Real-Time Systems (2019) 55:667–707
https://doi.org/10.1007/s11241-019-09332-0

The concept of Maximal Unschedulable Deadline Assignment for optimization in fixed-priority scheduled real-time systems

Yecheng Zhao[1] · **Haibo Zeng**[1]

Published online: 26 February 2019

Abstract
This paper considers the problem of design optimization for real-time systems scheduled with fixed priority, where task priority assignment is part of the decision variables, and the timing constraints and/or objective function linearly depend on the exact value of task response times (such as end-to-end deadline constraints). The complexity of response time analysis techniques makes it difficult to leverage existing optimization frameworks and scale to large designs. Instead, we propose an efficient optimization framework that is three orders of magnitude (1000 times) faster than Integer Linear Programming (ILP) while providing solutions with the same quality. The framework centers around three novel ideas: (1) an efficient algorithm that finds a schedulable task priority assignment for minimizing the average worst-case response time; (2) the concept of Maximal Unschedulable Deadline Assignment (MUDA) that abstracts the schedulability conditions, i.e., a set of maximal virtual deadline assignments such that the system is unschedulable; and (3) a new optimization procedure that leverages the concept of MUDA and the efficient algorithm to compute it.

Keywords Average worst-case response time · Maximal Unschedulable Deadline Assignment (MUDA) · Schedulability condition abstraction · Design optimization

1 Introduction

The design optimization of real-time systems is to find, at design time, a suitable design candidate that is (a) *predictably correct*, i.e., with design-time guarantees that

✉ Haibo Zeng
hbzeng@vt.edu

Yecheng Zhao
zyecheng@vt.edu

[1] Virginia Polytechnic Institute and State University, 302 Whittemore (0111), Virginia Tech, Blacksburg, VA 24061, USA

requirements on critical metrics (such as timing, control quality, and memory) are satisfied; and (b) (optionally) *optimal* with respect to a given optimization objective function. In this paper, we focus on real-time systems with fixed priority scheduling. We consider the problems where the task priority assignment is part of the decision variables, and the task worst-case response times (WCRTs) are involved in the design constraints and/or objective function.

We discuss a few application scenarios. The *first* is the direct minimization of the average WCRT over (a subset of) the tasks (Arzén et al. (2000); Samii et al. (2009)), i.e., the system performance is measured by the promptness of task completion given that task schedulability is guaranteed. The *second* is the design of control systems, where the control error is approximately proportional to the response time of the controller task (Lincoln and Cervin 2002; Bini and Cervin 2008). The *third* is that modern cyber-physical systems are characterized by complex functionality deployed on a distributed platform, where timing constraints and performance metrics are expressed on end-to-end paths (Davare et al. 2007). The worst-case end-to-end delay for each time-critical path equals the sum of the WCRTs and periods of all tasks and messages in the path. Examples include active safety features in automotive that spans over several Electronic Control Units (ECUs) connected by communication buses such as Controller Area Network (CAN).

Since the subproblem of task WCRT calculation is shown to be NP-hard (Eisenbrand and Rothvoß 2008), the overall problem complexity is NP-hard. A straightforward approach is to formulate the design optimization as a mathematical program. This approach may be appealing since modern mathematical program solvers, such as CPLEX for integer linear programs (ILP), incorporate many sophisticated techniques to efficiently prune the search space, and are typically much better than plain branch-and-bound. However, despite the efficiency of modern solvers, it is still very difficult to scale to large scale industrial systems.

Surprisingly, our framework runs 1000 times faster than CPLEX while maintaining the solution quality. The framework is carefully crafted based on a few problem-specific insights that are difficult to match for generic constraint solvers such as CPLEX. The first is that the WCRT calculation, although NP-hard, is actually very efficient in practice (Davis et al. 2008). However, the corresponding ILP formulation requires a possibly large number of integer variables (in the order of $O(n^2)$, where n is the number of tasks) (Zeng and Di Natale 2013). The second is that there are algorithms which can efficiently find a schedulable priority assignment if one exists, such as rate monotonic policy for periodic tasks with preemptive scheduling (Liu and Layland 1973) or Audsley's Algorithm for many task models (Audsley 2001), both of which run in polynomial time to the number of tasks. The intelligence in such problem specific algorithms, carefully studied by real-time systems experts (e.g., Audsley 2001; Davis and Burns 2007, 2009), may not be captured in solvers like CPLEX.

Hence, we propose an optimization framework that judiciously leverage the power of commercial ILP solver (for generic branch-and-bound based search) and develop a problem-specific algorithm (to efficiently find the optimal solution for a subproblem). We establish an abstraction layer that hides the detail of WCRT analysis from the ILP solver but still faithfully respects its accuracy. We envision such a drastically improved optimization capability will have profound impacts for the considered class of real-

time systems. For example, it may enable a new, agile and fluid design flow in which the designers can interact with the optimization tools.

Specifically, our work makes the following contributions.

- We study the problem of minimum average WCRT and show that it has an efficient optimal solution.
- We leverage the above algorithm, and introduce the concepts of virtual deadline and Maximal Unschedulable Deadline Assignment (MUDA), the latter is a set of maximal virtual deadlines for individual tasks and weighted sum of task WCRTs such that the task system is unschedulable.
- We devise an optimization framework based on the concept of MUDA, which prudently combines the efficient algorithm for calculating MUDA and ILP for generic branch-and-bound.
- We apply our optimization framework to two industrial case studies and show that the proposed technique runs 1000 $\times$ faster over standard approach while maintaining optimality.

The rest of the paper is organized as follows. Section 2 summarizes the task model and optimization problem considered in this paper. Sections 3 and 4 study the problem of minimizing average WCRT and its weighted version, respectively. Section 5 introduces the concepts of virtual deadline and MUDA, while Sect. 6 presents the optimization framework built upon it. Section 7 discusses the applicability of the framework to other optimization objectives. Section 8 presents the experimental results. Section 9 discusses the related work. Finally, Sect. 10 concludes the paper.

Comparing to the conference version published at RTSS 2017 (Zhao and Zeng 2017b), this paper contains the following extensions.

- In Sect. 3, we also consider the accurate schedulability analysis for non-preemptive scheduling. We show that in this case WCET Monotonic Strategy is not optimal by providing a counterexample in Table 1.
- In Sect. 4, we prove that under a certain condition Scaled-WCET Monotonic is guaranteed to be optimal (Theorem 8). Also, we provide an example (in Table 3) showing that Scaled-WCET Monotonic is not optimal when the condition is violated. We then add Algorithms 2 and 4. These two algorithms, together with Algorithm 3, explain the details of the sifting adjustment. We also demonstrate how the algorithms work using the counterexample provided in Table 3.
- We add a new Sect. 7, where we discuss the applicability of the proposed optimization framework to other optimization objectives. Specifically, we present a family of objectives known as maximizing minimum laxity that the proposed technique can also efficiently handle. Comparably, we discuss a type of objectives expressed in the form of linear summation of virtual deadlines, which the proposed technique may have difficulty with. We provide an in-depth analysis from different perspectives on the objectives with respect to the efficiency of our algorithms.
- We enhance the section on experimental evaluation as follows. In Sect. 8.3, we discuss how the memory constraints of the fuel injection system case study can be formulated as additional design constraints compatible with the proposed framework. We add a new set of experiments (Sect. 8.4), where we apply the proposed

technique on optimizing the newly discussed laxity based objectives for the experimental vehicle system case study, and compare it with a direct ILP formulation.

2 System model and notations

We consider a real-time system Γ containing a set of periodic or sporadic tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ with *constrained deadline*. Each task τ_i is characterized by a tuple $\langle C_i, T_i, D_i \rangle$, where C_i is its worst-case execution time (WCET), T_i is the minimal inter-arrival time, and $D_i \leq T_i$ is the deadline. Without loss of generality, we assume *these parameters are all positive integers*. Each task τ_i is assigned with a *fixed priority* π_i that is subject to the designers' decision. The higher the number π_i, the higher the priority. Hence, $\pi_i > \pi_j$ denotes that τ_i has a higher priority than τ_j. For convenience, we also denote $\tau_i > \tau_j$ if $\pi_i > \pi_j$. We consider two possible scheduling policies: *preemptive scheduling and non-preemptive scheduling*. These scheduling policies are widely adopted in practical systems, such as the automotive AUTOSAR/OSEK real-time operating systems (RTOS) standard, the modern RTOSes including LynxOS and QNX Neutrino, and the controller area network (CAN) protocol and its recent extension CAN-FD (CAN with flexible data-rate).

For preemptive scheduling, the worst-case response time (WCRT) of task τ_i is the smallest fixed point solution of the following formula (Tindell et al. 1994)

$$R_i = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{1}$$

where $hp(i)$ represents the set of higher priority tasks than τ_i.

For non-preemptive scheduling, the worst-case blocking time τ_i will suffer is the largest WCET among all lower priority tasks (Davis et al. 2007)

$$B_i = \max_{\forall \tau_j \in lp(i)} \{C_j\} \tag{2}$$

where $lp(i)$ represents the set of lower priority tasks than τ_i. The WCRT of τ_i is the largest among all instances in the busy period. The length of the busy period L_i for a task τ_i is given by the least fixed point of the following equation

$$L_i = B_i + \left\lceil \frac{L_i}{T_i} \right\rceil C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{L_i}{T_j} \right\rceil C_j \tag{3}$$

Hence, the number of jobs of τ_i in the busy period is

$$q_i^{\max} = \left\lceil \frac{L_i}{T_i} \right\rceil \tag{4}$$

We index these jobs as $q = 0, \ldots, q_i^{\max} - 1$. The waiting time (also called queuing delay) of the q-th job in the busy period, the longest time that it stays in the waiting

queue before starting execution, is denoted as $w_i(q)$. The worst-case response time $R_i(q)$ of the q-th job and $w_i(q)$ are calculated as

$$\begin{cases} w_i(q) = B_i + qC_i + \sum\limits_{\forall \tau_j \in hp(i)} \left\lceil \dfrac{w_i(q)}{T_j} \right\rceil C_j \\ R_i(q) = w_i(q) - qT_i + C_i \end{cases} \tag{5}$$

The WCRT of τ_i is calculated as

$$R_i = \max_{q=0,\ldots,q_i^{\max}-1} R_i(q) \tag{6}$$

The exact analysis in Eqs. (2)–(6) is particularly sophisticated in that the number of jobs $q_i^{\max}$ in the busy period is unknown a priori. Hence, it is difficult to design an optimization algorithm using such an analysis. In this paper, we adopt the following safe WCRT analysis for tasks with constrained deadline (Davis et al. 2007). Specifically, the waiting time of task τ_i is approximated as

$$w_i = \max(B_i, C_i) + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i}{T_j} \right\rceil C_j \tag{7}$$

Here B_i denotes the maximum blocking time from lower priority tasks. The analysis is based on the observation that τ_i can either suffer the blocking of a lower priority task or the push through interference from the previous job of the same task, but not both. For convenience, we denote

$$\tilde{B}_i = \max(B_i, C_i) = \max_{\forall j \in lp(i) \cup \{i\}} C_j \tag{8}$$

and call $\tilde{B}_i$ as the *adjusted blocking time of* τ_i.

Finally, the WCRT of a non-preemptive task τ_i can be bounded as

$$R_i = C_i + w_i \tag{9}$$

The analysis in Eqs. (7)–(9) is only sufficient but not necessary, in the sense that it may over-estimate the WCRT of a non-preemptive task. However, we adopt this analysis instead of the exact one, for two reasons. One is that its accuracy is very good in practice, especially for CAN (Di Natale and Zeng 2013). The other is that we can derive some nice properties from such an analysis to provide an efficient optimization algorithm (see Sect. 3).

We consider a design optimization problem where the design variables include the task priority assignment. As part of the feasibility constraints, the tasks are required to be schedulable. Besides, the problem also requires the precise information on the WCRTs of (a subset of) the tasks.

Formally, the problem can be expressed as the following mathematical programming problem

$$\begin{aligned} \min \quad & \sum_{\forall i \in \Omega} \beta_i \cdot R_i \\ \text{s.t.} \quad & \text{Tasks are schedulable} \\ & \mathbb{G}(\mathbf{X}) \leq 0 \end{aligned} \tag{10}$$

Here $\mathbf{X}$ represents the vector of decision variables that include the task priority assignment $\mathbf{P}$ and the task WCRTs $\mathbf{R}$. The objective function is a weighted sum of the task WCRTs, where $\beta_i \geq 0$ is the weight for task τ_i. $\mathbb{G}(\mathbf{X}) \leq 0$ is the set of linear constraints on $\mathbf{X}$ that the solutions in the feasibility region shall satisfy, in addition to the schedulability of each task. For convenience, we denote $\Omega = \{\tau_i : \beta_i > 0\}$, i.e., the set of tasks contributing to the objective. We assume that $\mathbb{G}(\mathbf{X})$ is non-decreasing with task WCRTs, i.e., it imposes an upper bound on the task WCRTs.

The problem in (10) is a suitable representation of a wide variety of applications. For example, the control cost in real-time control systems depends on the WCRTs of the control tasks, which can typically be linearized (Mancuso et al. 2014). The end-to-end delay of distributed features in automotive systems is the sum of WCRTs and periods for all tasks and messages in the path (Davare et al. 2007), which may be subject to end-to-end deadline (i.e., formulated in the constraint $\mathbb{G}(\mathbf{X}) \leq 0$) or serve as the objective function. In the experiments, we will illustrate with two industrial case studies formulated in the above form.

3 Minimizing average WCRT

In this section, we first study a specific instance of (10), known as the minimum average WCRT problem. In this case, the weights of all tasks are the same, and the constraints consist of only the system schedulability that all tasks meet their deadline. The mathematical form of the problem is expressed as follows

$$\begin{aligned} \min \quad & \sum_{\forall i \in \Omega} R_i \\ \text{s.t.} \quad & \text{Tasks are schedulable} \end{aligned} \tag{11}$$

where $\Omega \subseteq \Gamma$ is the subset of tasks included in the objective for minimizing average WCRT. In the next section, we generalize the result and design an algorithm for cases where tasks have different weights.

We find that there is a simple optimal algorithm for (11), as detailed in Algorithm 1. It only needs to explore a quadratic number $O(n^2)$ of priority orders out of the $n!$ possible ones, where n is the number of tasks. This algorithm is a revision of Audsley's algorithm (Audsley 2001) by augmenting it with a simple strategy termed as "*WCET Monotonic*". Similar to Audsley's algorithm, it iteratively assigns a task to a priority from the lowest level to the highest. At each priority level, it checks if there is any unassigned task in $\Gamma \backslash \Omega$ is schedulable (Lines 3–8). If yes, it assigns this task with the

Algorithm 1 Priority assignment to minimize average WCRT

```
1: function MinAvgWCRT(Task set Γ, Concerned set Ω)
2:   for each priority level k, from lowest to highest do
3:     for each unassigned τ_i in Γ\Ω do
4:       if τ_i is schedulable at priority level k then
5:         Assign priority k to τ_i
6:         Continue to the next priority level
7:       end if
8:     end for
9:     for each unassigned τ_i in Ω in non-increasing WCET order do
10:      if τ_i is schedulable at priority level k then
11:        Assign priority k to τ_i
12:        Continue to the next priority level
13:      end if
14:    end for
15:    return unschedulable
16:  end for
17:  return schedulable
18: end function
```

current priority level. Otherwise, it chooses the unassigned task in Ω with the *longest WCET* among those schedulable at this level (Lines 9–14).

In the following, we first provide a set of sufficient conditions under which Algorithm 1 is optimal. Consider a task system and an associated WCRT analysis M. Assume that M is *compliant with* Audsley's algorithm, i.e., it satisfies the following three conditions identified in Davis and Burns (2009).

- The WCRT R_i of any task τ_i calculated by M does not depend on the relative order of tasks in $hp(i)$;
- Similarly, the calculation of R_i by M does not depend on the relative order of tasks in $lp(i)$;
- R_i is monotonically increasing with $hp(i)$, i.e., if τ_i is dropped to a lower priority (and hence $hp(i)$ becomes larger) while the relative order of other tasks remains the same, R_i will only increase.

Now let τ_s and τ_l be any two tasks such that $C_l \geq C_s$. Namely, τ_l is the long task and τ_s is the short task. Consider any *feasible* priority order in which τ_l has higher priority than τ_s. The main idea in the following theorem is to ensure that swapping the priority levels of τ_l and τ_s, *if schedulable*, will not increase the average WCRT. We study the two priority assignments before and after the swapping, denoted respectively as **B** and **A** in Fig. 1. *Here* **A** *stands for* ***A****fter and* **B** *stands for* ***B****efore*. **A** and **B** only differ in the priorities of τ_l and τ_s. The three sets of tasks $\mathcal{H}$, $\mathcal{M}$, and $\mathcal{L}$, i.e., the sets of tasks with priority higher than, in between, and lower than τ_l and τ_s respectively, remain the same.

Theorem 1 *Consider a task system Γ and an associated WCRT analysis M that is compliant with Audsley's algorithm. As in Fig. 1, we swap two tasks τ_s and τ_l with $C_s \leq C_l$, such that both* **A** *and* **B***, the priority orders after and before swapping respectively, are schedulable. If M additionally satisfies the following two conditions (where $R_i^{\mathbf{A}}$ and $R_i^{\mathbf{B}}$ denote the WCRTs of τ_i in priority assignments* **A** *and* **B** *respectively)*

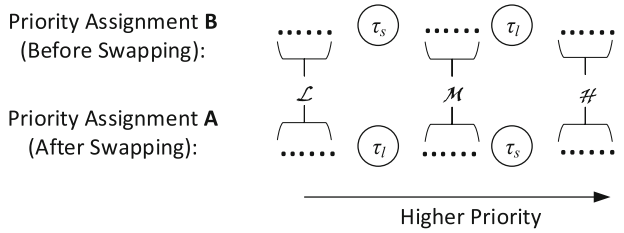


Fig. 1 Swapping τ_l with a lower priority task τ_s where $C_l \geq C_s$, if maintaining schedulability, reduces the average WCRT

$$\begin{aligned} &R_l^{\mathbf{B}} + R_s^{\mathbf{B}} \geq R_l^{\mathbf{A}} + R_s^{\mathbf{A}} \\ &\forall \tau_m \in \mathcal{M}, \quad R_m^{\mathbf{B}} \geq R_m^{\mathbf{A}} \end{aligned} \tag{12}$$

then Algorithm 1 is optimal for the problem in (11).

Proof The WCRT of any task in $\mathcal{H} \cup \mathcal{L}$ remains the same, since the sets of its higher priority and lower priority tasks are unchanged, and the analysis is compliant with Audsley's algorithm. This, combined with the conditions in (12), means that the average WCRT only decreases after the swapping.

Now consider at any point in Algorithm 1 where it tries to find an unassigned task to allocate the current priority level k. Let $\mathcal{S}$ be the set of unassigned tasks that are schedulable at this priority level. Further partition $\mathcal{S}$ into two disjoint sets $\mathcal{C} \cup \mathcal{T} = \mathcal{S}$ where $\mathcal{C} \subseteq \Omega$ and $\mathcal{T} \subseteq \Gamma \backslash \Omega$. Assume that $\mathcal{S} \neq \emptyset$ (otherwise the system is unschedulable). There are two cases.

Case 1 $\mathcal{T} \neq \emptyset$. We now show that for any $\tau_p \in \mathcal{T}$, there exists an optimal priority assignment that assigns τ_p at the current priority level k. Consider any optimal priority assignment $\mathbf{P}$ that assigns a different task τ_q at priority k and τ_p at a higher priority. Construct another priority assignment $\mathbf{P}'$ by inserting τ_p immediately behind τ_q in the priority order. Since $\tau_p \notin \Omega$, the increased WCRT of τ_p will not affect the total WCRT in Ω. However all other tasks will have equal or smaller WCRT since their priority is either shifted up by one level or remains the same. Hence, the cost of $\mathbf{P}'$ cannot be larger than that of $\mathbf{P}$, indicating that $\mathbf{P}'$ is also optimal.

Case 2 $\mathcal{T} = \emptyset$. Let τ_p be a task in $\mathcal{C}$ that has no smaller WCET than any other task in $\mathcal{C}$. Similarly, we show that there always exists an optimal priority assignment that assigns τ_p at the current level k. Consider any optimal priority assignment $\mathbf{P}$ that assigns a different task τ_q at level k and τ_p at a higher priority. τ_q has to be from $\mathcal{C}$ (since $\mathcal{T} = \emptyset$), hence its WCET cannot be smaller than τ_p. Now construct another priority assignment $\mathbf{P}'$ by swapping τ_p and τ_q in the priority order. Since $C_p \geq C_q$, by (12) the cost of $\mathbf{P}'$ can only be equal to or smaller than that of $\mathbf{P}$, indicating that $\mathbf{P}'$ is also optimal.

The rest of the proof follows that of the Audsley's algorithm (Davis and Burns 2009), as Algorithm 1 always constructs an optimal priority assignment by minimizing the cost at each priority level. □

We now study whether the WCRT analysis methods in Eqs. (1)–(9) satisfy the conditions in Theorem 1. We note that they are already known to be compliant with Audsley's algorithm (Davis et al. 2016). For preemptive scheduling, we first observe the property of *monotonicity with priority order* as formally stated in Lemma 2. We then demonstrate that the WCRT analysis in (1) satisfies the conditions in (12), as stated in Theorem 3.

Lemma 2 *In any* feasible *priority assignment for systems with preemptive scheduling, the WCRT of a lower priority task τ_i is always no smaller than that of a higher priority task τ_j.*

Proof We define the WCRT delay function $\mathbb{R}_i(\cdot)$ for τ_i as

$$\mathbb{R}_i(x) = C_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{x}{T_k} \right\rceil C_k$$

For τ_i and any higher priority task τ_j,

$$\begin{aligned} \forall x > 0,\ \mathbb{R}_i(x) &\geq C_i + \left\lceil \frac{x}{T_j} \right\rceil C_j + \sum_{\forall k \in hp(j)} \left\lceil \frac{x}{T_k} \right\rceil C_k \\ &\geq C_j + \sum_{\forall k \in hp(j)} \left\lceil \frac{x}{T_k} \right\rceil C_k \\ &= \mathbb{R}_j(x) \end{aligned}$$

The delay function and WCRT satisfy the following property

$$\forall x > 0, \mathbb{R}_i(x) \geq \mathbb{R}_j(x) \quad \Rightarrow \quad R_i \geq R_j \tag{13}$$

since R_i (resp. R_j) is the first fixed point solution to the equation $x = \mathbb{R}_i(x)$ (resp. $x = \mathbb{R}_j(x)$). □

Theorem 3 *Theorem 1 holds for preemptive systems with the WCRT analysis in Eq. (1).*

Proof First, $R_l^{\mathbf{B}} \geq R_s^{\mathbf{A}}$, as $R_l^{\mathbf{B}}$ and $R_s^{\mathbf{A}}$ have the same set of higher priority tasks $\mathcal{H}$, and R_i calculated in (1) is monotonically increasing with C_i.

Second, we prove $R_s^{\mathbf{B}} = R_l^{\mathbf{A}}$. Let $R^* = \min\{R_s^{\mathbf{B}}, R_l^{\mathbf{A}}\}$. Obviously $R^* \leq \min\{D_s, D_l\} \leq \min\{T_s, T_l\}$. Hence, R^* is the first fixed point solution of the following equation

$$R^* = C_s + C_l + \sum_{\forall j \in \mathcal{H} \cup \mathcal{M}} \left\lceil \frac{R^*}{T_j} \right\rceil C_j \tag{14}$$

We observe that R^* is a fixed point solution to the following equation for calculating $R_s^{\mathbf{B}}$

$$R_s^{\mathbf{B}} = C_s + \left\lceil \frac{R_s^{\mathbf{B}}}{T_l} \right\rceil C_l + \sum_{\forall j \in \mathcal{H} \cup \mathcal{M}} \left\lceil \frac{R_s^{\mathbf{B}}}{T_j} \right\rceil C_j \tag{15}$$

Also, since $R^* \leq R_s^{\mathbf{B}}$, and $R_s^{\mathbf{B}}$ is the first fixed point solution to the above equation, it must be $R^* = R_s^{\mathbf{B}}$. Likewise, $R^* = R_l^{\mathbf{A}}$. Thus, $R_s^{\mathbf{B}} = R_l^{\mathbf{A}}$, and $R_s^{\mathbf{B}} + R_l^{\mathbf{B}} \geq R_s^{\mathbf{A}} + R_l^{\mathbf{A}}$.

Now consider any task $\tau_m \in \mathcal{M}$. By Lemma 2 and the above proven equation $R_s^{\mathbf{B}} = R_l^{\mathbf{A}}$, $R_m^{\mathbf{A}} \leq R_l^{\mathbf{A}} = R_s^{\mathbf{B}} \leq T_s$. Hence, τ_m only suffers one interference from τ_s in $\mathbf{A}$. Since $C_l \geq C_s$, the amount of interference from τ_l to τ_m in $\mathbf{B}$ will only be larger than that from τ_s in $\mathbf{A}$. This, combined with the fact that the set of higher priority tasks for τ_m only differs from τ_l in $\mathbf{B}$ to τ_s in $\mathbf{A}$, implies $R_m^{\mathbf{B}} \geq R_m^{\mathbf{A}}$. □

In the following, we show that Theorem 1 also holds for the analysis in Eqs. (7)–(9) for *non-preemptive scheduling*. We first establish a property similar to Lemma 2, but for the waiting time calculated in (7). It relies on the definition of the *waiting delay function* as below.

Definition 1 The *waiting delay function* of a task τ_i in systems with non-preemptive scheduling is defined as

$$\mathbb{W}_i(x) = \tilde{B}_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{x}{T_k} \right\rceil C_k$$

Similar to (13), w_i and $\mathbb{W}_i(x)$ have the following property

$$\forall x > 0, \mathbb{W}_i(x) \geq \mathbb{W}_j(x) \quad \Rightarrow \quad w_i \geq w_j \tag{16}$$

Lemma 4 *In any priority assignment for systems with non-preemptive scheduling that is feasible according to the sufficient test in Eqs. (7)–(9), if τ_i has lower priority than τ_j, there is $w_i \geq w_j$.*

Proof We consider τ_i and its immediate higher priority task τ_{i-1}.

$$\begin{aligned}
\forall x > 0,\ \mathbb{W}_i(x) &\geq \tilde{B}_i + \left\lceil \frac{x}{T_{i-1}} \right\rceil C_{i-1} + \sum_{\forall k \in hp(i-1)} \left\lceil \frac{x}{T_k} \right\rceil C_k \\
&\geq \tilde{B}_i + C_{i-1} + \sum_{\forall k \in hp(i-1)} \left\lceil \frac{x}{T_k} \right\rceil C_k \\
&\geq \max\{\tilde{B}_i, C_{i-1}\} + \sum_{\forall k \in hp(i-1)} \left\lceil \frac{x}{T_k} \right\rceil C_k \\
&= \max\{B_{i-1}, C_{i-1}\} + \sum_{\forall k \in hp(i-1)} \left\lceil \frac{x}{T_k} \right\rceil C_k \\
&= \mathbb{W}_{i-1}(x)
\end{aligned}$$

By induction, this relationship can be generalized to τ_i and any higher priority task τ_j. Hence, the waiting time w_i of τ_i is no smaller than that of any higher priority task. □

The following two lemmas establish that the first condition in (12) is satisfied by the sufficient WCRT analysis for non-preemptive scheduling.

Lemma 5 *In Fig. 1, it is $w_l^{\mathbf{A}} \leq w_s^{\mathbf{B}}$ for the analysis in Eqs. (7)–(9) for non-preemptive scheduling.*

Proof The waiting time $w_s^{\mathbf{B}}$ is computed as follows

$$w_s^{\mathbf{B}} = \tilde{B}_s^{\mathbf{B}} + \left\lceil \frac{w_s^{\mathbf{B}}}{T_l} \right\rceil C_l + \sum_{\forall j \in \mathcal{H} \bigcup \mathcal{M}} \left\lceil \frac{w_s^{\mathbf{B}}}{T_j} \right\rceil C_j \tag{17}$$

Similarly for $w_l^{\mathbf{A}}$, it is computed as

$$w_l^{\mathbf{A}} = \tilde{B}_l^{\mathbf{A}} + \left\lceil \frac{w_l^{\mathbf{A}}}{T_s} \right\rceil C_s + \sum_{\forall j \in \mathcal{H} \bigcup \mathcal{M}} \left\lceil \frac{w_l^{\mathbf{A}}}{T_j} \right\rceil C_j \tag{18}$$

On the right hand side of (18) we substitute $w_l^{\mathbf{A}}$ with $w_s^{\mathbf{B}}$. Since $w_s^{\mathbf{B}} \leq T_s$, we derive the following quantity

$$\begin{aligned} w^* &= \tilde{B}_l^{\mathbf{A}} + \left\lceil \frac{w_s^{\mathbf{B}}}{T_s} \right\rceil C_s + \sum_{\forall j \in \mathcal{H} \bigcup \mathcal{M}} \left\lceil \frac{w_s^{\mathbf{B}}}{T_j} \right\rceil C_j \\ &= \tilde{B}_l^{\mathbf{A}} + C_s + \sum_{\forall j \in \mathcal{H} \bigcup \mathcal{M}} \left\lceil \frac{w_s^{\mathbf{B}}}{T_j} \right\rceil C_j \end{aligned} \tag{19}$$

We now prove that $w^* \leq w_s^{\mathbf{B}}$. We consider the following two cases.

Case 1 $\max_{\forall \tau_j \in \mathcal{L}} C_j \geq C_l$.

In this case, $\tilde{B}_l^{\mathbf{A}} = \tilde{B}_s^{\mathbf{B}} = \max_{\forall \tau_j \in \mathcal{L}} C_j$. Also, since $\left\lceil \frac{w_s^{\mathbf{B}}}{T_l} \right\rceil C_l \geq C_s$, we have $w^* \leq w_s^{\mathbf{B}}$.

Case 2 $\max_{\forall \tau_j \in \mathcal{L}} C_j < C_l$.

In this case, there is $\tilde{B}_l^{\mathbf{A}} = C_l$ and $\tilde{B}_s^{\mathbf{B}} \in [C_s, C_l]$. Since $\left\lceil \frac{w_s^{\mathbf{B}}}{T_l} \right\rceil C_l \geq C_l = \tilde{B}_l^{\mathbf{A}}$ and $\tilde{B}_s^{\mathbf{B}} \geq C_s$, it is again $w^* \leq w_s^{\mathbf{B}}$.

Combining the two cases, there is $w^* \leq w_s^{\mathbf{B}}$. With $w_s^{\mathbf{B}} \leq D_s \leq T_s$, this means that

$$w_s^{\mathbf{B}} \geq \tilde{B}_l^{\mathbf{A}} + \left\lceil \frac{w_s^{\mathbf{B}}}{T_s} \right\rceil C_s + \sum_{\forall j \in \mathcal{H} \bigcup \mathcal{M}} \left\lceil \frac{w_s^{\mathbf{B}}}{T_j} \right\rceil C_j \tag{20}$$

The above equation implies that the first fixed point solution of (18) must be no larger than $w_s^{\mathbf{B}}$, i.e., $w_l^{\mathbf{A}} \leq w_s^{\mathbf{B}}$. □

Lemma 6 *In Fig. 1, we have $w_l^{\mathbf{B}} = w_s^{\mathbf{A}}$ for the analysis in Eqs. (7)–(9) for non-preemptive scheduling.*

Proof It can be easily seen that

$$\tilde{B}_l^{\mathbf{B}} = \tilde{B}_s^{\mathbf{A}} = \max_{\forall j \in \mathcal{L} \cup \mathcal{M} \cup \{\tau_l, \tau_s\}} \{C_j\}$$

Thus, $w_s^{\mathbf{A}}$ and $w_l^{\mathbf{B}}$ are computed as the first fixed point of the same equation as below

$$w = \max_{\forall j \in \mathcal{L} \cup \mathcal{M} \cup \{\tau_l, \tau_s\}} \{C_j\} + \sum_{\forall k \in \mathcal{H}} \left\lceil \frac{w}{T_k} \right\rceil C_k \tag{21}$$

This implies that $w_s^{\mathbf{A}} = w_l^{\mathbf{B}}$. □

With the above three lemmas, we now are ready to formally prove Theorem 1 for non-preemptive scheduling.

Theorem 7 *Theorem 1 holds for the analysis in Eqs. (7)–(9) for non-preemptive scheduling.*

Proof By Lemmas 5 and 6

$$R_s^{\mathbf{A}} + R_l^{\mathbf{A}} - (R_s^{\mathbf{B}} + R_l^{\mathbf{B}}) = (w_l^{\mathbf{A}} - w_s^{\mathbf{B}}) + (w_s^{\mathbf{A}} - w_l^{\mathbf{B}}) \leq 0 \tag{22}$$

This proves the first condition in (12) is satisfied.

Now consider τ_m of intermediate priority level, i.e., $\tau_m \in \mathcal{M}$. By Lemma 4, there is $w_m^{\mathbf{A}} \leq w_l^{\mathbf{A}}$. By Lemma 5, $w_m^{\mathbf{A}} \leq w_l^{\mathbf{A}} \leq w_s^{\mathbf{B}} \leq D_s \leq T_s$. Therefore after swapping, τ_m suffers exactly one instance of interference from τ_s. We use $\mathcal{N}$ denote the set of tasks in $\mathcal{M}$ with priority higher than τ_m. Hence, in priority assignment $\mathbf{A}$, the set of tasks with priority higher than τ_m is $\mathcal{H} \cup \mathcal{N} \cup \{\tau_s\}$, and $w_m^{\mathbf{A}}$ is

$$\begin{aligned} w_m^{\mathbf{A}} &= \tilde{B}_m^{\mathbf{A}} + \left\lceil \frac{w_m^{\mathbf{A}}}{T_s} \right\rceil C_s + \sum_{\forall j \in \mathcal{H} \cup \mathcal{N}} \left\lceil \frac{w_m^{\mathbf{A}}}{T_j} \right\rceil C_j \\ &= \tilde{B}_m^{\mathbf{A}} + C_s + \sum_{\forall j \in \mathcal{H} \cup \mathcal{N}} \left\lceil \frac{w_m^{\mathbf{A}}}{T_j} \right\rceil C_j \end{aligned} \tag{23}$$

Likewise, in priority assignment $\mathbf{B}$, the set of tasks with priority higher than τ_m is $\mathcal{H} \cup \mathcal{N} \cup \{\tau_l\}$, and $w_m^{\mathbf{B}}$ is

$$w_m^{\mathbf{B}} = \tilde{B}_m^{\mathbf{B}} + \left\lceil \frac{w_m^{\mathbf{B}}}{T_l} \right\rceil C_l + \sum_{\forall j \in \mathcal{H} \cup \mathcal{N}} \left\lceil \frac{w_m^{\mathbf{B}}}{T_j} \right\rceil C_j \tag{24}$$

We now prove $w_m^{\mathbf{A}} \leq w_m^{\mathbf{B}}$, by considering the following two cases for $\tilde{B}_m^{\mathbf{B}}$.

Case 1 $C_l < \tilde{B}_m^{\mathbf{B}}$.

In this case, swapping C_l to a lower priority than τ_m does not change its blocking time, and $\tilde{B}_m^{\mathbf{A}} = \tilde{B}_m^{\mathbf{B}}$. From the Eqs. (23)–(24) for calculating $w_m^{\mathbf{A}}$ and $w_m^{\mathbf{B}}$, the right hand side of the equations, i.e., the waiting time functions satisfy the property that

$$\begin{aligned}\forall x > 0,\ \mathbb{W}_m^{\mathbf{B}}(x) &= \tilde{B}_m^{\mathbf{B}} + \left\lceil \frac{x}{T_l} \right\rceil C_l + \sum_{\forall j \in \mathcal{H} \cup \mathcal{N}} \left\lceil \frac{x}{T_j} \right\rceil C_j \\ &\geq \tilde{B}_m^{\mathbf{A}} + C_s + \sum_{\forall j \in \mathcal{H} \cup \mathcal{N}} \left\lceil \frac{x}{T_j} \right\rceil C_j \\ &= \mathbb{W}_m^{\mathbf{A}}(x)\end{aligned}$$

Hence, by (16), we have $w_m^{\mathbf{A}} \leq w_m^{\mathbf{B}}$.

Case 2 $C_l \geq \tilde{B}_m^{\mathbf{B}}$.

In this case, it can only be that $\tilde{B}_m^{\mathbf{A}} = C_l$. Also, $\tilde{B}_m^{\mathbf{B}} \geq C_s$. The waiting time functions satisfy the following equation

$$\begin{aligned}\forall x > 0,\ \mathbb{W}_m^{\mathbf{B}}(x) &= \tilde{B}_m^{\mathbf{B}} + \left\lceil \frac{x}{T_l} \right\rceil C_l + \sum_{\forall j \in \mathcal{H} \cup \mathcal{N}} \left\lceil \frac{x}{T_j} \right\rceil C_j \\ &\geq C_s + \tilde{B}_m^{\mathbf{A}} + \sum_{\forall j \in \mathcal{H} \cup \mathcal{N}} \left\lceil \frac{x}{T_j} \right\rceil C_j \\ &= \mathbb{W}_m^{\mathbf{A}}(x)\end{aligned}$$

Again, by (16), $w_m^{\mathbf{A}} \leq w_m^{\mathbf{B}}$.

Merging the above two cases, the second condition in (12) is also satisfied, which concludes the proof of the theorem. □

In contrast to the sufficient-only analysis, for the exact analysis of non-preemptive scheduling in Eqs. (2)–(6), the WCET Monotonic strategy is not optimal. We demonstrate with the counterexample in Table 1.

Example 1 For the task system in Table 1, WCET Monotonic will produce a priority assignment $\tau_2 > \tau_3 > \tau_1 > \tau_4 > \tau_5$, with an objective of $\sum_i R_i = 776$. Differently,

Table 1 An example task system to show the suboptimality of WCET monotonic with accurate analysis on non-preemptive scheduling

τ_i	T_i	D_i	C_i
τ_1	300	300	29
τ_2	100	100	27
τ_3	150	150	2
τ_4	350	350	73
τ_5	250	250	49

the optimal priority assignment is $\tau_2 > \tau_3 > \tau_4 > \tau_5 > \tau_1$, and the optimal objective is 771. The WCRT of each task under the two priority assignments is presented in Table 2.

In the example, each task has a busy period no larger than its period. This highlights that the WCET Monotonic policy is not optimal even if it is sufficient to analyze the WCRT of the first job in the busy period. We note that in this case, the exact analysis [Eqs. (2)–(6)] and the sufficient only analysis [Eqs. (7)–(9)] only differ on their consideration of the push through interference. Still, this makes a distinction in the optimality of the WCET Monotonic policy.

Specifically, in the example, the WCET Monotonic policy and the optimal priority order vary in their priority assignments on τ_4 and τ_1. By the accurate analysis, at priority level 3, τ_4 suffers a blocking time of $\max\{C_1, C_5\} = 49$ whereas τ_1 suffers a blocking time of $\max\{C_4, C_5\} = 73$. As a result, at priority level 3, τ_1 will suffer interferences of two jobs from τ_2 as opposed to one for τ_3, and the worst-case queuing delay of τ_1 is substantially longer than that of τ_4. In this special scenario, it is more beneficial to have τ_4 at priority level 3.

We now recompute the WCRTs for the two priority assignments using sufficient only analysis, where each task is also assumed to suffer the push through interference (which equals its WCET). The results are also summarized in Table 2. At priority level 3, τ_4 has the same amount of *adjusted* blocking time $\max\{C_4, C_5, C_1\} = 73$ as τ_1, making its worst-case queuing delay the same as that of τ_1 (which reflects Lemma 6). At priority level 1, τ_1 now suffers an adjusted blocking time equal to its own WCET $C_1 = 23$ and interference from τ_4 at higher priority. Similarly, τ_4 suffers an adjusted blocking time equal to its own WCET $C_4 = 73$ and interference from τ_1 at higher priority. The worst-case queuing delay turns out to be the same for both tasks. Thus the sums of WCRTs of τ_4 and τ_1 are the same for both priority assignments ($265 + 202 = 158 + 309$). However, the WCRT of τ_5 becomes smaller when scheduling the smaller WCET task τ_1 at higher priority. Intuitively, in the optimal priority assignment, τ_5 suffers interference from higher priority task τ_4 and a blocking time equal to its own WCET. However in WCET Monotonic priority assignment, τ_5 suffers interference from higher priority task τ_1 and blocking due to τ_4. Since $C_1 \leq C_5$, the overall queuing delay becomes smaller. As a result, WCET Monotonic priority assignment gives smaller average WCRT.

Table 2 WCRTs of optimal and WCET monotonic priority assignments

τ_i	Optimal order			WCET monotonic		
	π_i	Accurate-R_i	Sufficient-R_i	π_i	Accurate-R_i	Sufficient-R_i
τ_1	1	209	265	3	158	158
τ_2	5	100	100	5	100	100
τ_3	4	102	102	4	102	102
τ_4	3	151	202	1	207	309
τ_5	2	209	229	2	209	209

4 Minimizing weighted average WCRT

The optimality of WCET Monotonic strategy does not hold if tasks have different weights in problem (10). In fact, this general form of problem allows the designer to differentiate the importance of tasks, for example, the impact of their response times on the control performance. Consider the swapping in Fig. 1 but the weight β_l of τ_l is significantly higher than other tasks. Then scheduling τ_l at a lower priority incurs substantial cost that may outweigh the collective benefit from τ_s and tasks in $\mathcal{M}$, and Theorem 1 is not valid anymore. In this section, we propose a heuristic solution to the problem that is near optimal as demonstrated in the experiments. It consists of two ideas. The first is a *Scaled-WCET Monotonic* strategy, which mimics the WCET Monotonic strategy but divides the task WCET by the weight and use this scaled WCET to order tasks. The second is a refinement scheme to search for better solutions in the neighborhood.

Similar to the non-weighted case, the weighted problem can be interpreted as finding an *evaluation order* for use in Line 9 of Algorithm 1, which specifies the preferred order within the unassigned tasks to be checked for the current priority level. We introduce the concept of *scaled WCET*: for each task τ_i, its scaled WCET $\tilde{C}_i$ is defined as

$$\tilde{C}_i = \frac{C_i}{\beta_i} \tag{25}$$

We use the non-increasing scaled WCET as the evaluation order, as it intuitively puts a task with *smaller weight or longer WCET* to a lower priority.

In the following, we show that Scaled-WCET Monotonic strategy is optimal for preemptive scheduling when the system satisfies the following condition

$$\sum_{\forall \tau_i} C_i \leq \min_{\forall \tau_i} D_i \tag{26}$$

Condition (26) implies two properties: (i) the system is schedulable with any priority ordering; (ii) a task suffers exactly one job of interference from any higher priority task. The WCRT of τ_i can thus be computed as

$$R_i = \sum_{\forall \tau_j \in hp(i)} C_i \tag{27}$$

Theorem 8 *Scaled-WCET Monotonic policy is optimal for optimizing the weighted average WCRT of tasks with preemptive scheduling when condition* (26) *is satisfied.*

Proof Let τ_i and τ_j be two tasks of adjacent priorities, where τ_i is the higher priority task and $\frac{C_i}{\beta_i} \geq \frac{C_j}{\beta_j}$. It is sufficient to show that swapping the priority of τ_i and τ_j cannot increase the weighted average WCRT.

Let $\mathcal{H}$ be the set of tasks with higher priority than both τ_i and τ_j. Let $R_i^{\mathbf{A}}$ and $R_i^{\mathbf{B}}$ denote the WCRTs of τ_i after and before swapping the priority order respectively. Similarly, let $R_j^{\mathbf{A}}$ and $R_j^{\mathbf{B}}$ be the WCRTs of τ_j after and before swapping the priority

Table 3 An example task system to show the suboptimality of Scaled-WCET Monotonic

τ_i	T_i	D_i	C_i	β_i	$\tilde{C}_i = \frac{C_i}{\beta_i}$
τ_1	20	20	4	2	2
τ_2	20	10	6	1	6
τ_3	20	20	1	1	1

order respectively. Since τ_i and τ_j have adjacent priorities, swapping their priority order does not affect the WCRTs of other tasks. Thus the change in the weighted sum of WCRTs after swapping can be computed as

$$\begin{aligned}
&(\beta_i R_i^{\mathbf{A}} + \beta_j R_j^{\mathbf{A}}) - (\beta_i R_i^{\mathbf{B}} + \beta_j R_j^{\mathbf{B}}) \\
&= \left(\beta_i \sum_{\forall \tau_k \in \mathcal{H}} C_k + \beta_j \sum_{\forall \tau_k \in \mathcal{H} \cup \{\tau_i\}} C_k \right) - \left(\beta_i \sum_{\forall \tau_k \in \mathcal{H} \cup \{\tau_j\}} C_k + \beta_j \sum_{\forall \tau_k \in \mathcal{H}} C_k \right) \\
&= \beta_j C_i - \beta_i C_j \geq 0
\end{aligned} \tag{28}$$

This indicates that swapping the priority order of τ_i and τ_j can only improve the objective of minimizing the weighted average WCRT. □

However, Scaled-WCET Monotonic strategy is not guaranteed to be optimal when Condition (26) does not hold. We illustrate with the example system in Table 3.

Example 2 For the example system in Table 3, the optimal priority ordering of the above system is $\tau_1 > \tau_2 > \tau_3$, which has the minimized weighted WCRT summation of

$$2 \times 4 + 1 \times (4 + 6) + 1 \times (1 + 4 + 6) = 29 \tag{29}$$

In contrast, the Scaled-WCET Monotonic strategy will produce a priority assignment $\tau_3 > \tau_2 > \tau_1$, which has a weighted WCRT summation of

$$2 \times (1 + 4 + 6) + 1 \times (1 + 6) + 1 \times 1 = 30 \tag{30}$$

In this example, τ_2 cannot be assigned the lowest priority due to its constrained deadline and thus must have higher priority than either τ_1 or τ_3. Whichever task of τ_2 or τ_3 is scheduled at the lowest priority, its WCRT has to account for τ_2's interference. However, when τ_1 is scheduled at the lowest priority, its higher weight $\beta_1 = 2$ increases the impact of τ_2 on the objective.

To handle such cases that Scaled-WCET Monotonic strategy may be suboptimal, we introduce a sifting adjustment scheme that can further improve it based on the following observations. First, typically only the priority order of a very small portion of tasks is suboptimal, while the rest is still optimal. Second, suboptimality is usually caused by (i) a high-weight task is assigned with a priority that is too low; (ii) a low-weight task is assigned with a priority that is too high.

Algorithm 2 Sifting operations for adjusting priority orders

```
1: function SiftUp(τ_i, P)
2:     for each τ_j ∈ hp(i), lower priority first do
3:         Insert τ_j at the immediate lower priority than τ_i
4:         if system remains schedulable then
5:             return Success
6:         else
7:             Restore the priority of τ_j
8:         end if
9:     end for
10:    return Failure
11: end function
12: function SiftDown(τ_i, P)
13:    for each τ_j ∈ lp(i), higher priority first do
14:        Insert τ_j at the immediate higher priority than τ_i
15:        if system remains schedulable then
16:            return Success
17:        else
18:            Restore the priority of τ_j
19:        end if
20:    end for
21:    return Failure
22: end function
```

Algorithm 3 Tuning up priority levels

```
1: function TuneUp(Task set Γ, P)
2:     P^opt = P
3:     for each τ_i ∈ Γ do
4:         P = P^opt
5:         while SiftUp(τ_i, P) succeeds do
6:             if P is better than P^opt then
7:                 P^opt = P
8:             end if
9:         end while
10:    end for
11:    P = P^opt
12: end function
```

Thus our idea is to systematically adjust upward (downward) the priority level of potentially misplaced high (low) weight task to avoid local suboptimal solutions. For this, we define the following two operations $\texttt{SiftUp}(\tau_i, \mathbf{P})$ and $\texttt{SiftDown}(\tau_i, \mathbf{P})$. Specifically, $\texttt{SiftUp}(\tau_i, \mathbf{P})$ finds the lowest possible priority task τ_j in $hp(i)$ such that τ_j can be inserted after τ_i (i.e., with a priority immediately lower than τ_i) while maintaining system schedulability. τ_j is then inserted after τ_i. Similarly, $\texttt{SiftDown}(\tau_i, \mathbf{P})$ finds the highest possible priority task τ_j in $lp(i)$ that can be put ahead of τ_i while maintaining system schedulability. The pseudocode for the two operations is summarized in Algorithm 2.

Based on the two operations, we design two greedy adjustment algorithms `TuneUp` and `TuneDown`, which are performed after Algorithm 1. Consider `TuneUp` as an example, where the procedure is detailed in Algorithm 3. Specifically, for each task τ_i in the system, it repetitively applies the $\texttt{SiftUp}(\tau_i, \mathbf{P})$ operation as often as possible

(i.e., until there is no task in $hp(i)$ that can be inserted after τ_i while maintaining schedulability). Each time `SiftUp`(τ_i, $\mathbf{P}$) succeeds, the new priority order $\mathbf{P}$ is compared with the current best order $\mathbf{P}^{\text{opt}}$. If $\mathbf{P}$ has a smaller cost than $\mathbf{P}^{\text{opt}}$, then $\mathbf{P}^{\text{opt}}$ is updated as $\mathbf{P}$. At the end of the procedure, the best order $\mathbf{P}^{\text{opt}}$ is returned. `TuneDown` is the same as `TuneUp` except that `SiftUp`(τ_i, $\mathbf{P}$) is replaced with `SiftDown`(τ_i, $\mathbf{P}$) in Line 5.

`TuneUp` and `TuneDown` can be applied together to jointly improve priority assignment. Specifically, the adjusted priority assignment by one procedure can potentially provide room for further improvement by the other. Thus we introduce the following Algorithm 4, which integrates both tuning procedures to iteratively improve priority assignments.

Algorithm 4 Sifting adjustment

```
1: function SiftingAdjustment(Task set Γ, P)
2:     P^opt = P
3:     while true do
4:         P = P^opt
5:         TuneUp(Γ, P)
6:         TuneDown(Γ, P)
7:         if P is the same as P^opt then
8:             break
9:         end if
10:        P^opt = P
11:    end while
12: end function
```

In the algorithm, the order of Lines 5 and 6 can be altered. Each time the priority assignment is improved by either `TuneUp` or `TuneDown`, it is passed to the other for further improvement. The procedure stops when neither is capable of finding any improvement.

Example 3 We now demonstrate the proposed sifting adjustment algorithm on the example system in Table 3. The algorithm takes as input the initial priority assignment by Scaled-WCET Monotonic strategy $\mathbf{P} : \tau_3 > \tau_2 > \tau_1$. At Line 5, the algorithm first applies `TuneUp` on the priority assignment. For simplicity, consider the iteration (Lines 4–9 in Algorithm 3) that processes the lowest priority task τ_1. In the first iteration of the while loop, `SiftUp` operation will identify τ_3, which can be scheduled at the immediate lower priority level than τ_1. The priority assignment is then adjusted to $\tau_2 > \tau_1 > \tau_3$. However the new priority assignment does not improve upon the currently best priority assignment. Thus it is not stored into $\mathbf{P}^{opt}$. The algorithm then proceeds to the next iteration of the while loop, where `SiftUp` will identify τ_2, which can be scheduled at the immediate lower priority level than τ_1. The priority assignment is thus adjusted to $\tau_1 > \tau_2 > \tau_3$. The new adjusted priority assignment has smaller cost and is stored to $\mathbf{P}^{opt}$. After `TuneUp` returns, the adjusted priority assignment is given to `TuneDown` for further processing. Since it is already optimal, `SiftingAdjustment` eventually returns the adjusted priority assignment as the final result.

5 The concept of MUDA

We now consider the full version of the problem in (10), where there are additional linear constraints on task WCRT besides schedulability of individual tasks. Algorithm 1 is generally inapplicable here due to the additional constraints. A straightforward solution is to formulate it in standard mathematical programming framework such as Integer Linear Programming (ILP). However, this approach does not scale up to medium- or large-sized systems.

In the following, we present an efficient algorithm that runs several magnitudes faster than ILP. It is based on the following observations. First, the major difficulty of using ILP for solving (10) lies in the formulation of WCRT, which requires many integer variables (Zeng and Di Natale 2013). However, as detailed in the previous two sections, the subproblem of finding a schedulable priority assignment that minimizes the (weighted) average WCRT can be efficiently solved. Thus, our main idea is to free ILP solver from the burden of computing task WCRT. The proposed optimization framework is an iterative procedure that judiciously combines the power of ILP solver and the algorithms in the previous two sections, Sects. 3 and 4.

In this section, we introduce the concept of Maximal Unschedulable Deadline Assignment (MUDA), which is used to interact between the ILP solver and the algorithms in Sects. 3 and 4. In the next section, we detail the MUDA guided optimization framework. Throughout the two sections, we use the small example system in Table 4 with preemptive scheduling to illustrate. The objective is to minimize the average WCRT for all tasks (i.e., all $\beta_i = 1$).

Definition 2 A *virtual deadline (VD)* is a tuple $\langle \tau_i, d \rangle$ where d is an integer no larger than the deadline D_i of task τ_i. It represents a stricter deadline requirement for τ_i, i.e., $R_i \leq d$.

Although the concept of virtual deadline is proposed in, e.g., Baruah et al. (2012), it is used for scheduling, i.e., to be enforced at runtime under certain scenarios. Differently, we use it purely for design optimization, which does not affect the scheduling.

Definition 3 A *weighted average deadline (WAD)* is a tuple $\langle \Omega, d \rangle$ where $\Omega \subseteq \Gamma$ is the set of concerned tasks in the objective of problem (10), and d is an integer no larger than $\sum_{i \in \Omega} \beta_i \cdot D_i$. A WAD $\langle \Omega, d \rangle$ denotes a constraint upper bounding the objective of the optimization problem (10), i.e.,

$$\sum_{i \in \Omega} \beta_i \cdot R_i \leq d \tag{31}$$

Table 4 An example task system Γ_e for Sects. 5 and 6

τ_i	T_i	D_i	C_i	β_i
τ_1	10	10	2	1
τ_2	20	20	3	1
τ_3	40	40	10	1
τ_4	100	100	3	1

Definition 4 A *deadline assignment set* $\mathcal{R}$ is a collection of one VD for each task τ_i and one WAD, i.e., $\mathcal{R} = \{\langle\tau_1, d_1\rangle, ..., \langle\tau_n, d_n\rangle, \langle\Omega, d_\Omega\rangle\}$, which represents the *conjunction* (logic-AND, denoted by either the "{" or the "$\wedge$" symbol) of constraints as follows

$$\begin{cases} R_i \leq d_i, & \forall\langle\tau_i, d_i\rangle \in \mathcal{R} \\ \sum_{j\in\Omega} \beta_j \cdot R_j \leq d_\Omega, & \langle\Omega, d_\Omega\rangle \in \mathcal{R} \end{cases} \tag{32}$$

Example 4 Consider a deadline assignment set $\mathcal{R} = \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 20\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 35\rangle\}$. It represents the conjuncted set of constraints

$$\begin{aligned} &(R_1 \leq 10) \wedge (R_2 \leq 20) \wedge (R_3 \leq 20) \wedge (R_4 \leq 100) \\ \wedge&(R_1 + R_2 + R_3 + R_4 \leq 35) \end{aligned}$$

Definition 5 $\mathcal{R}_1$ is said to *dominate* $\mathcal{R}_2$, denoted as $\mathcal{R}_1 \succeq \mathcal{R}_2$, if and only if the constraints represented by $\mathcal{R}_1$ are looser than or the same as those of $\mathcal{R}_2$. More specifically, the VD and WAD in $\mathcal{R}_1$ are component wise no smaller than in $\mathcal{R}_2$. $\mathcal{R}_1$ is said to *strictly dominate* $\mathcal{R}_2$, denoted as $\mathcal{R}_1 \succ \mathcal{R}_2$, if $\mathcal{R}_1 \succeq \mathcal{R}_2$ and at least one component in $\mathcal{R}_1$ is larger than $\mathcal{R}_2$.

Example 5 Consider the following deadline assignment sets

$$\begin{aligned} \mathcal{R}_1 &= \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 20\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 35\rangle\} \\ \mathcal{R}_2 &= \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 20\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 40\rangle\} \\ \mathcal{R}_3 &= \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 40\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 35\rangle\} \\ \mathcal{R}_4 &= \{\langle\tau_1, 10\rangle, \langle\tau_2, 16\rangle, \langle\tau_3, 20\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 170\rangle\} \end{aligned} \tag{33}$$

We have $\mathcal{R}_2 \succeq \mathcal{R}_1$ and $\mathcal{R}_3 \succeq \mathcal{R}_1$, since $\mathcal{R}_1$ denotes stricter requirements than both $\mathcal{R}_2$ and $\mathcal{R}_3$. However, neither $\mathcal{R}_4$ nor $\mathcal{R}_1$ dominates each other, since $\mathcal{R}_1$ is more relaxed on the VD of τ_2, but is stricter on the WAD.

Definition 6 A system Γ is $\mathcal{R}$-*schedulable* (or informally, $\mathcal{R}$ is schedulable) if there exists a priority assignment $\mathbf{P}$ such that the task WCRTs satisfy the constraints represented by $\mathcal{R}$.

Example 6 Consider two deadline assignment sets as follows

$$\begin{aligned} \mathcal{R}_1 &= \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 20\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 35\rangle\} \\ \mathcal{R}_2 &= \{\langle\tau_1, 10\rangle, \langle\tau_2, 3\rangle, \langle\tau_3, 40\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 35\rangle\} \end{aligned} \tag{34}$$

Γ_e is $\mathcal{R}_1$-schedulable since in the WCET monotonic priority order $\tau_1 > \tau_2 > \tau_4 > \tau_3$, the task WCRTs are $R_1 = 2$, $R_2 = 5$, $R_3 = 20$, and $R_4 = 8$, which satisfy the constraints represented by $\mathcal{R}_1$. However, Γ_e is not $\mathcal{R}_2$-schedulable. This is because to satisfy $R_2 \leq 3$, τ_2 must have the highest priority. Also τ_1 must have higher priority than τ_3 (due to $C_3 > D_1$). With these priority orders in place, the priority assignment $\tau_2 > \tau_1 > \tau_4 > \tau_3$ minimizes the sum (= 36) of WCRTs, where the task WCRTs are $R_1 = 5$, $R_2 = 3$, $R_3 = 20$ and $\mathcal{R}_4 = 8$. However, they still violate the WAD in $\mathcal{R}_2$.

Obviously the following holds for deadline assignment sets with dominance relationship.

Theorem 9 *If Γ is $\mathcal{R}$-schedulable, it is schedulable for any $\mathcal{R}' \succeq \mathcal{R}$. If Γ is not $\mathcal{R}$-schedulable, it is not schedulable for any $\mathcal{R}' \preceq \mathcal{R}$.*

Proof This follows directly from the monotonicity of schedulability w.r.t. deadline assignments. If the system is $\mathcal{R}$-schedulable, then it is still schedulable with increased task deadlines (i.e., with a dominating $\mathcal{R}' \succeq \mathcal{R}$). Note that the schedulability analysis (as summarized in Sect. 2) is sustainable with respect to the deadline (Baruah and Burns 2006). □

Definition 7 A deadline assignment set $\mathcal{U}$ is a *Maximal Unschedulable Deadline Assignment (MUDA)* if and only if

- Γ is not $\mathcal{U}$-schedulable; and
- Γ is $\mathcal{R}$-schedulable for any strictly dominating $\mathcal{R} \succ \mathcal{U}$.

By Theorem 9, to verify the second condition of MUDA, it suffices to test only those $\mathcal{R}$s that increment (i.e., increase by one) any WAD or VD in $\mathcal{U}$.

Example 7 $\mathcal{R}_2$ in Example 6 is not a MUDA for the following reasons. For $\mathcal{U} = \{\langle\tau_1, 10\rangle, \langle\tau_2, 4\rangle, \langle\tau_3, 40\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 35\rangle\}$ which dominates $\mathcal{R}_2$, Γ_e is not schedulable. However, $\mathcal{U}$ is a MUDA. It suffices to verify that Γ_e is schedulable for both of the following deadline assignment sets

$$\begin{aligned} \mathcal{U}_1 &= \{\langle\tau_1, 10\rangle, \langle\tau_2, 5\rangle, \langle\tau_3, 40\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 35\rangle\} \\ \mathcal{U}_2 &= \{\langle\tau_1, 10\rangle, \langle\tau_2, 4\rangle, \langle\tau_3, 40\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 36\rangle\} \end{aligned} \tag{35}$$

(We note that $\langle\tau_1, 10\rangle$, $\langle\tau_3, 40\rangle$, or $\langle\tau_4, 100\rangle$ cannot be increased, since they already equal the respective task deadlines.)

Algorithm 1 can efficiently check whether Γ is $\mathcal{R}$-schedulable for any given $\mathcal{R}$. For every VD element $\langle\tau_i, d\rangle$ in $\mathcal{R}$, set the deadline of τ_i to be d. Compute the minimum weighted sum of WCRTs of tasks in Ω using Algorithm 1. If the minimized weighted sum is smaller than d_Ω, then Γ is $\mathcal{R}$-schedulable. A MUDA can be computed from an unschedulable deadline assignment set $\mathcal{R}$ using Algorithm 5. It uses binary search to find the maximal value that each deadline component d in $\mathcal{R}$ can be increased to while maintaining unschedulability. The algorithm requires $O(n \cdot \log d_{\max})$ number of $\mathcal{R}$-schedulability tests where $d_{\max} = \max\{d_1, \ldots, d_n, d_\Omega\}$. Note that the algorithm utilizes the fact that the schedulability analysis (as summarized in Sect. 2) is sustainable with respect to the deadline (Baruah and Burns 2006), i.e., the system schedulability only becomes better with larger deadlines.

Note that an unschedulable deadline assignment $\mathcal{R}$ may contain more than one MUDAs. To compute multiple different MUDAs from the same $\mathcal{R}$, our solution is to carefully perturb the input $\mathcal{R}$ to Algorithm 5 to avoid computing repetitive MUDAs. The observation is that Algorithm 5 computes MUDA by maximally increasing the deadlines in $\mathcal{R}$. Thus if a MUDA $\mathcal{U}$ is computed from an unschedulable deadline

Algorithm 5 Algorithm for computing MUDA

```
1: function MUDA(System Γ, deadline assignment set R)
2:     for each ⟨τ_i, d⟩ or ⟨Ω, d⟩ ∈ R do
3:         Use binary search to find out the largest value that d can be increased to while keeping Γ unschedulable
4:     end for
5:     return R
6: end function
```

assignment $\mathcal{R}$, it must be $\mathcal{U} \succeq \mathcal{R}$. Contra-positively, if $\mathcal{U} \not\succeq \mathcal{R}$, then $\mathcal{U}$ cannot be computed from $\mathcal{R}$ by Algorithm 5. Specifically, let $\mathcal{U}_1$ be an MUDA computed by Algorithm 5. To compute a different MUDA $\mathcal{U}_2$, it suffices to perturb the input $\mathcal{R}$ such that $\mathcal{U}_1 \not\succeq \mathcal{R}$. A straightforward way is to find $\langle \tau_i, d_i \rangle \in \mathcal{U}$ and $\langle \tau_i, d_i' \rangle \in \mathcal{R}$ that satisfy $d_i' \leq d_i < D_i$ and then set $d_i' = d_i + 1$. We give an example below.

Example 8 Consider the following deadline assignment set

$$\mathcal{R} = \{\langle \tau_1, 2 \rangle, \langle \tau_2, 3 \rangle, \langle \tau_3, 10 \rangle, \langle \tau_4, 3 \rangle, \langle \Omega, 18 \rangle\} \tag{36}$$

which is obviously unschedulable as each task deadline is set to the WCET. We now apply Algorithm 5 twice to compute two different MUDAs. Suppose the first MUDA computed is the following.

$$\mathcal{U}_1 = \{\langle \tau_1, 4 \rangle, \langle \tau_2, 4 \rangle, \langle \tau_3, 40 \rangle, \langle \tau_4, 100 \rangle, \langle \Omega, 170 \rangle\} \tag{37}$$

To avoid computing the same $\mathcal{U}_1$ for the second MUDA, we select the virtual deadline assignment $\langle \tau_1, 4 \rangle \in \mathcal{U}_1$ and set the corresponding one in $\mathcal{R}$ to be $\langle \tau_1, 5 \rangle$. Specifically, the input $\mathcal{R}$ for computing the second MUDA is perturbed to be

$$\mathcal{R}' = \{\langle \tau_1, 5 \rangle, \langle \tau_2, 3 \rangle, \langle \tau_3, 10 \rangle, \langle \tau_4, 3 \rangle, \langle \Omega, 18 \rangle\} \tag{38}$$

Since obviously $\mathcal{U}_1 \not\succeq \mathcal{R}'$, it is guaranteed that $\mathcal{U}_1$ will not be computed again.

This strategy can be generalized to the problem of computing a different MUDA with respect to a set of existing MUDAs $\mathbb{U} = \{\mathcal{U}_1, \cdots, \mathcal{U}_k\}$. In this case, it is sufficient to perturb the input $\mathcal{R}$ such that $\mathcal{U}_i \not\succeq \mathcal{R}, \forall i = 1, \cdots, k$.

However the perturbation must be carefully performed such that it does not cause the input $\mathcal{R}$ to become schedulable. In this paper, we propose the following recursive procedure Algorithm 6, which systematically traverse through all possible perturbations and find the one that i) maintains unschedulability and ii) guarantees to avoid computing repetitive MUDAs.

The algorithm takes as input the task system Γ, a deadline assignment $\mathcal{R}$, a set of collected MUDAs $\mathbb{U}$, and a parameter k which controls the total number of MUDAs to be computed. The algorithm first checks if the number of MUDAs collected in $\mathbb{U}$ has reached k or the input deadline assignment $\mathcal{R}$ is schedulable, in both cases the algorithm has no need to proceed and simply returns. Otherwise, the algorithm starts computing MUDAs. As discussed, it is necessary to ensure that MUDAs already

Algorithm 6 Algorithm for computing multiple MUDAs

```
1: function ComputeMUDAs(System Γ, deadline assignment set R, collected MUDAs 𝕌, k)
2:   if |𝕌| ≥ k or Γ is R-schedulable then
3:     return
4:   end if
5:   if no more unvisited 𝒰 to pick from 𝕌 then
6:     𝒰^new = MUDA(Γ, R)
7:     𝕌 = 𝕌 ∪ {𝒰^new}
8:   end if
9:   Pick next 𝒰 from 𝕌
10:  if 𝒰 ⋡ R then
11:    ComputeMUDAs(Γ, R, 𝕌)
12:  else
13:    for ⟨τ_i, d_i⟩ ∈ 𝒰 do
14:      if d_i < D_i then
15:        set d'_i = d_i + 1 where ⟨τ_i, d'_i⟩ ∈ R
16:        ComputeMUDAs(Γ, R, 𝕌)
17:        restore the value of d'_i
18:      end if
19:    end for
20:  end if
21: end function
```

computed in $\mathbb{U}$ will not be re-computed, or equivalently $\mathcal{U} \not\succeq \mathcal{R}, \forall \mathcal{U} \in \mathbb{U}$. This is done by picking an unvisited $\mathcal{U} \in \mathbb{U}$ and checking if $\mathcal{U} \succeq \mathcal{R}$ (Lines 9 and 10). If $\mathcal{U} \succeq \mathcal{R}$, the algorithm tries all possible perturbations (Lines 13–15) before proceeding to the next recursion level. If not, the algorithm directly proceed to the next recursion level (Line 11). In the next recursion level, the algorithm similarly picks the next unvisited $\mathcal{U} \in \mathbb{U}$. If all $\mathcal{U} \in \mathbb{U}$ has been processed, the algorithm proceeds to compute a new MUDA $\mathcal{U}^{new}$ safely and adds it to $\mathbb{U}$ (Lines 5–8).

6 MUDA guided optimization

We now present the optimization framework based on the concept of MUDA for solving the problem (10). We first observe that (10) can be equivalently formulated as a problem of finding the set of deadline assignment variables $\mathbf{d} = [d_1, \ldots, d_n, d_\Omega]$, such that (a) d_Ω is minimized; (b) Γ is schedulable with the deadline assignment set $\mathcal{R} = \{\langle \tau_1, d_1 \rangle, \ldots \langle \tau_n, d_n \rangle, \langle \Omega, d_\Omega \rangle\}$; (c) $\mathbb{G}(\mathbf{X}) \leq 0$ is satisfied assuming $R_i = d_i$. Formally, we re-formulate the problem as follows

$$\begin{aligned} &\min\ d_\Omega \\ &\text{s.t. } C_i \leq d_i \leq D_i, \forall \tau_i \in \Gamma \\ &\quad d_\Omega \geq \sum_{i \in \Omega} \beta_i \cdot C_i \\ &\quad R_i = d_i, \forall \tau_i \in \Gamma \\ &\quad \mathbb{G}(\mathbf{X}) \leq 0 \\ &\quad \Gamma \text{ is } \mathcal{R} - \text{schedulable} \end{aligned} \tag{39}$$

Intuitively, any unschedulable deadline assignment set $\mathcal{R}$ (and in particular any MUDA) denotes a combination of deadline assignments that cannot be simultaneously satisfied by any feasible priority assignment. Hence, $\mathcal{R} = \{\langle\tau_1, d_1^*\rangle, \ldots, \langle\tau_n, d_n^*\rangle, \langle\Omega, d_\Omega^*\rangle\}$ denotes the *disjunction* (logic-OR, denoted with either the "$\|$" or the "$\vee$" symbol) of constraints that any feasible deadline assignment $\{\langle\tau_1, d_1\rangle, \ldots, \langle\tau_n, d_n\rangle, \langle\Omega, d_\Omega\rangle\}$ must satisfy

$$\left\| \begin{array}{ll} d_i > d_i^* & \forall\langle\tau_i, d_i^*\rangle \in \mathcal{R} \\ d_\Omega > d_\Omega^* & \langle\Omega, d_\Omega^*\rangle \in \mathcal{R} \end{array} \right. \tag{40}$$

In this sense, any unschedulable deadline assignment set $\mathcal{R}$ partially shapes the feasibility region of the problem. We call (40) the *induced schedulability constraints* by $\mathcal{R}$.

The feasibility region of $\mathbf{d} = [d_1, d_2, \ldots, d_n, d_\Omega]$ can be defined by the set of all MUDAs of Γ. However computing all of them is obviously impractical as the number of MUDAs is exponential to the number of tasks. We note that in many cases, the objective is sensitive to only a small set of MUDAs. Thus, we devise a MUDA guided optimization framework which judiciously and gradually adds MUDAs into the problem until its optimal solution is found. The algorithm is illustrated in Figure 2. The calculation of WCRT is never explicit in the ILP formulation. Instead, it is abstracted into the form of MUDAs as an alternative representation of the feasibility region.

Step 1: Priority assignment evaluation order In this step, we find an evaluation order in Line 9 of Algorithm 1, which we will later use as $\mathcal{R}$-schedulability test in MUDA computation (Step 5). For the case of average WCRT, the optimal evaluation order is WCET monotonic, i.e., in non-increasing order of task WCET. For the case of weighted average WCRT, we use the algorithm in Sect. 4. Specifically, we first ignore extra design constraints and consider only schedulability. Then we apply the initial order by scaled-WCET in Algorithm 1 to obtain a first solution $\mathbf{P}$. We utilize the sifting adjustment to improve $\mathbf{P}$ into $\mathbf{P}'$, and then employ $\mathbf{P}'$ as the evaluation order in all subsequent MUDA calculations. If it fails to find any schedulable order, the procedure terminates since the problem is infeasible.

Step 2: Initial ILP The initial ILP Π contains only the constraints $\mathbb{G}(\mathbf{X}) \leq 0$, but not that Γ is $\mathcal{R}$-schedulable.

$$\begin{aligned} &\min\ d_\Omega \\ &\text{s.t. } C_i \leq d_i \leq D_i, \forall\tau_i \in \Gamma \\ &\quad d_\Omega \geq \sum_{i\in\Omega} \beta_i \cdot C_i \\ &\quad R_i = d_i, \forall\tau_i \in \Gamma \\ &\quad \mathbb{G}(\mathbf{X}) \leq 0 \end{aligned} \tag{41}$$

Step 3: Solving Π. Solve the ILP problem Π. Let d_Ω^* denote the objective value. If Π is infeasible, it implies that the original system Γ is unschedulable under the extra design constraints $\mathbb{G}(\mathbf{X}) \leq 0$, and the procedure terminates.

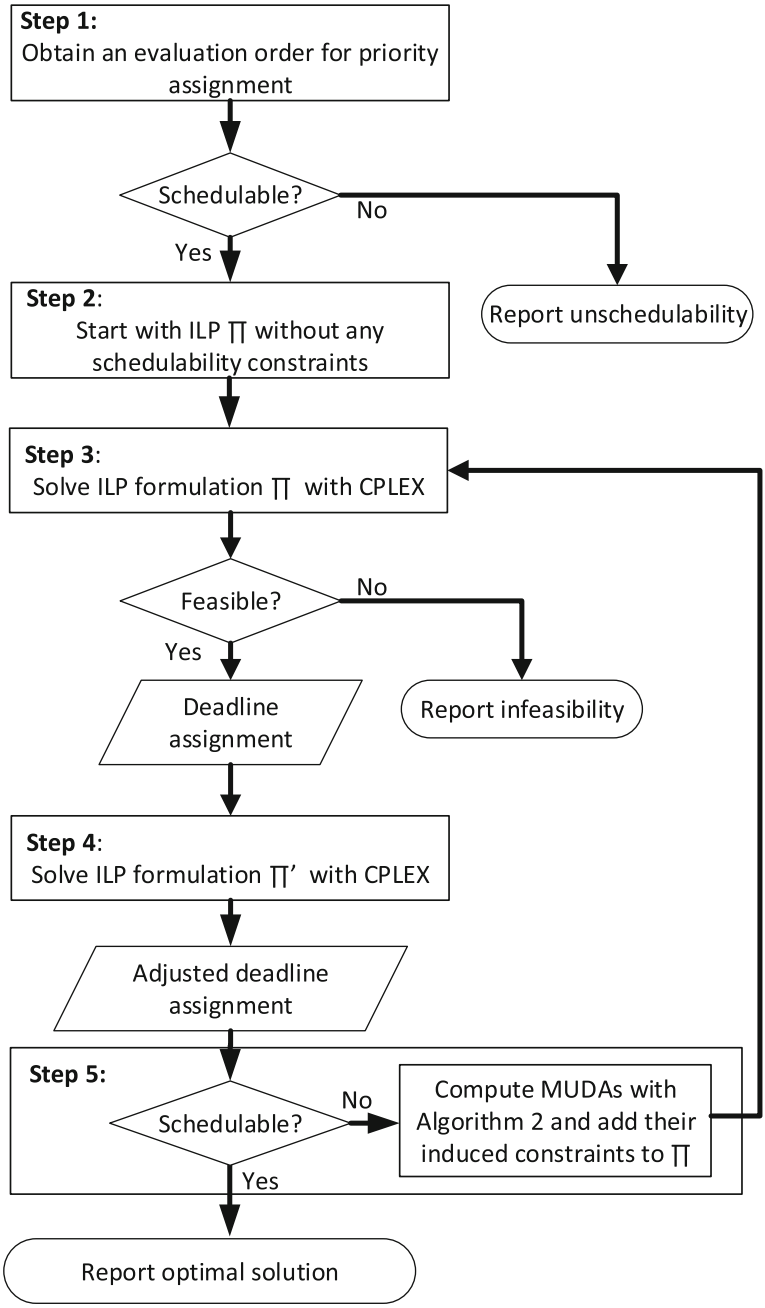


Fig. 2 The MUDA guided iterative optimization framework

Step 4: $\mathbf{d_i}$ ***relaxation*** Solve another ILP problem Π' constructed from Π as follows.

$$\begin{aligned} &\max \sum_{\forall \tau_i \in \Gamma} d_i \\ &\text{s.t. the constraints in } \Pi \text{ are satisfied} \\ &\quad d_\Omega = d^*_\Omega \end{aligned} \tag{42}$$

The main purpose of this step is to relax the deadline assignment of individual tasks that are not involved in the constraints while maintaining the same objective value.

Step 5: MUDA computation Let $\mathbf{d}^* = [d^*_1, \ldots, d^*_n, d^*_\Omega]$ be the solution from Π'. Construct a deadline assignment set $\mathcal{R}$ from $\mathbf{d}^*$ as follows.

$$\mathcal{R} = \{\langle \tau_1, d^*_1 \rangle, \ldots, \langle \tau_n, d^*_n \rangle, \langle \mathcal{C}, d^*_\Omega \rangle\} \tag{43}$$

If Γ is $\mathcal{R}$-schedulable, then the returned solution is optimal (see the remark below). Otherwise, compute a set of MUDAs from $\mathcal{R}$, add the induced schedulability constraints as in (40) to Π, and go to Step 3.

Remark 1 In Step 5, given any unschedulable deadline assignment, it is generalized to MUDA to rule out similar mistakes. This reduces the number of iterations, each of which needs to solve costly ILP problems. Also, the framework keeps in Π a subset of all constraints, i.e., it maintains an over-approximation of the feasibility region. Hence, if the solution from solving Π (which is optimal for Π) is indeed feasible, then it must be optimal for the original problem as well.

Example 9 We now illustrate the procedure on the system Γ_e in Table 4. Besides task schedulability, an additional constraint shall be satisfied $R_2 + R_3 \leq 20$. Since all tasks have the same weight, the evaluation order is WCET monotonic. The problem is then reformulated as the following ILP Π.

$$\begin{aligned} &\min d_\Omega \\ &\text{s.t. } C_i \leq d_i \leq D_i, \forall i = 1, \ldots, 4 \\ &\quad d_\Omega \geq \sum_{i=1}^{4} \beta_i \cdot C_i = 18 \\ &\quad R_i = d_i, \forall i = 1, \ldots, 4 \\ &\quad R_2 + R_3 \leq 20 \end{aligned} \tag{44}$$

The algorithm then iterates between Step 3 and Step 5.
Iteration 1 Solving the ILP Π in (44), and the solution is

$$\mathbf{d}^* = [d^*_1, d^*_2, d^*_3, d^*_4, d^*_\Omega] = [2, 3, 10, 3, 18]$$

Now note that d_1 and d_4 are not involved in the objective function or any design constraint, but they are assigned the lowest possible value. Also for d_2 and d_3, the

solution gives $d_2 + d_3 = 13$ while the maximum allowed bound for their sum is 20. Thus the deadline assignments can be further relaxed by solving the following problem Π' as described in Step 4.

$$\begin{aligned} \max\ & d_1 + d_2 + d_3 + d_4 \\ \text{s.t. } & C_i \le d_i \le D_i, \forall i = 1, \ldots, 4 \\ & d_\Omega \ge \sum_{i=1}^{4} \beta_i \cdot C_i = 18 \\ & R_i = d_i, \forall i = 1, \ldots, 4 \\ & R_2 + R_3 \le 20 \\ & d_\Omega = 18 \end{aligned} \tag{45}$$

Solving problem (45) returns following adjusted solution.

$$\mathbf{d}^* = [d_1^*, d_2^*, d_3^*, d_4^*, d_\Omega^*] = [10, 10, 10, 100, 18]$$

This new deadline assignment has the same d_Ω^* while satisfying all the constraints in Π, but is more relaxed in deadline assignments of individual tasks (i.e., d_1–d_4). Construct the corresponding deadline assignment set $\mathcal{R}_1$ as

$$\mathcal{R}_1 = \{\langle\tau_1, 10\rangle, \langle\tau_2, 10\rangle, \langle\tau_3, 10\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 18\rangle$$

Since Γ_e is not $\mathcal{R}_1$-schedulable, the following two MUDAs are computed. They both are still unschedulable, but they dominate $\mathcal{R}_1$ thus inducing more relaxed constraints than $\mathcal{R}_1$.

$$\mathcal{U}_{1,1} = \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 40\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 34\rangle\}$$
$$\mathcal{U}_{1,2} = \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 19\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 43\rangle\}$$

The induced schedulability constraints of the above two MUDAs are shown as follows, which are added to Π.

$$\begin{cases} (d_1 \ge 11) \vee (d_2 \ge 21) \vee (d_3 \ge 41) \vee (d_4 \ge 101) \vee (d_\Omega \ge 35) \\ (d_1 \ge 11) \vee (d_2 \ge 21) \vee (d_3 \ge 20) \vee (d_4 \ge 101) \vee (d_\Omega \ge 44) \end{cases}$$

Iteration 2 Solving the augmented ILP Π and Π' returns the following solution

$$\mathbf{d}^* = [d_1^*, d_2^*, d_3^*, d_4^*, d_\Omega^*] = [10, 10, 10, 100, 44]$$

Construct the corresponding deadline assignment set $\mathcal{R}_2$ as

$$\mathcal{R}_2 = \{\langle\tau_1, 10\rangle, \langle\tau_2, 10\rangle, \langle\tau_3, 10\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 44\rangle\}$$

Γ_e is not $\mathcal{R}_2$-schedulable. Thus we compute two MUDAs as below.

$$\mathcal{U}_{2,1} = \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 13\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 170\rangle\}$$
$$\mathcal{U}_{2,2} = \{\langle\tau_1, 10\rangle, \langle\tau_2, 20\rangle, \langle\tau_3, 16\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 52\rangle\}$$

The following induced constraints are updated to the ILP Π.

$$\begin{cases} (d_1 \geq 11) \vee (d_2 \geq 21) \vee (d_3 \geq 14) \vee (d_4 \geq 101) \vee (d_\Omega \geq 171) \\ (d_1 \geq 11) \vee (d_2 \geq 21) \vee (d_3 \geq 17) \vee (d_4 \geq 101) \vee (d_\Omega \geq 53) \end{cases}$$

Iteration 3 Solving the ILP Π returns the following solution

$$\mathbf{d}^* = [d_1^*, d_2^*, d_3^*, d_4^*, d_\Omega^*] = [10, 3, 17, 100, 44]$$

Construct the corresponding deadline assignment set $\mathcal{R}_3$ as

$$\mathcal{R}_3 = \{\langle\tau_1, 10\rangle, \langle\tau_2, 3\rangle, \langle\tau_3, 17\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 44\rangle\}$$

Γ_e is not $\mathcal{R}_3$-schedulable, and the MUDA below is computed

$$\mathcal{U}_{3,1} = \{\langle\tau_1, 10\rangle, \langle\tau_2, 4\rangle, \langle\tau_3, 19\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 44\rangle\}$$

The following induced constraints are added to the ILP Π.

$$(d_1 \geq 11) \vee (d_2 \geq 5) \vee (d_3 \geq 20) \vee (d_4 \geq 101) \vee (d_\Omega \geq 45)$$

Iteration 4 Solving the updated ILP Π and Π' returns the following solution

$$\mathbf{d}^* = [d_1^*, d_2^*, d_3^*, d_4^*, d_\Omega^*] = [10, 3, 17, 100, 45]$$

Construct the corresponding deadline assignment set $\mathcal{R}_4$ as

$$\mathcal{R}_4 = \{\langle\tau_1, 10\rangle, \langle\tau_2, 3\rangle, \langle\tau_3, 17\rangle, \langle\tau_4, 100\rangle, \langle\Omega, 45\rangle\}$$

Γ_e is now $\mathcal{R}_4$-schedulable. The returned priority assignment $\tau_2 > \tau_1 > \tau_3 > \tau_4$ is the optimal solution, and the minimized sum of WCRTs is 45.

7 Applicability to other objectives

The idea behind the proposed optimization framework is to efficiently abstract the feasibility region of schedulability using MUDA implied constraints. In this sense, the framework is not limited to optimizing the weighted average deadline d_Ω. In this section, we extend our discussion to its performance on other forms of optimization objectives. Specifically, we first study the objective known as maximizing minimum

laxity, for which the proposed technique can also handle efficiently. We then discuss another type of objective function expressed as a linear summation of virtual deadlines, i.e., $\sum_{\forall \tau_i} \beta_i d_i$, which is substantially difficult for the proposed technique.

A commonly used metric in real-time systems design optimization is the minimum laxity of all tasks. The laxity of a task is defined as the difference between its WCRT and deadline, namely, $L_i = D_i - R_i$. The amount of laxity of a task reflects its robustness to unknown random interference in a complicated environment. Thus it is desirable to maximize the minimum laxity of all tasks in the system. The problem can be formally expressed as

$$\begin{aligned} \max \quad & \min_{\forall \tau_i} D_i - R_i \\ \text{s.t.} \quad & \text{Tasks are schedulable} \\ & \mathbb{G}(\mathbf{X}) \leq 0 \end{aligned} \tag{46}$$

For systems subject to end-to-end latency deadline constraints (i.e., the sum of WCRTs of tasks on a communication chain must not exceed a certain bound), it is also possible to define laxity w.r.t an end-to-end path: the laxity of an end-to-end path is the difference between the end-to-end latency and the end-to-end latency deadline. For example, consider a distributed automotive system using synchronous activation (Zheng et al. 2007). The end-to-end latency of a path p is defined as

$$E_p = \sum_{\forall \tau_i \in p} (R_i + T_i) \tag{47}$$

The end-to-end laxity of a path is therefore

$$L_p = D_p - E_p \tag{48}$$

Similarly, it is desirable to maximize the minimum end-to-end laxity of all paths to increase robustness. The corresponding optimization problem can be formally expressed as

$$\begin{aligned} \max \quad & \min_{\forall p} L_p \\ \text{s.t.} \quad & \text{Tasks are schedulable} \\ & \mathbb{G}(\mathbf{X}) \leq 0 \end{aligned} \tag{49}$$

The proposed MUDA-guided optimization can be readily adapted to solve the above optimization problems, i.e., by replacing the original objective $\min d_\Omega$ in (39) with the corresponding objective in the target problem. In our experimental evaluation, we will show that the proposed framework can also achieve significant speedup comparing to a straightforward ILP formulation (see Sect. 8.4).

We now consider an objective in the form of

$$\min \sum_{\forall \tau_i} \beta_i d_i \tag{50}$$

The objective is in fact just an alternative expression of minimizing weighted summation of WCRTs. It can be similarly handled by the proposed framework by replacing the objective function (39) with (50). However, in our empirical study, we observe that the proposed technique performs rather poorly for the above form of objective. In the following we provide a detailed analysis on why this subtle difference changes the average performance of our algorithm.

When the objective function is written as $\min \sum_{\forall \tau_i} \beta_i d_i$, the ILP solver will attempt to assign a small value to each individual virtual deadline d_i instead of the single weighted average deadline d_Ω. Although this achieves a similar goal of minimizing the weighted average WCRT, the resulting deadline assignment returned by the ILP solver is much less likely to be schedulable, which makes it harder for the framework to terminate early.

Consider Example 9 in Sect. 6. Let the objective function in (44) be $\min d_1 + d_2 + d_3 + d_4$ instead of $\min d_\Omega$. At the end of iteration 1, the ILP solver will return the following deadline assignment

$$\mathcal{R}' = \{\langle \tau_1, 2\rangle, \langle \tau_2, 3\rangle, \langle \tau_3, 10\rangle, \langle \tau_4, 3\rangle, \langle \Omega, 170\rangle$$

as opposed to the original one

$$\mathcal{R} = \{\langle \tau_1, 10\rangle, \langle \tau_2, 10\rangle, \langle \tau_3, 10\rangle, \langle \tau_4, 100\rangle, \langle \Omega, 18\rangle$$

Though both being unschedulable, $\mathcal{R}'$ is much worse in the sense that the deadline assignments are much harder to meet. In other words, ILP solver will always tend to return deadline assignments that are likely to fail the schedulability test in *Step 5* of Fig. 2, which increases the number of iterations. From the perspective of mathematical programming, consider the projected feasibility region of problem Π in each iteration onto variables $d_1, \ldots, d_n$. If the projected region is an over-approximation of the exact one [i.e., that of problem (39)], it will allow some unschedulable but small-valued deadline assignments. In this case, the ILP solver will always tempt to return them as the solution, since they correspond to smaller objective values. As a result, in order for the algorithm to terminate with a schedulable deadline assignment, it needs to compute a sufficient amount of MUDAs for modeling the exact projected feasibility region, which can be prohibitively large when n is big (i.e, for large task systems).

The problem is much less severe for objective $\min d_\Omega$. Intuitively, the ILP solver will only assign small value to d_Ω, but assign other deadlines $d_1, \ldots, d_n$ with values as relaxed as possible. This makes the resulting deadline assignment more likely to succeed in passing the schedulability test of *Step 5* in Fig. 2. Similarly, from the perspective of forming the feasibility region, the algorithm only needs to compute MUDAs that are sufficient to model the exact projected feasibility region onto variable d_Ω. This, in practice, usually requires a very small number of MUDAs. At the same time, however, it should also be noted that the fundamental reason that we are able to convert the objective $\min \sum_{\forall \tau_i} \beta_i d_i$ to $\min d_\Omega$ is the availability of Algorithm 1. Otherwise, it is not possible to compute MUDAs involving d_Ω, which are necessary to model the projected feasibility region.

We now use a similar way of reasoning to explain why laxity based optimization objectives can also be handled efficiently. Although the minimum laxity seems to be related to all variables d_i's, which will require an accurate modeling of the projected feasibility region on $d_1, \ldots, d_n$, yet in most cases in practice, it is usually only a small number of critical tasks that are affecting the minimum laxity (i.e, tasks with very tight deadlines). Most of the other tasks can be given relatively more relaxed virtual deadlines without affecting the minimum laxity of the system. The way the proposed framework works naturally discovers these critical tasks. For example, after a few iterations, the algorithm may discover that it is always some common set of tasks whose deadlines cannot be met (the algorithm always maintains an over-approximation of the feasibility region and thus will always give optimistic estimation of minimum laxity first, which mainly affects the schedulability of critical tasks). Thus, the algorithm will be guided to compute MUDAs only relevant to them. In this sense, the algorithm naturally focuses on modeling the projected feasibility regions onto d_is of critical tasks only.

In summary, our observation is that *the proposed technique is most effective when the number of variables that the optimization objective is sensitive to is small*.

8 Experimental results

In this section, we present the results of experimental study. We first evaluate the quality of the heuristics on minimizing weighted average WCRT with only schedulability constraints (Sect. 4). Two industrial case studies are then used for evaluating the MUDA guided optimization technique in minimizing weighted average WCRT with extra design constraints. The first is an experimental vehicle system with advanced active safety features, and the second is a simplified version of fuel injection system.

8.1 Quality of heuristics for min weighted average WCRT

In this experiment, we focus on measuring the average suboptimality by the proposed heuristics in Sect. 4. We note the one in Sect. 3 is proven optimal. The following three methods are compared.

- `Scaled-WCET Monotonic`: Algorithm 1, with non-increasing scaled-WCET as the evaluation order in Line 9.
- `Scaled-WCET Monotonic + Sifting`: `Scaled-WCET Monotonic` with both Tune-down and Tune-up adjustments applied afterwards.
- `ILP`: Formulating the problem as an integer linear program and solving it with CPLEX.

`ILP` guarantees to return global optimal solutions upon termination, which can then be used to calculate the suboptimality of the other methods, defined as $\frac{sub-opt}{opt} \times 100\%$, where *sub* is the solution from `Scaled-WCET Monotonic` or `Scaled-WCET Monotonic + Sifting`, and *opt* is the optimal solution from `ILP`. We use randomly generated periodic systems with varying number of tasks and system utilization. Each task is assigned a period following the log-uniform distribution in the range [10,

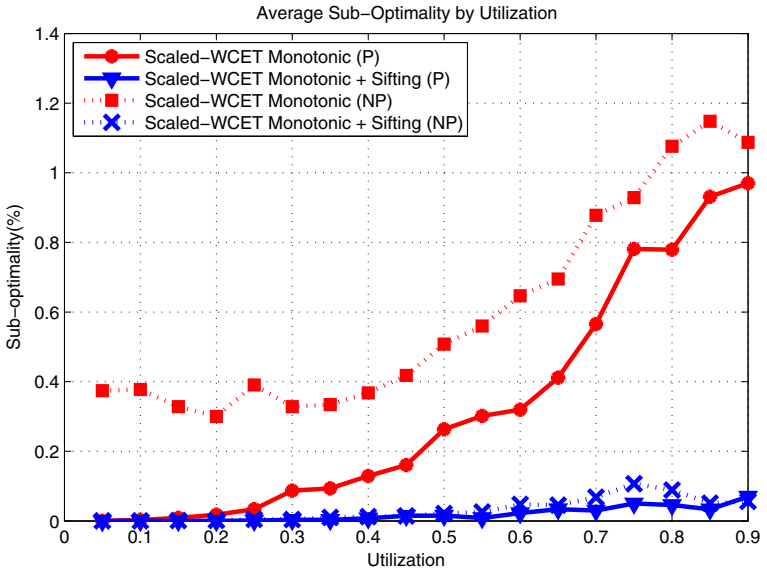


Fig. 3 Suboptimality for minimizing weighted average WCRT versus utilization

1000], and a utilization using the UUnifast-Discard algorithm (Davis and Burns 2009). The weight for each task is a random number between 1 and 10,000. This is the typical range for coefficients of control cost (Mancuso et al. 2014). We consider both preemptive scheduling (denoted as P in the figures) and non-preemptive scheduling (denoted as NP in the figures). Each point in the figures is the average result of 1000 randomly generated task sets.

We first fix the number of tasks to 20 and vary the total utilization. The results are plotted in Fig. 3, where the suboptimality of the heuristics generally increases with utilization. This is because task WCRTs are more sensitive to priority assignment in systems with higher utilization. However, the suboptimality is kept to be very small. For example, for both preemptive scheduling and non-preemptive scheduling, `Scaled-WCET Monotonic` is about 1% worse than `ILP` at 90% utilization. Sifting adjustment further improves the solution quality, as the average suboptimality is below 0.1% for both P and NP.

We then fix the system total utilization at 90% and vary the number of tasks. As in Fig. 4, the `Scaled-WCET Monotonic` heuristic and its sifting adjustment are again able to provide close-to-optimal solutions.

We now redraw the above results using Tukey type box plots to illustrate the distribution of the suboptimaltiy. Figures 5 and 6 show the results for `Scaled-WCET Monotonic`. Each box plot displays the distribution of suboptimality based on the five number summary: first quartile ($Q1$), median ($Q2$), third quartile ($Q3$), and maximum and minimum values that are still within the range $[Q1-1.5\times(Q3-Q1), Q3+1.5\times(Q3-Q1)]$. However, since `Scaled-WCET Monotonic + Sifting` achieves optimality for more than 75% of the test cases in both Figs. 3 and 4 (i.e., $Q1$ and $Q3$ are both 0), its box plots appear empty and we omit them. Instead, we only give its

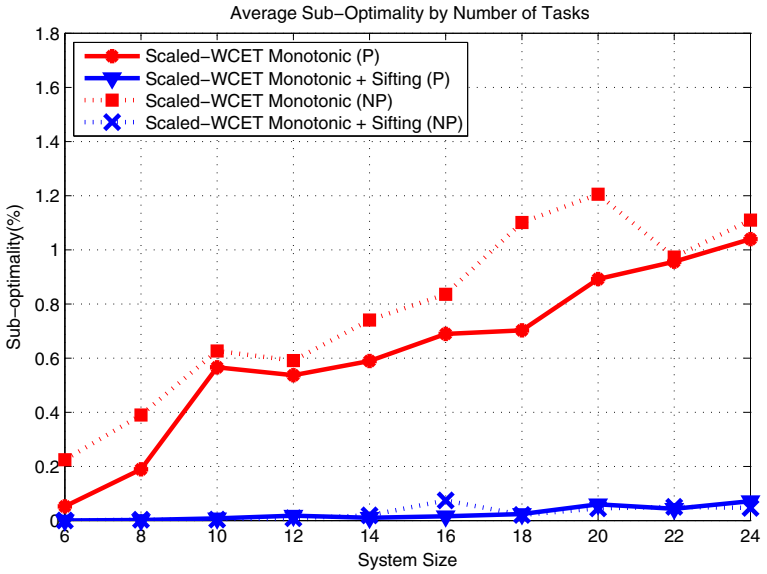


Fig. 4 Suboptimality for minimizing weighted average WCRT versus number of tasks

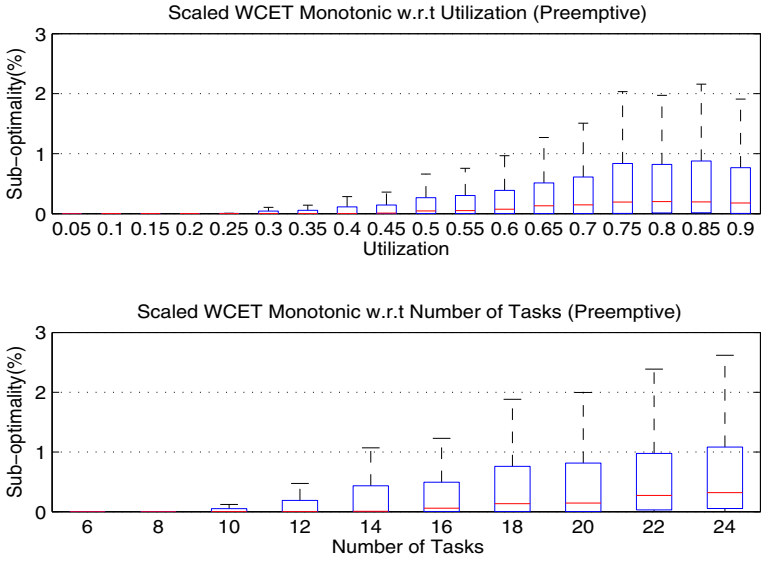


Fig. 5 Box plots for `Scaled-WCET Monotonic` (P)

maximum sub-optimality. For Fig. 3, the maximum sub-optimality is 7% for preemptive, and 10% non-preemptive scheduling. For Fig. 4, the maximum sub-optimality is 25% for preemptive, and 14% non-preemptive scheduling. Note that since most of the cases `Scaled-WCET Monotonic + Sifting` gives the optimal solution, its average suboptimality is always lower than 0.2%.

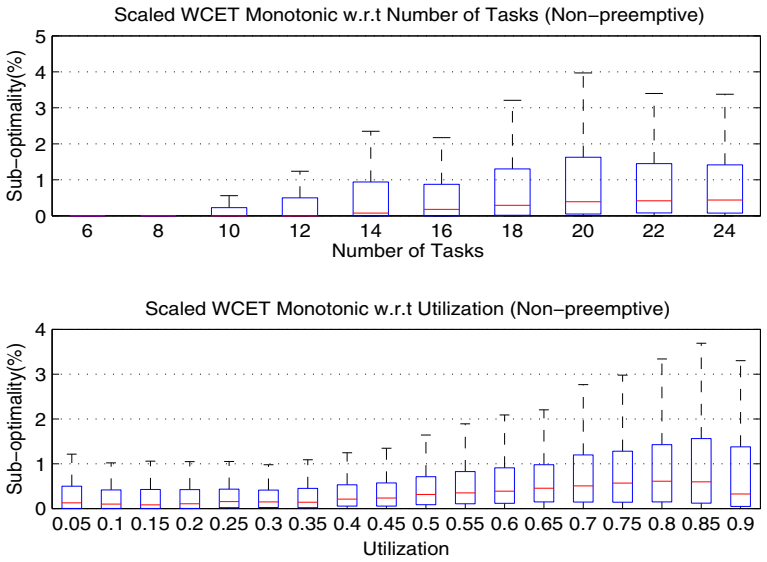


Fig. 6 Box plots for `Scaled-WCET Monotonic` (NP)

We also compare the proposed techniques with a default priority assignment as follows. For preemptive scheduling, we use deadline monotonic priority assignment, which is optimal for finding a schedulable priority assignment. For non-preemptive scheduling, we use Audsley's algorithm with the following revision: at each priority level, instead of randomly picking any schedulable task, it always selects the task with the longest deadline among all schedulable ones. Intuitively, this revised Audsley's algorithm follows the deadline monotonic policy as much as the system schedulability allows.

The default priority assignment has substantial suboptimality. For the systems in Fig. 3, the default priority assignment is 57–67% and 15–46% worse than the optimal solution for preemptive scheduling and non-preemptive scheduling respectively. For the systems in Fig. 4, the suboptimality is 30–63% and 7–40% for preemptive scheduling and non-preemptive scheduling respectively. This highlights the necessity to develop a problem-specific approach, as the default solution may be of low quality.

8.2 Experimental vehicle system with active safety features

The first industrial case study is an experimental vehicle system with advanced active safety features. It consists of 29 Electronic Control Units (ECUs) connected through 4 CAN buses, 92 tasks, and 192 CAN messages (Davare et al. 2007). Tasks are preemptive and CAN messages are scheduled non-preemptively. End-to-end delay deadlines are imposed on 12 pairs of source-sink tasks, which contain a total of 222 unique paths. The allocation of tasks and messages onto corresponding execution platforms are given. The problem is to find a priority assignment that minimizes the average WCRT of all tasks and messages, subject to the end-to-end deadline constraints and

Table 5 Result on min average WCRT for the experimental vehicle ("N/A" denotes no solution found)

Method	Objective	Time	Status
`MUDA-Guided`	305,828	234.9 s	Terminate
`ILP`	N/A	$\geq$ 24 h	Timeout

Table 6 Result on min weighted average WCRT for the experimental vehicle

Method	Objective	Time	Status
`MUDA-Guided`	7,995,881	3.36 s	Terminate
`ILP`	7,995,881	51,616.65 s	Terminate

the schedulability of individual tasks/messages. As discussed in (Davare et al. 2007), the end-to-end delay of a path is the sum of the WCRTs and periods for all tasks and messages in the path.

We compare the proposed MUDA-based technique (denoted as `MUDA-Guided`) with a straightforward ILP formulation (denoted as `ILP`). The results are summarized in Table 5. The proposed algorithm `MUDA-Guided` finds the optimal solution within just 235 seconds while the ILP solver fails to find any feasible solution within 24 h.

We now modify the problem to get a weighted version. Each task is assigned a weight of one plus the number of critical paths the task is involved. This metric has the benefit of being more aware of the critical tasks. The results are summarized in Table 6. Both the proposed approach (`MUDA-Guided`) and `ILP` return the same optimal solution, however, `MUDA-Guided` is $10,000\times$ faster.

8.3 Fuel injection system

The second industrial case study is the task system implementing a Simulink model of fuel injection system (Natale et al. 2010). It contains 90 tasks with preemptive scheduling, and 106 communication links. The communication link carries data between two tasks with harmonic periods, as required by Simulink. The total system utilization is 94.1%.

To protect the shared resource and preserve the same behavior as in the Simulink model, a wait-free buffer is introduced whenever the executions of the writer task and the reader task may overlap. Hence, the total memory cost incurred by the wait-free buffers is positively dependent on the task WCRTs: the wait-free buffer is avoided if there is no preemption between the writer and reader [(i.e., by ensuring that the WCRT of the lower rate task in the communication is no larger than the period of the higher rate task (Natale et al. 2010)]. The scenario is demonstrated in Fig. 7, where τ_j and τ_i are the lower and higher rate tasks respectively. It can be seen that as long as τ_j can finish before the next activation of τ_i, no preemption occurs. This remains to be true even when τ_j is assigned the higher priority. The condition can be re-interpreted as whether τ_j can be schedulable when assigned a constrained deadline $d_j \leq T_i$.

We first discuss how to formulate the memory requirement as additional constraint $\mathbb{G}(\mathbf{X}) \leq 0$ that is compatible with the proposed framework. For each pair of commu-

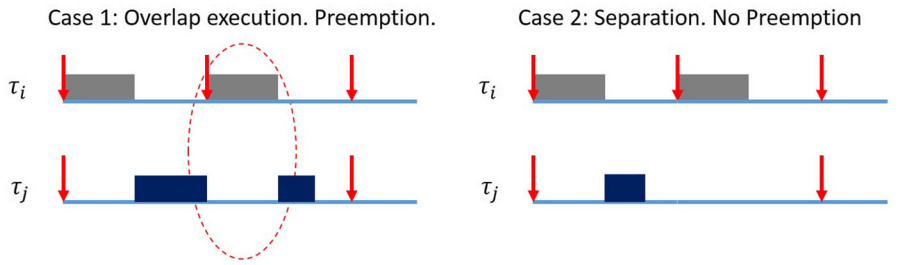


Fig. 7 The scenarios in which preemption does and does not occur

nicating task $\langle \tau_i, \tau_j \rangle$ (where τ_j is the lower rate task), we introduce a binary variable $b_{i,j}$ defined as follows

$$b_{i,j} = \begin{cases} 0 \text{ no preemption between } \tau_i \text{ and } \tau_j \\ 1 \text{ otherwise} \end{cases} \tag{51}$$

$b_{i,j}$ is subject to the following constraint

$$d_j \leq T_i + b_{i,j} \times D_j \tag{52}$$

Specifically, when $b_{i,j} = 0$, the above constraint enforces that $d_j \leq T_i$, which is equivalent to the condition for ensuring the absence of preemption. When $b_{i,j} = 0$, the constraint becomes $d_j \leq T_i + D_i$, which is trivial as $d_i \leq D_i$ shall always hold.

The memory requirement constraint can then be written as

$$\sum_{\forall \langle \tau_i, \tau_j \rangle} m_{i,j} \cdot b_{i,j} \leq M \tag{53}$$

where $m_{i,j}$ is the memory requirement of deploying a wait-free buffer for communication pairs τ_i and τ_j and M is the total amount of memory.

$\mathbb{G}(\mathbf{X}) \leq 0$ is then the aggregation of constraints (52) for all communication pairs $\langle \tau_i, \tau_j \rangle$ and (53).

We consider the problem of minimizing the average and weighted average WCRT of all tasks subject to the constraint of available memory for wait-free buffers. The weight of each task is randomly generated between 1 and 10,000. We compare the proposed approach `MUDA-Guided` with an ILP formulation (denoted as `ILP`). To test the efficiency of the techniques under different tightness of design constraints, we give four settings on memory constraints. The result are summarized in Tables 7 and 8. `ILP` is unable to find any feasible solution in 24 h except for two most relaxed memory constraint settings. On the other hand, `MUDA-Guided` solves all problem settings within a few minutes, either finding a much better solution than `ILP` or detecting infeasibility.

Table 7 Result on min average WCRT for the fuel injection system ("N/A" denotes no solution found)

Memory	MUDA-Guided			ILP		
	Objective	Time	Status	Objective	Time	Status
8900	Infeasible	138.21 s	Terminate	N/A	$\geq$ 24 h	Timeout
8950	14,090,418	183.68 s	Terminate	N/A	$\geq$ 24 h	Timeout
9000	13,392,124	5.11 s	Terminate	16,862,700	$\geq$ 24 h	Timeout
9100	13,384,171	1.20 s	Terminate	13,487,800	$\geq$ 24 h	Timeout

Table 8 Result on min weighted average WCRT for the Fuel injection system ("N/A" denotes no solution found)

Memory	MUDA-Guided			ILP		
	Objective	Time	Status	Objective	Time	Status
8900	Infeasible	483.87 s	Terminate	N/A	$\geq$ 24 h	Timeout
8950	4.67e+10	352.34 s	Terminate	N/A	$\geq$ 24 h	Timeout
9000	4.13e+10	22.62 s	Terminate	N/A	$\geq$ 24 h	Timeout
9100	4.12e+10	0.95 s	Terminate	N/A	$\geq$ 24 h	Timeout

Table 9 Result on maximizing laxity among tasks for the experimental vehicle

Method	Objective	Time	Status
MUDA-Guided	7791	2.84 s	Terminate
ILP	7754	$\geq$ 24 h	Timeout

Table 10 Result on maximizing laxity among end-to-end paths for the experimental vehicle

Method	Objective	Time	Status
MUDA-Guided	9589	21.64 s	Terminate
ILP	63	$\geq$ 24 h	Timeout

8.4 Optimization of laxity based objectives

We now apply our algorithms on the problems with other objectives, i.e., (i) maximizing the minimum laxity among all tasks, and (ii) maximizing the minimum laxity among all the end-to-end paths. We use the vehicle system case study and compare our technique with direct formulations in ILP.

The results are summarized in Tables 9 and 10 respectively. For both versions of the optimization problems, the proposed technique is capable of finding the optimal solutions in less than half a minute, while ILP timeouts and only finds suboptimal solutions within the time limit of 24 h. This demonstrates that our approach is able to efficiently handle several optimization objectives as discussed in Sect. 7.

9 Related work

There has been a large body of work for priority assignment in real-time systems scheduled with fixed priority. The seminal work from Liu and Layland (1973) shows that for periodic task system where task deadline equals period, rate-monotonic (RM) priority assignment is optimal for schedulability in the sense that there is no system that can be scheduled by some priority assignment but not by RM. When tasks have constrained deadline (i.e., no larger than period), deadline monotonic (DM) policy is shown to be optimal for schedulability (Audsley et al. 1991). For tasks with arbitrary deadline, Audsley's algorithm (Audsley 2001) guarantees to give a feasible priority assignment if one exists. It only needs to explore a quadratic $O(n^2)$ number of priority assignments (among the total of $n!$) where n is the number of tasks. Audsley's algorithm is optimal in terms of schedulability for a variety of task models and scheduling policies, as summarized in the authoritative survey by Davis et al. (2016). The three necessary and sufficient conditions for its optimality are presented in Davis and Burns (2009). Besides schedulability, Audsley's algorithm can be revised to optimize several other objectives, including the number of priority levels (Audsley 2001), lexicographical distance (the perturbation needed to make the system schedulable from an initial order) (Chu and Burns 2008; Davis et al. 2016), and robustness (ability to tolerate additional interferences) (Davis and Burns 2007).

For complex problems on priority assignment optimization where Audsley's algorithm do not apply, the current approaches include (a) meta heuristics such as simulated annealing (e.g., Tindell et al. 1992; Bate and Emberson 2006) and genetic algorithm (e.g., Hamann et al. 2004; Mehiaoui et al. 2013); (b) problem specific heuristics (e.g., Saksena and Wang 2000; Zeng et al. 2014; Wang et al. 2016); and (c) directly applying existing optimization frameworks such as branch-and-bound (BnB) (e.g., Wang and Saksena 1999) and ILP (e.g., Natale et al. 2010; Zeng and Di Natale 2012). These approaches either do not have any guarantee on solution quality, or suffer from scalability issues and may have difficulty to handle large industrial designs. Differently, a framework based on the concept of unschedulability core, i.e., the irreducible set of priority orders that cause the system unschedulable, is proposed for systems that Audsley's algorithm is optimal for schedulability (Zhao and Zeng 2017a, 2018b). Furthermore, it is generalized to systems where the schedulability of a task not only depends on the set of higher/lower priority tasks, but also on the response times of other tasks (hence Audsley's algorithm is not directly applicable) (Zhao and Zeng 2018a). However, the approach in Zhao and Zeng (2017a, 2018a, b) does not handle problems studied here, i.e., those involving the task WCRTs in the objective or additional constraints.

With respect to design optimization problems which are sensitive to the actual task response times, a branch-and-bound algorithm is developed to optimize both priority and period assignments (Mancuso et al. 2014). In the paper, a linear lower bound is adopted as an approximation to response time. The problem of optimizing period assignment for distributed systems is formulated in geometric programming framework, where the task WCRT is approximated with a linear function on task rates (Davare et al. 2007). Lukasiewycz et al. (2016) study the problem of ID (i.e., priority) obfuscation for CAN messages. The optimization procedure contains a first stage of

minimizing the average task WCRT by formulating the problem as a Quadratically Constrained Integer Quadratic Program. In Shin and Sunwoo (2007), a genetic algorithm is used for the problem of priority and period assignment that minimize the sum of end-to-end delays in networked control systems. Zhu et al. (2013) consider the problem of finding task allocation and priority assignment that maximize the minimum end-to-end laxity in distributed systems. The approach is to divide the problem into two stages, each of which is then formulated as an ILP program. Also, problem specific heuristics are developed under various settings (such as Zeng et al. 2010), but none of them provide any guarantee on solution quality. Our approach differs from all the above in that it designs a customized, exact optimization procedure specialized for the minimization of (weighted) average WCRT. It has been extended to simultaneous optimization of period and priority assignment (Zhao et al. 2018).

10 Conclusion

In this paper, we propose an optimization framework for task systems with constrained deadlines and preemptive/non-preemptive scheduling. We propose a new concept, Maximal Unschedulable Deadline Assignment, that defines a set of virtual deadline assignments to tasks such that they are not all schedulable. We develop an iterative framework that maintains a relaxed version of the original problem and leverages this concept to quickly rule out unschedulable solutions. We discuss the applicability of the framework with respect to several optimization objectives, including the minimization of weighted average WCRT and maximization of minimum laxity. The proposed technique significantly reduces the runtime while providing close-to-optimal solutions, as indicated in the case studies. As future work, we plan to investigate other types of real-time systems with different task models and scheduling policies.

Acknowledgements This paper is partially supported by NSF Grants No. 1739318 and 1812963.

References

Arzén KE, Cervin A, Eker J, Sha L (2000) An introduction to control and scheduling co-design. In: IEEE conference on decision and control

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

Audsley N, Burns A, Richardson M, Wellings A (1991) Hard real-time scheduling: the deadline-monotonic approach. In: IEEE workshop on real-time operating systems and software

Baruah S, Bonifaci V, D'Angelo G, Li H, Marchetti-Spaccamela A, van der Ster S, Stougie L (2012) The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In: Euromicro conference on real-time systems

Baruah S, Burns A (2006) Sustainable scheduling analysis. 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

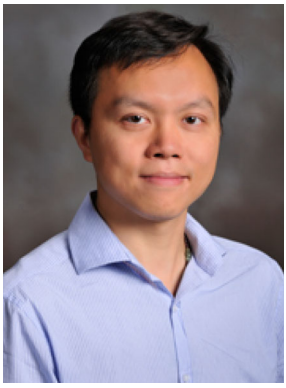
Bini E, Cervin A (2008) Delay-aware period assignment in control systems. In: IEEE real-time systems symposium

Chu Y, Burns A (2008) Flexible hard real-time scheduling for deliberative ai systems. Real Time Syst 40(3):241–263

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: ACM/IEEE design automation conference
Davis R, Burns A (2007) Robust priority assignment for fixed priority real-time systems. In: IEEE real-time systems symposium
Davis R, Burns A (2009) Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In: IEEE real-time systems symposium
Davis R, Burns A, Bril R, Lukkien J (2007) Controller area network (can) schedulability analysis: Refuted, revisited and revised. Real Time Syst 35(3):239–272
Davis R, Cucu-Grosjean L, Bertogna M, Burns A (2016) A review of priority assignment in real-time systems. J Syst Archit 65(C):64–82
Davis R, Zabos A, Burns A (2008) Efficient exact schedulability tests for fixed priority real-time systems. IEEE Trans Comput 57(9):1261–1276
Di Natale M, Zeng H (2013) Practical issues with the timing analysis of the controller area network. In: IEEE conference on emerging technologies & factory automation
Eisenbrand F, Rothvoß T (2008) Static-priority real-time scheduling: response time computation is np-hard. 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
Lincoln B, Cervin A (2002) Jitterbug: a tool for analysis of real-time control performance. In: IEEE conference decision and control
Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61
Lukasiewycz M, Mundhenk P, Steinhorst S (2016) Security-aware obfuscated priority assignment for automotive can platforms. ACM Trans Des Autom Electron Syst 21(2):1–27
Mancuso G, Bini E, Pannocchia G (2014) Optimal priority assignment to control tasks. ACM Trans Embed Comput Syst 13(5s):1–17
Mehiaoui A, Wozniak E, Tucci Piergiovanni S, Mraidha C, Natale MD, Zeng H, Babau J, Lemarchand L, Gérard S (2013) A two-step optimization technique for functions placement, partitioning, and priority assignment in distributed systems. In: SIGPLAN/SIGBED conference on languages, compilers and tools for embedded systems, pp 121–132
Natale MD, Guo L, Zeng H, Sangiovanni-Vincentelli A (2010) Synthesis of multi-task implementations of simulink models with minimum delays. IEEE Trans Ind Inf 6(4):637–651
Saksena M, Wang Y (2000) Scalable real-time system design using preemption thresholds. In: IEEE real-time systems symposium
Samii S, Yin Y, Peng Z, Eles P, Zhang Y (2009) Immune genetic algorithms for optimization of task priorities and flexray frame identifiers. In: IEEE conference embedded and real-time computing systems and applications
Shin M, Sunwoo M (2007) Optimal period and priority assignment for a networked control system scheduled by a fixed priority scheduling system. Int J Automot 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
Tindell KW, Burns A, Wellings AJ (1994) An extendible approach for analyzing fixed priority hard real-time tasks. Real Time Syst 6(2):133–151
Wang C, Gu Z, Zeng H (2016) Global fixed priority scheduling with preemption threshold: Schedulability analysis and stack size minimization. IEEE Trans Parallel Distrib Syst 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 (2012) Efficient implementation of autosar components with minimal memory usage. In: 7th IEEE international symposium on industrial embedded systems, pp 130–137
Zeng H, Di Natale M (2013) An efficient formulation of the real-time feasibility region for design optimization. IEEE Trans Comput 62(4):644–661
Zeng H, Di Natale M, Zhu Q (2014) Minimizing stack and communication memory usage in real-time embedded applications. ACM Trans Embed Comput Syst 13(5s):1–25
Zeng H, Ghosal A, Di Natale M (2010) Timing analysis and optimization of flexray dynamic segment. In: 10th IEEE international conference on computer and information technology, pp 1932–1939
Zhao Y, Gala V, Zeng H (2018) A unified framework for period and priority optimization in distributed hard real-time systems. IEEE Trans Comput Aided Des Integr Circuits Syst 37(11):2188–2199

Zhao Y, Zeng H (2017a) 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 (2017b) The virtual deadline based optimization algorithm for priority assignment in fixed-priority scheduling. In: IEEE real-time systems symposium, pp 116–127
Zhao Y, Zeng H (2018a) The concept of response time estimation range for optimizing systems scheduled with fixed priority. In: IEEE real-time and embedded technology and applications symposium (RTAS), pp 283–294
Zhao Y, Zeng H (2018b) The concept of unschedulability core for optimizing real-time systems with fixed-priority scheduling. IEEE transactions on computers, p 1
Zheng W, Di Natale M, Pinello C, Giusto P, Vincentelli AS (2007) Synthesis of task and message activation models in real-time distributed automotive systems. In: Design, automation & test in Europe conference & exhibition, IEEE, pp 1–6
Zhu Q, Zeng H, Zheng W, Natale MD, Sangiovanni-Vincentelli A (2013) Optimization of task allocation and priority assignment in hard real-time distributed systems. ACM Trans Embed Comput Syst 11(4):1–30

Yecheng Zhao is currently pursuing the PhD degree in Computer Engineering at Virginia Tech. He received his B.E. in Electrical Engineering from Harbin Institute of Technology, Harbin, China in 2014. His main research interest is design optimization techniques for real-time embedded systems. He received a best student paper award in RTSS 2017.



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 four best paper / best student paper awards in the above fields.