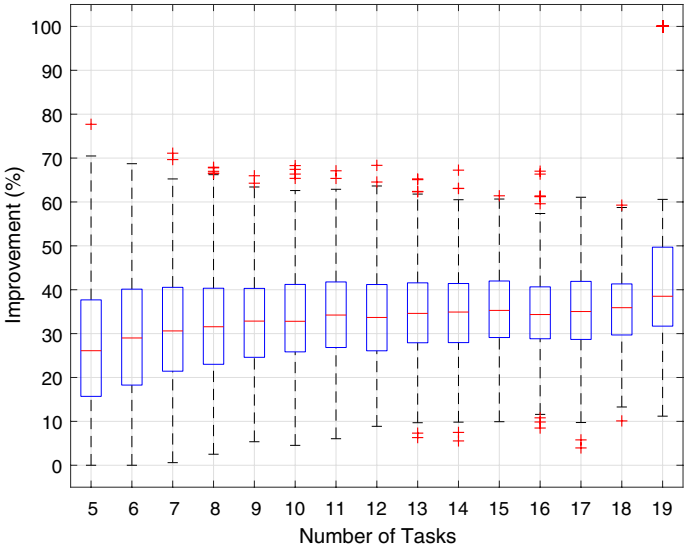


Fig. 12 Improvement in solution quality by co-optimizing priority assignment

MIGP is no longer applicable, we evaluate the improvement in solution quality brought by co-optimizing priority assignment. Let p_1, p_2 denote the objective value by MUA-incremental and MUA-incremental-PA respectively. The improvement is calculated as $s = \frac{\max\{p_1 - p_2, 0\}}{p_1}$.

Figure 12 gives a whisker box plot of the distribution of improvement. Co-optimizing priority assignment gives an average of 30% improvement in solution quality. In quite a number of cases, the improvement can be as high as 70%. This demonstrates the benefit of the flexibility of the proposed framework, which is usually difficult to achieved with traditional mathematical programming framework. Figure 13 compares its average runtime with MUA-incremental. The increased runtime is mainly due to the additional decision space from the extra design variables of task priority assignments, which requires more iterations to solve.

8 Future work

As shown earlier, the performance by the proposed framework can be heavily impacted by how the MUA-driven branching tree is built and searched. In this paper, we introduce a rather simple solution of using fixed size limit parameters K and only adding/searching at most K nodes with the best objective values. Some potential problems for further investigation include (1) how K should be selected, (2) can K be adaptively adjusted from iteration to iteration, and (3) is there another way to choose the K nodes that gives better performance. Finding the best answers to these question is generally difficult and would almost certainly vary depending

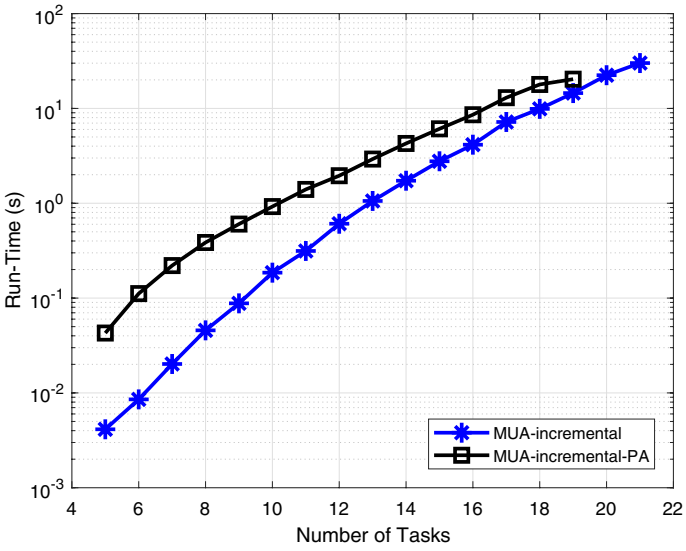


Fig. 13 Comparison of run time w/wo co-optimizing priority assignment

on the optimization problem. However, this is where learning based technique may be helpful. For example, when the algorithm terminates, a separate step can be performed to revisit the branching path and evaluate how decisions in different iteration have contributed to the final solution. A simple question would be if the same path, and thus the solution, can be reached using a different, perhaps smaller K in different iterations. This process can be repeated over a large number of randomly generated systems and eventually have the algorithm learn a policy that works effectively for this particular problem formulation.

So far we have only consider problems where schedulability is the only constraint. However, the system may often times include additional constraints, such as end-to-end deadline (Davare et al. 2007), memory limitations (Ferrari et al. 2009), and so on. Usually these constraints may be conflicting with the schedulability requirement. For example, while higher task period helps schedulability, it may increase the end-to-end latency. How the framework can be adapted to accommodate additional constraints is another topic for further investigation. Some possible options include taking into account the constraints while constructing new layer of branching tree nodes, or modelling the constraints as an additional cost in the objective function.

9 Conclusion

In this paper, we propose a framework for optimizing the design of real-time system with sustainable schedulability analysis. We propose the concept of Maximal-Unschedulable-Assignment (MUA) and show how it can be used to abstractly interpret the schedulability constraints. Based on the concept, we develop an

iterative optimization procedure that uses MUAs to iteratively refine the schedulability region. It contains three key steps: (i) an algorithm for solving the optimization problem consisting of MUA-implied constraints, (ii) an algorithm for computing MUAs that leads to faster convergence, and (iii) use of MUAs to exploring good-quality schedulable solutions. We perform experiments on two optimization problems with different settings to demonstrate the advantage of the proposed approach in applicability and scalability. This paper extends the original RTSS 2020 conference version (Zhao et al. 2020) with the discussion of an improved MUA conversion algorithm, and new experiments to evaluate its effectiveness and the impact of different settings of K for building the MUA-driven branching tree.

Acknowledgements This work is partially funded by NSF Grants No. 1812963 and No. 1837519.

References

Al-Bayati Z, Sun Y, Zeng H, Di Natale M (2015) Zhu, Q., Meyer, B.: Task placement and selection of data consistency mechanisms for real-time multicore applications. In: IEEE Real-Time and Embedded Technology and Applications Symposium, pp. 172–181. IEEE

Audsley N (2001) On priority assignment in fixed priority scheduling. Informat Proc Lett 79(1):39–44

Audsley NC, Burns A, Richardson MF, Wellings AJ (1991) in proc. ieee workshop on real-time operating systems and software. In: real-time scheduling: the deadline-monotonic approach, pp. 133–137

Baker T, Baruah S (2009) Sustainable multiprocessor scheduling of sporadic task systems. In: Euromicro Conference on Real-Time Systems

Bambagini M, Marinoni M, Aydin H, Buttazzo G (2016) Energy-aware scheduling for real-time systems: a survey. ACM Transact Embedded Comput Sys (TECS) 15(1):1–34

Bansal S, Zhao Y, Zeng H, Yang K (2018) Optimal implementation of simulink models on multicore architectures with partitioned fixed priority scheduling. In: IEEE Real-Time Systems Symposium, pp. 242–253. IEEE

Baruah S, Burns A (2006) Sustainable scheduling analysis. In: 27th IEEE Real-Time Systems Symposium

Baruah S, Burns A, Davis R (2011) Response-time analysis for mixed criticality systems. In: IEEE Real-Time Systems Symposium

Bate I, Emberson P (2006) Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In: IEEE Real-Time and Embedded Technology and Applications Symposium

Bini E, Di Natale M (2005) Optimal task rate selection in fixed priority systems. In: 26th IEEE Real-Time Systems Symposium

Boyd S, Vandenberghe L (2004) Convex optimization. Cambridge University Press, Cambridge

Davare A, Zhu Q, Di Natale M. Pinello C, Kanajan S (2007) Sangiovanni-Vincentelli, A.: Period optimization for hard real-time distributed automotive systems. In: ACM/IEEE Design Automation Conference

Davare A, Zhu Q, Di Natale M, Pinello C, Kanajan S, Sangiovanni-Vincentelli A (2007) Period optimization for hard real-time distributed automotive systems. In: Design Automation Conference

Davis R, Burns A (2009) Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In: 30th IEEE Real-Time Systems Symposium

Davis RI, Cucu-Grosjean L, Bertogna M, Burns A (2016) A review of priority assignment in real-time systems. J Sys Architec 65:64–82

Ebert C, Favaro J (2017) Automotive software. IEEE Soft 34(3):33–39

Ekberg P, Yi W (2015) Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly conp-complete. In: 27th Euromicro Conference on Real-Time Systems, pp. 281–286

Ekberg P, Yi W (2017) Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard. In: IEEE Real-Time Systems Symposium

Ferrari A, Di Natale M, Gentile G, Reggiani G, Gai P (2009) Time and memory tradeoffs in the implementation of autosar components. In: 2009 Design, Automation & Test in Europe Conference & Exhibition, pp. 864–869. IEEE

Gu Z, Han G, Zeng H, Zhao Q (2016) Security-aware mapping and scheduling with hardware co-processors for flexray-based distributed embedded systems. IEEE Transac Parallel Distributed Sys 27(10):3044–3057

Guo Z, Sruti S, Ward BC, Baruah S (2017) Sustainability in mixed-criticality scheduling. In: IEEE Real-Time Systems Symposium

Hamann A, Jersak M, Richter K, Ernst R (2004) Design space exploration and system optimization with symta/s - symbolic timing analysis for systems. In: IEEE Real-Time Systems Symposium

Han G, Zeng H, Di Natale M, Liu X, Dou W (2013) Experimental evaluation and selection of data consistency mechanisms for hard real-time applications on multicore platforms. IEEE Transac Indus Informat 10(2):903–918

Hassanalian M, Abdelkefi A (2017) Classifications, applications, and design challenges of drones: A review. Prog Aerospace Sci 91:99–131

Huang P, Giannopoulou G, Stoimenov N, Thiele L (2014) Service adaptions for mixed-criticality systems. In: IEEE Asia and South Pacific Design Automation Conference

Huang P, Kumar P, Giannopoulou G, Thiele L (2014) Energy efficient dvfs scheduling for mixed-criticality systems. In: ACM Conference on Embedded Software

International Business Machines Corporation: CPLEX Optimizer

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20(1):46–61. https://doi.org/10.1145/321738.321743

Löfberg J (2004) Yalmip : A toolbox for modeling and optimization in matlab. In: IEEE Symposium on Computer Aided Control Systems Design

Mancuso G, Bini E, Pannocchia G (2014) Optimal priority assignment to control tasks. ACM Trans Embedded Comput Syst 13(5s):161

Mutapcic A, Koh K, Kim S, Boyd S (2006) Ggplab version 1.00: a matlab toolbox for geometric programming

Nelson A, Moreira O, Molnos A, Stuijk S, Nguyen BT, Goossens K (2011) Power minimisation for real-time dataflow applications. In: Euromicro Conference on Digital System Design

Pagani S, Chen JJ (2014) Energy efficiency analysis for the single frequency approximation (sfa) scheme. ACM Transact Embedded Comput Syst (TECS) 13(5s):158

Saksena M, Wang Y (2000) Scalable real-time system design using preemption thresholds. In: IEEE Real-Time Systems Symposium

Seto D, Lehoczky JP, Sha L (1998) Task period selection and schedulability in real-time systems. In: 19th IEEE Real-Time Systems Symposium

Shin M, Sunwoo M (2007) Optimal period and priority assignment for a networked control system scheduled by a fixed priority scheduling system. Int J Automotive Technol 8:39–48

Tindell K, Burns A, Wellings A (1992) Allocating hard real-time tasks: An np-hard problem made easy. Real-Time Syst. 4(2):145–165

Tripakis S, Lublinerman R (2018) Modular code generation from synchronous block diagrams: Interfaces, abstraction, compositionality. In: Principles of Modeling, pp. 449–477. Springer

Wang C, Gu Z, Zeng H (2016) Global fixed priority scheduling with preemption threshold: Schedulability analysis and stack size minimization. IEEE Trans. Parallel and Distributed Sys 27(11):3242–3255

Wang Y, Saksena M (1999) Scheduling fixed-priority tasks with preemption threshold. In: International Conference on Real-Time Computing Systems and Applications

Zeng, H., Di Natale, M.: Mechanisms for guaranteeing data consistency and flow preservation in autosar software on multi-core platforms. In: IEEE Symposium on Industrial and Embedded Systems (2011)

Zeng H, Di Natale M (2012) Efficient implementation of autosar components with minimal memory usage. In: 7th IEEE Symposium on Industrial Embedded Systems

Zhao Y, Gala V, Zeng H (2018) A unified framework for period and priority optimization in distributed hard real-time systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37(11):2188–2199

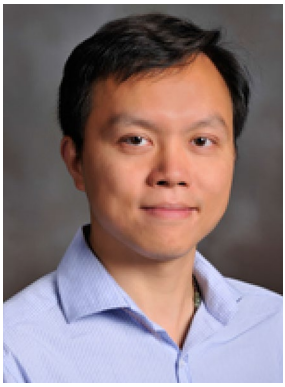
Zhao Y, Zeng H (2017) The concept of unschedulability core for optimizing priority assignment in real-time systems. In: Conference on Design, Automation and Test in Europe

Zhao Y, Zeng H (2017) The virtual deadline based optimization algorithm for priority assignment in fixed-priority scheduling. In: IEEE Real-Time Systems Symposium

Zhao Y, Zeng H (2018) The concept of response time estimation range for optimizing systems scheduled with fixed priority. In: IEEE Real-Time and Embedded Technology and Applications Symposium (2018)
Zhao Y, Zeng H (2018) The concept of unschedulability core for optimizing real-time systems with fixed-priority scheduling. IEEE Transact Comput 68(6):926–938
Zhao Y, Zeng H (2019) The concept of maximal unschedulable deadline assignment for optimization in fixed-priority scheduled real-time systems. Real-Time Syst 55(3):667–707
Zhao Y, Zhou R, Zeng H (2020) An optimization framework for real-time systems with sustainable schedulability analysis. In: 2020 IEEE Real-Time Systems Symposium (RTSS), pp. 333–344. IEEE

Yecheng Zhao received his B.E. in Electrical Engineering from Harbin Institute of Technology, Harbin, China, and PhD degree in Computer Engineering from Virginia Tech. His main research focus is design optimization for real-time embedded systems. He is current working as a software engineer with Google Inc on secure firmware and OS.

Runzhi Zhou is a master student in Electrical Engineering at University of Pennsylvania School of Engineering and Applied Science. He received a bachelor's degree in Computer Science and Electrical Engineering from Case Western Reserve University. His research interests include real-time system and optimization.



Haibo Zeng is with Department of Electrical and Computer Engineering at Virginia Tech, USA. He received his Ph.D. in Electrical Engineering and Computer Sciences from University of California at Berkeley. He was a senior researcher at General Motors R&D until October 2011, and an assistant professor at McGill University until August 2014. His research interests are embedded systems, cyber-physical systems, and real-time systems. He received five paper awards in the above fields.