An Efficient Formulation of the Real-Time Feasibility Region for Design Optimization

Haibo Zeng, *Member*, *IEEE*, and Marco Di Natale, *Senior Member*, *IEEE*

Abstract—In the design of time-critical applications, schedulability analysis is used to define the feasibility region of tasks with deadlines, so that optimization techniques can find the best design solution within the timing constraints. The formulation of the feasibility region based on the response time calculation requires many integer variables and is too complex for solvers. Approximation techniques have been used to define a convex subset of the feasibility region, used in conjunction with a branch and bound approach to compute suboptimal solutions for optimal task period selection, priority assignment, or placement of tasks onto CPUs. In this paper, we provide an improved and simpler real-time schedulability test that allows an exact and efficient definition of the feasibility region in Mixed Integer Linear Programming (MILP) optimization. Our method requires a significantly smaller number of binary variables and is viable for the treatment of industrial-size problem, as shown by the experiments.

Index Terms—Real-time systems, schedulability analysis, mixed integer linear programming

1 Introduction

The design of complex embedded real-time systems is subject to many requirements and constraints, including limited resources, cost, performance of control algorithms, and energy consumption. In hard real-time systems, the design space (or the feasibility region) must satisfy the *schedulability constraints*, requiring that tasks (or threads, scheduled on CPUs) complete before their deadlines. Formally, the design optimization problem can be represented by parameters, decision variables, constraints, and objective functions. The decision variables represent the set of design choices under the control of the designers. The set of constraints defines the feasibility region, the domain of the allowed values for the decision variables. Also, an objective function characterizes the optimization goal.

Sometimes the problem is quite simple, for example, when the only design variables are task priorities and there is no optimization metric. Thus, the designer simply needs a feasible solution, which may be provided by a simple policy or algorithm. For example, in uniprocessor systems with preemptive scheduling where tasks are independent and their deadlines are smaller than the periods, the Deadline Monotonic (DM) priority assignment is optimal.

Practical design problems, however, are typically much more complicated. Even if we only consider the problem of optimal priority assignment, there are many examples where finding the optimal solution is of exponential complexity (and DM is no longer guaranteed to be optimal):

- systems with nonpreemptive scheduling [7] or in which tasks share resources [16];
- systems in which the designer seeks an optimal solution that maximizes robustness (for example, defined as the difference between the deadline and the worst case response time) [10]; and
- systems in which preemption thresholds are used to optimize stack memory usage [8].

In addition, there are design problems in which multiple control loops are scheduled on the same processor and the objective function to be optimized is the overall control performance [5] [23].

In general, the optimal design can be obtained by solving an optimization problem where the system performance is maximized within the feasibility region. This approach has been used in several works to optimize the design of real-time systems, e.g., [6] [27] [28]. In this paper, we will use a Mixed Integer Linear Programming (MILP) framework. An MILP problem in standard form is

$$\begin{array}{ll} \text{minimize} & \mathbf{c^T X} \\ \text{subject to} & \mathbf{AX} \leq \mathbf{b} \\ & \mathbf{X} \geq \mathbf{0}, \end{array} \tag{1}$$

where $\mathbf{X} = (x_1, \ldots, x_n)$ is a vector of positive real, integer, or binary-valued decision variables. MILP problems can be solved very efficiently by a variety of solvers. In this paper, we discuss two sets of optimization variables, *the worst-case execution times, and the task priorities.*.

We highlight two contexts in which the *priority assignment* problem cannot be formulated in a trivial way because of additional design variables or constraints. The *first* problem instance is especially relevant for real-time systems-on-a-chip, in which the amount of RAM memory used by the system stack can be reduced by selectively disabling preemption [8]. Preemption can be disabled by artificially imposing critical sections on the task segments that require large amounts of stack space or, alternatively, by using preemption thresholds. In both cases, the problem is defined as the optimal assignment of priorities and

- *H. Zeng is with the Department of Electrical and Computer Engineering, McGill University, Room 633, McConnell Engineering Building, 3480 University Street, Montreal, Quebec H3A 0E9, Canada. E-mail: haibo.zeng@mcgill.ca.*
- *M. Di Natale is with the TECIP Institute, Scuola Superiore S. Anna, via Moruzzi 1, Pisa 56124, Italy. E-mail: marco@sssup.it.*

Manuscript received 27 Oct. 2010; revised 23 Dec. 2011; accepted 30 Dec. 2011; published online 16 Jan. 2012.
Recommended for acceptance by S.H. Son.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-10-0594.
Digital Object Identifier no. 10.1109/TC.2012.21.

possibly preemption thresholds such that schedulability is guaranteed and the memory required by the system is minimized. The *second* problem instance occurs in the context of model-based development flows. The design problem consists of mapping the actions of functional blocks onto tasks and of the assignment of priorities to tasks. The objective is to define a task model that is feasible with respect to deadlines, preserves the communication flows with respect to the model semantics, and minimizes the amount of memory required for the wait-free buffers implementing the communication. We elaborate the problem with an automotive case study in Section 9.3.

There are at least three representative scenarios where the *worst case task execution times* are design variables. The *first* is when the designer can choose among multiple versions of the same algorithm, such as in some control applications, including fuel injection systems. The fuel and air intakes need to be controlled together with the injection time in the combustion chamber to optimize the efficiency of the engine (torque produced) and minimize pollutants. The control algorithm has several tasks activated with a rate that depends on the revolution speed of the engine shaft. At low speeds, the interarrival time is relatively large and sophisticated versions of the control algorithms are used. At higher speeds, schedulability is at risk and less complicated (and faster) versions of the controls are used. The selection of which version of the control algorithm to use at each engine speed can be a quite complex problem. The *second* scenario is dynamic voltage scaling, or at least the class of voltage scaling solutions where the processor clock can be assigned at design time to each task. In this case, the execution time of a task may change according to the clock and voltage selected for its execution. The problem is to minimize the energy consumed by the system while keeping all task response times within the deadlines. The *third* scenario occurs in model-based design flows. The behaviors associated with the model blocks or components (*Runnables* in AUTOSAR or *step functions* in Simulink) need to be mapped into tasks. In multicore systems, there can be many mapping options. A runnable with period T_i can be mapped into any task with the same period or a submultiple of it ($T_j = T_i/k$ with the runnable executing once every k activations) on any core. Depending on the mapping decisions, the worst case task execution times may be different.

In the formulation of the optimization problem, the definition of the feasibility region based on the computation of task response times allows an exact formulation only with a very large set of integer or binary variables (see Section 2.1). This makes the problem practically impossible to solve in reasonable amount of time for industrial-size systems.

In this paper, we provide an improved formulation of the feasibility region for *single-processor priority-based scheduling*, with a much smaller number of disjunctive linear constraints. ***The new test is marginally important as an analysis method (current methods already allow to deal with extremely large task sets), but it is of fundamental importance for an exact and efficient formulation of the real-time feasibility region in an optimization process.***

Our new method extends previous work on schedulability analysis based on the request bound function. The *request bound function* of a task τ_i in $t \geq 0$, $rbf_i(t)$, is defined as the maximum cumulative execution time required by τ_i in any time interval of length t. In [11], a task τ_i with deadline D_i not larger than its period T_i is demonstrated to be feasible if and only if there exists at least one time instant $t \leq D_i$ when the available CPU time in $[0, t]$ is larger than, or equal to the time required for execution by τ_i and higher or equal priority tasks (the set $he(i)$), that is, $\sum_{j \in he(i)} rbf_j(t) \leq t$. For each task τ_i, only time instants belonging to a set $\mathcal{S}_i$ need to be considered for a necessary and sufficient condition. In [11], $\mathcal{S}_i$ is given as the set of integer multiples of the periods of the tasks with higher priority than τ_i that are no greater than D_i. Satisfaction of any of these linear constraints for $t \in \mathcal{S}_i$ demonstrates feasibility of τ_i. In other words, the feasibility region of τ_i can be defined by a set of disjunctive linear constraints, which can be expressed in an MILP formulation by adding a set of binary variables (see Section 3).

A large set $\mathcal{S}_i$ requires the use of many such binary variables. This makes the solution of the optimization problem difficult or even impossible for large-size designs. We show how the set of points that need to be checked for feasibility can be substantially reduced with respect to the previous work in [11], [2].

We provide theory and algorithms for the computation of a set of nonredundant points to represent the feasibility region. All the algorithms presented here are to be *performed offline as part of the design optimization process*.

We performed several experiments to show the effectiveness and applicability of our method. The first experiment uses randomly generated task sets to evaluate the algorithm complexity and the effectiveness in the reduction of points. The method is then applied to an automotive fuel injection system, to compute the schedulable solution within the memory constraint and with minimum latencies. *The computation time of a reduced set of points for this case study is several seconds, but it saves hours of runtime in the following optimization process.*

Section 2 provides an introduction to the definition of the feasibility region and refers to results in the literature. Section 3 discusses the new formulation of the feasibility region and the improvement with respect to existing tests. Sections 4 and 5 present the theory and algorithms to find nonredundant test sets. The use of our formulation for the optimization of the priority assignment is discussed in Section 6. Section 7 defines an extension to the case of tasks sharing critical sections. Systems with preemption threshold scheduling are addressed in Section 8. Section 9 shows the application to random task sets as well as to an industrial case. Finally, we draw conclusions in Section 10.

2 Definition of the Feasibility Region

Consider a set of n periodic tasks $\Gamma = \{\tau_i = (C_i, D_i, T_i) : i = 1, \ldots, n\}$ *scheduled by fixed priority on a uniprocessor system with* $D_i \leq T_i$. A task τ_i is characterized by design parameters, such as its period T_i and deadline D_i. It also involves design variables subject to optimization, such as its priority, or its worst case execution time C_i. For example, the worst case execution time of a task can be approximated by the sum of the worst case execution times of the functions called by the task. This is especially true for tasks generated from models, where the task body consists of calling the system reactions mapped into it. Thus, C_i can be expressed as $C_i = \sum_k \mu_{i,k}\omega_k$, where ω_k is the worst case execution time of the kth reaction, and $\mu_{i,k}$ is 1 if τ_i implements the reaction (and 0 otherwise). We denote the vector of the C_i of all tasks as $\mathbf{C}$, the vector of the priority orders $p_{i,j}$ as $\mathbf{P}$ (see Section 2.1), and the vectors of periods and deadlines as $\mathbf{T}$ and $\mathbf{D}$, respectively.

In this work, the periods $\mathbf{T}$ and deadlines $\mathbf{D}$ are always parameters. The design variables $\mathbf{X}$ can be the computation times $\mathbf{X} \equiv \mathbf{C}$, the priority order $\mathbf{X} \equiv \mathbf{P}$, or both.

Definition 1. *The* feasibility region $\mathbb{M}_i(\Gamma)$ of task $\tau_i \in \Gamma$, *simply denoted as* $\mathbb{M}_i$, *is the region of the design variables* $\mathbf{X} \in \mathbb{X}$ *where* τ_i *is schedulable, i.e.,*

$$\mathbb{M}_i(\Gamma) = \{\mathbf{X} \in \mathbb{X} : \tau_i \; is \; schedulable\}. \tag{2}$$

Definition 2. *The task set* Γ *is schedulable if and only if every task in the set is schedulable. The* feasibility region $\mathbb{M}(\Gamma)$ of the task set Γ, *simply denoted as* $\mathbb{M}$, *is defined as* ($\bigcap$ *stands for the set intersection)*

$$\mathbb{M}(\Gamma) = \{\mathbf{X} \in \mathbb{X} : \Gamma \; is \; schedulable\} = \bigcap_{\tau_i \in \Gamma} \mathbb{M}_i(\Gamma). \tag{3}$$

2.1 Response Time-Based Formulation

The feasibility region for τ_i can be expressed by comparing its worst case response time r_i to the deadline D_i. r_i is given by the well-known formula

$$r_i = C_i + \sum_{j \in hp(i)} I_{j,i} C_j \leq D_i, \tag{4}$$

where $hp(i)$ is set of tasks with priority higher than τ_i, and $I_{j,i}$ is the integer number of interferences by τ_j on τ_i

$$I_{j,i} = \left\lceil \frac{r_i}{T_j} \right\rceil.$$

$I_{j,i}$ can also be expressed using the linear bounds

$$\frac{r_i}{T_j} \leq I_{j,i} < \frac{r_i}{T_j} + 1.$$

If the priority assignments are design variables, then a set of binary variables is defined to encode the priority order: $p_{j,i}$ is defined as 1 if the priority of τ_j is higher than the priority of τ_i, and 0 otherwise. Equation (4) can be reformulated as

$$r_i = C_i + \sum_{j \neq i} p_{j,i} I_{j,i} C_j \leq D_i, \tag{5}$$

where j spans over all the other tasks in the system. The formula is not a linear constraint because of the product $p_{j,i} I_{j,i}$. However, an additional set of integer variables $\Pi_{j,i}$ can be used to transform the problem into a linear one as in (6), even if (5) can be handled by most solvers [27]. Here, M is a constant larger than any $I_{j,i}$ and it is typically used (as "big M" formulation) to encode alternative constraints that depend on a binary variable (the value of $p_{j,i}$ makes one of the constraints trivially true)

$$\begin{aligned} & r_i = C_i + \sum_{j \neq i} \Pi_{j,i} C_j \leq D_i \\ & I_{j,i} - M(1 - p_{j,i}) \leq \Pi_{j,i} \leq I_{j,i} \\ & 0 \leq \Pi_{j,i} \leq M p_{j,i}. \end{aligned} \tag{6}$$

If the worst case task execution times are also optimization variables, other linearization techniques could be used [17]. This formulation of the response time has been proposed in several research works, including [27] and [15] where the feasibility problem is part of the design optimization based on a SAT-solver. The corresponding test is exact, but requires a possibly large number of integer variables (in the order of n^2, where n is the number of tasks). Solving the MILP problem in feasible time is impossible for medium and sometimes small size systems.

In the cited research works, alternative solutions based on the approximation of the response times have been used. Linear upper and lower bounds $r_i^{\uparrow}$ and $r_i^{\downarrow}$ can be obtained by upper and lower bounding the ceiling function and using real values for $I_{j,i}^{\uparrow} = \frac{r_i}{T_j} + 1$ and $I_{j,i}^{\downarrow} = \frac{r_i}{T_j}$, respectively. A linear combination $\tilde{r}_i = \alpha r_i^{\downarrow} + (1 - \alpha) r_i^{\uparrow}$ of these bounds with coefficient $\alpha \in [0, 1)$ is used to optimize the task periods in [6] and the task and message placement and priority assignment in [27].

A tighter upper bound can be obtained by using the definition of the *load executed at higher priority* in [4]

$$r_i^{\uparrow} - \sum_{j \in hp(i)} \left(U_j r_i^{\uparrow} + C_j (1 - U_j) \right) = C_i. \tag{7}$$

Other feasibility tests, especially utilization based, have been developed for single-processor systems. They could be used to obtain a subset of the feasibility region and a suboptimal solution. In some systems, possibly with many tasks and a performance function with small variations, they could be viable. Of course, they don't guarantee the global optimum and, depending on the task set, the region obtained with a sufficient test could be significantly smaller than the exact feasibility region. An estimate of the pessimism can be obtained with reference to the utilization-based hyperbolic bound in [2], which is not convex (therefore, not easily usable for optimization purposes), but is the best bound that can be found using the task utilizations. Its schedulability gain with respect to the classical Liu and Layland bound approaches $\sqrt{2}$ for large task sets. Finally, and possibly most importantly, *these tests typically require knowledge of the priority assignment, therefore cannot be used for the optimization of the priority assignment*.

2.2 Request Bound Function-Based Formulation

The feasibility region $\mathbb{M}$ can also be determined using constraints based on the *request bound function*. In the following, we discuss the optimization of the worst case execution time selection, that is, $\mathbf{X} \equiv \mathbf{C}$. The optimization of the priority assignment is addressed in Section 6.

Since the load from higher priority tasks, as expressed by the request bound function, can only change at integer multiples of their periods, Lehoczky et al. [11] found that the feasibility region $\mathbb{M}_i$ can be expressed as

$$\begin{aligned} & \mathbb{M}_i(\Gamma) = \left\{ \mathbf{C} \in \mathbb{R}_n^+ : \bigvee_{t \in \mathcal{S}_i} \sum_{j \in he(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t \right\} \\ & \text{where } \mathcal{S}_i = \left\{ k T_j : j \in hp(i), k = 1, \ldots, \left\lfloor \frac{D_i}{T_j} \right\rfloor \right\} \bigcup \{D_i\}. \end{aligned} \tag{8}$$

Here, $\bigvee$ stands for the logic OR operation and $\bigcup$ represents the union of sets. The definition of the function

$$\gamma_j(t) = \frac{\left\lceil \frac{t}{T_j} \right\rceil}{t} \tag{9}$$

TABLE 1
An Example with Four Tasks

τ_i	τ_1	τ_2	τ_3	τ_4
$D_i = T_i$	2	5	8	50
$\mathcal{S}_i$	$\{2\}$	$\{2,4,5\}$	$\{2,4,5,6,8\}$	$\{k\cdot 2 : k = 1,\ldots,25\}$ $\bigcup\{k\cdot 5 : k = 1,\ldots,10\}$
$\mathcal{P}_i$	$\{2\}$	$\{4,5\}$	$\{4,5,8\}$	$\{44,45,48,50\}$

allows to rewrite the feasibility condition for τ_i at each point $t \in \mathcal{S}_i$ in a much simpler way

$$\sum_{j\in he(i)} \gamma_j(t)C_j \le 1. \tag{10}$$

Each such inequality represents the case where the total request from task τ_i and higher priority tasks within $[0,t]$ is no greater than t. If this is true at any point $t \in \mathcal{S}_i$ (which implies $t \le D_i$), then τ_i completes no later than t; thus, it is schedulable. The feasibility region of τ_i is the union of subspaces defined in the positive quadrant $\mathbf{R}_n^+$ by the set of (linear) inequalities on the $\mathbf{C}$ variables with parameters $\mathbf{T}$ and $\mathbf{D}$ for $t \in \mathcal{S}_i$. *In general, the feasibility region is a nonconvex polytope.* Furthermore, for large task set, the number of points in $\mathcal{S}_i$ may be very large.

Bini et al. [2] showed that only a subset (denoted as $\mathcal{P}_i(D_i)$) of the points in $\mathcal{S}_i$ needs to be considered for a sufficient and necessary test.

Theorem 1. *Assuming task indices are assigned according to priorities (lowest index for highest priority), the feasibility region $\mathbf{M}$ of task set Γ where $D_i \le T_i$ for each task τ_i is given by ($\bigwedge$ is the logic AND)*

$$\mathbf{M}(\Gamma) = \left\{ \mathbf{C} \in \mathbf{R}_n^+ : \bigwedge_{\tau_i\in\Gamma} \bigvee_{t\in\mathcal{P}_i(D_i)} \sum_{j\in he(i)} \gamma_j(t)C_j \le 1 \right\},$$
$$\text{where } \mathcal{P}_i \text{ is defined as} \tag{11}$$
$$\begin{cases} \mathcal{P}_1(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}\left(\left\lfloor \frac{t}{T_{i-1}} \right\rfloor T_{i-1}\right) \bigcup \mathcal{P}_{i-1}(t). \end{cases}$$

Compared to $\mathcal{S}_i$, the number of points in $\mathcal{P}_i(D_i)$ (or simply $\mathcal{P}_i$) is reduced so that the feasibility analysis becomes faster than the methods based on response time analysis [1], [21].

Example. Consider the example of Table 1 with four tasks indexed by priority (lowest index for highest priority) and with deadlines equal to their periods. For each τ_i, $\mathcal{P}_i$ is a (much smaller) subset of $\mathcal{S}_i$, especially for the last task.

However, ***the set $\mathcal{P}_i$ is still redundant. We show how a smaller set can be computed for a necessary and sufficient feasibility test, or an exact representation of the feasibility region***. The computation of this reduced set requires an algorithm that can be too complex to overcome any possible advantages if used as an analysis tool. However, it can be very useful for the definition of a feasibility region for design optimization purposes: a much smaller set can be computed ***statically once and for all, and the resulting efficient formulation of the exact feasibility region saves significant time in the optimization process.*** In addition, (11) cannot be used for the optimization of the priority assignment, given that knowledge of the priority order is needed to identify the points in $\mathcal{P}_i$.

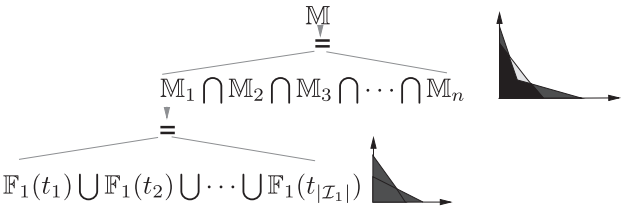


Fig. 1. The system feasibility region as the intersection/union composition of the point regions of the tasks.

3 Improving the Representation of the Feasibility Region: Formulation

As a generalization of (8) and (11), the feasibility region of τ_i with constrained deadline $D_i \le T_i$ can be defined using a set of points $\mathcal{I}_i$ (possibly redundant). $\mathcal{I}_i$ can be $\mathcal{S}_i$ as in (8), or $\mathcal{P}_i(D_i)$ as in (11)

$$\mathbf{M}(\Gamma) = \bigcap_{\tau_i\in\Gamma} \mathbf{M}_i(\Gamma)$$
$$\mathbf{M}_i(\Gamma) = \left\{ \mathbf{C} \in \mathbf{R}_n^+ : \bigvee_{t\in\mathcal{I}_i} \sum_{j\in he(i)} \gamma_j(t)C_j \le 1 \right\}. \tag{12}$$

We denote each of the halfspaces defined by one of the linear inequalities in (12) as

$$\mathbf{F}_i(\Gamma,t) = \left\{ \mathbf{C} \in \mathbf{R}_n^+ : \sum_{j\in he(i)} \gamma_j(t)C_j \le 1 \right\}. \tag{13}$$

$\mathbf{F}_i(\Gamma,t)$, or simply $\mathbf{F}_i(t)$, is called a *point region* as it corresponds to a time point t. Then, the feasibility region can be expressed as

$$\mathbf{M} = \bigcap_{\tau_i\in\Gamma} \bigcup_{t\in\mathcal{I}_i} \mathbf{F}_i(t). \tag{14}$$

Fig. 1 provides a graphical representation of the system feasibility region. It is the intersection of the task regions, which are in turn the union of the point regions.

We index the points $t_{j,m} \in \mathcal{I}_j$ in increasing order, such that $t_{j,0}$ denotes the smallest, and t_{j,m_j-1} the largest, where $m_j = |\mathcal{I}_j|$. We use $k_j = \lceil \log_2 m_j \rceil$ binary variables $b_{j,k}, k = 0,\ldots,k_j-1$ to encode these m_j constraints. The following function associates a binary encoding for each variable $b_{j,k}$ with the constraint on point $t_{j,m}$

$$E_j(m,k) = \begin{cases} b_{j,k} & \text{if } (m \text{ bitwise-AND } 2^k) = 0 \\ 1 - b_{j,k} & \text{otherwise.} \end{cases} \tag{15}$$

Thus, the disjunction on constraints that define the schedulability of τ_j can be encoded using the standard "big-M" formulation for conditional constraints (M is a constant larger than any other quantity involved in the constraint). For all τ_j, $j = 1\ldots n$

$$\begin{cases} \forall t_{j,m} \in \mathcal{I}_j, \sum_{i\in he(j)} \gamma_i(t_{j,m})C_i \le 1 + M\sum_{k=0}^{k_j-1} E_j(m,k) \\ \sum_{k=0}^{k_j-1} (2^k \times b_{j,k}) \le m_j - 1. \end{cases} \tag{16}$$

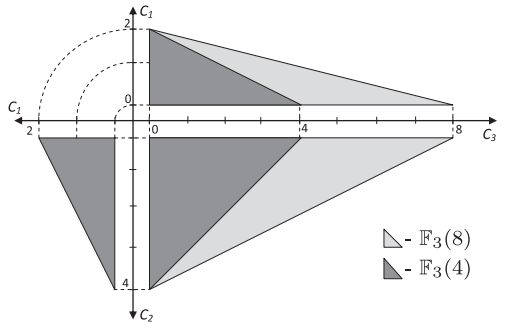


Fig. 2. Orthogonal projection views of $\mathbb{F}_3(4)$ and $\mathbb{F}_3(8)$.

All the constraints in the first set of inequalities are trivially true except the one for which the values of the binary variables match the corresponding index. The bottom constraint in (16) ensures that the binary encoding does not exceed the maximum index of the points.

Given the dependency of the number of binary variables on the number of constraints, our objective is to remove redundancy in $\mathcal{I}_i$, by checking whether the removal of the points (and the corresponding point regions) changes the feasibility region. We define three types of redundancies:

- *System-level point redundancy*. For any $p \in \mathcal{I}_i$, if the resulting region $\mathbb{M}^-$ after removing p from $\mathcal{I}_i$ is the same as $\mathbb{M}$, then p is system-level redundant.
- *System-level task redundancy*. For any task τ_i, if the resulting region $\mathbb{M}^-$ after removing all points in $\mathcal{I}_i$ is the same as $\mathbb{M}$, then τ_i is system-level redundant.
- *Task-level point redundancy*. For any $p \in \mathcal{I}_i$, if the resulting region $\mathbb{M}_i^-$ after removing p from $\mathcal{I}_i$ is the same as $\mathbb{M}_i$, then p is task-level redundant.

The first two types require the formulation of the feasibility region of other tasks. The third is a subset of the system-level point redundancy: any point that is task-level redundant is also system-level redundant. However, its identification is much faster. In the following, we provide methods to efficiently remove these redundancies.

4 Task-Level Point Redundancy

The points in $\mathcal{I}_i$ correspond to linear inequalities that are OR-combined to define the feasibility region $\mathbb{M}_i$ of τ_i. *It is therefore possible to remove all the points that define point regions contained in the union of the other regions* (the satisfaction of their constraints is implied by the satisfaction of constraints on other points).

Fig. 2 shows the point regions $\mathbb{F}_3(4)$ and $\mathbb{F}_3(8)$ for task τ_3 in Table 1. It is clear that $\mathbb{F}_3(4)$ is fully contained inside $\mathbb{F}_3(8)$, thus point 4 is redundant. Also, from Fig. 3, although neither $\mathbb{F}_4(48)$ nor $\mathbb{F}_4(50)$ contains $\mathbb{F}_4(44)$, their union does. Thus, point 44 can be removed from $\mathcal{I}_4$.

We now provide a lemma to detect feasibility regions fully contained in other regions.

Lemma 2. *Considering two regions $\mathbb{G} = \{\mathbf{C} \in \mathbb{R}_n^+ : \sum_{i=1}^n g_i C_i \leq 1\}$ and $\mathbb{H} = \{\mathbf{C} \in \mathbb{R}_n^+ : \sum_{i=1}^n h_i C_i \leq 1\}$. If all the coefficients of the constraint defining $\mathbb{G}$ are componentwise no smaller than $\mathbb{H}$, i.e., $\forall i = 1, 2, \ldots, n$, $g_i \geq h_i$, then $\mathbb{G}$ is contained in $\mathbb{H}$, or $\mathbf{G} \subseteq \mathbf{H}$.*

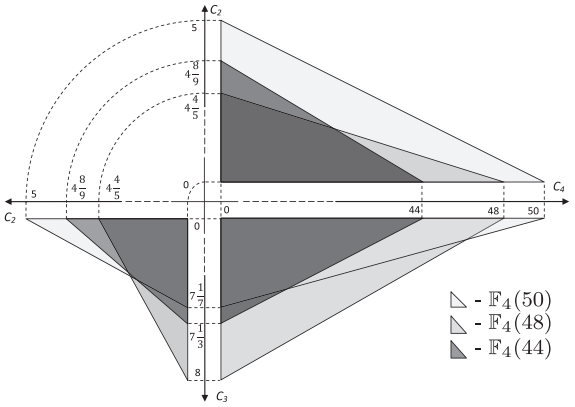


Fig. 3. Orthogonal projection views of $\mathbb{F}_4(44)$, $\mathbb{F}_4(48)$, and $\mathbb{F}_4(50)$. Axis C_1 is omitted.

Proof. For any $\mathbf{C}^* \in \mathbb{G}$, since $\forall i$, $C_i^* \geq 0$ and $g_i \geq h_i$, $\sum_{i=1}^n g_i C_i^* - \sum_{i=1}^n h_i C_i^* = \sum_{i=1}^n (g_i - h_i) C_i^* \geq 0$, thus

$$\sum_{i=1}^{n} g_i C_i^* \leq 1 \Rightarrow \sum_{i=1}^{n} h_i C_i^* \leq 1,$$

and $\mathbf{C}^* \in \mathbb{H}$. In other words, any member of $\mathbb{G}$ is also a member of $\mathbb{H}$. □

Lemma 2 can be leveraged to reason about the containment relationship between the feasibility regions defined by the points in $\mathcal{I}_i$. The simplest case is to check the componentwise dominance between the coefficients associated with a pair of points p and q.

Theorem 3. *The set of redundant points in $\mathcal{I}_i$ includes every point p for which there exists a point q such that*

$$\forall j \in he(i), \gamma_j(p) \geq \gamma_j(q). \tag{17}$$

Proof. By the definitions of $\mathbb{F}_i(p)$ and $\mathbb{F}_i(q)$, it follows Lemma 2 that if condition (17) is satisfied, then $\mathbb{F}_i(p) \subseteq \mathbb{F}_i(q)$. Thus, point p is redundant. □

Example. In the task set of Table 1, the set of points that need to be checked for the feasibility of τ_3 is reduced from $\mathcal{P}_3 = \{4, 5, 8\}$ to obtain $\mathcal{I}_3 = \{5, 8\}$ since the pair $p = 4, q = 8$ satisfies (17) (point 8 makes 4 redundant).

τ_i	τ_1	τ_2	τ_3	τ_4
$\mathcal{P}_i$	$\{2\}$	$\{4, 5\}$	$\{4, 5, 8\}$	$\{44, 45, 48, 50\}$
$\mathcal{I}_i$ Theorem 3	$\{2\}$	$\{4, 5\}$	$\{5, 8\}$	$\{44, 45, 48, 50\}$

The following theorem extends Theorem 3 to a more general case: a sufficient condition that the feasibility region defined by a point p is contained in the union of the feasibility regions defined by other points in $\mathcal{I}_i$.

Theorem 4. *A point $p \in \mathcal{I}_i$ is redundant if for all the other points $q_k \in \mathcal{I}_i, q_k \neq p$, there exists a solution $\alpha = (\alpha_1, \ldots, \alpha_{|\mathcal{I}_i|-1})$ which satisfies the following constraints:*

$$\begin{cases} \forall j \in he(i), \gamma_j(p) \geq \sum\limits_k \gamma_j(q_k)\alpha_k \\ \sum\limits_k \alpha_k = 1 \\ \forall k, \alpha_k \geq 0. \end{cases} \tag{18}$$

Proof. Consider the region $\mathbb{H} = \{\mathbf{C} \in \mathbb{R}_n^+ : \sum_{j \in hp(i)} \beta_j C_j \leq 1\}$ where the coefficient of the constraint $\beta_j = \sum_k \alpha_k \gamma_j(q_k)$ is the affine combination of the coefficients associated with points q_k. We first prove that $\mathbb{H}$ is a subset of the feasibility region $\bigcup_k \mathbf{F}_i(q_k)$ by contradiction.

Suppose there exists a solution $\mathbf{C}^*$ such that $\mathbf{C}^* \in \mathbb{H}$, and $\mathbf{C}^* \notin \bigcup_k \mathbf{F}_i(q_k)$. Then,

$$\forall k, \sum_{j \in he(i)} \gamma_j(q_k) C_j^* > 1.$$

By multiplying each inequality (for one of the q_k) with the corresponding α_k of the solution of (18) and adding the left- and the right-hand sides for all q_k, we obtain

$$\sum_k \alpha_k \sum_{j \in he(i)} \gamma_j(q_k) C_j^* = \sum_{j \in he(i)} C_j^* \sum_k \alpha_k \gamma_j(q_k) > \sum_k \alpha_k.$$

By the definition of β_j and the fact that $\sum_k \alpha_k = 1$, the above inequality is equivalent to $\sum_{j \in he(i)} \beta_j C_j^* > 1$. This contradicts the assumption $\mathbf{C}^* \in \mathbb{H}$, thus $\mathbb{H} \subseteq \bigcup_k \mathbf{F}_i(q_k)$.

Since $\forall j \in he(i), \gamma_j(p) \geq \sum_k \gamma_j(q_k)\alpha_k = \beta_j$, by Lemma 2, $\mathbf{F}_i(p) \subseteq \mathbb{H}$. Thus, $\mathbf{F}_i(p) \subseteq \bigcup_k \mathbf{F}_i(q_k)$, and point p is redundant. □

Example. For τ_4 in the example of Table 1, the test at 44 is redundant. The tests at points 44, 48, and 50 for τ_4 are

$$\mathbf{F}_4(44) = \left\{\mathbf{C} \in \mathbf{R}_4^+ : \frac{22}{44}C_1 + \frac{9}{44}C_2 + \frac{6}{44}C_3 + \frac{1}{44}C_4 \leq 1\right\}$$
$$\mathbf{F}_4(48) = \left\{\mathbf{C} \in \mathbf{R}_4^+ : \frac{24}{48}C_1 + \frac{10}{48}C_2 + \frac{6}{48}C_3 + \frac{1}{48}C_4 \leq 1\right\}$$
$$\mathbf{F}_4(50) = \left\{\mathbf{C} \in \mathbf{R}_4^+ : \frac{25}{50}C_1 + \frac{10}{50}C_2 + \frac{7}{50}C_3 + \frac{1}{50}C_4 \leq 1\right\}.$$

The following affine combination of $\mathbf{F}_4(48)$ and $\mathbf{F}_4(50)$

$$\frac{6}{11}\mathbf{F}_4(48) + \frac{5}{11}\mathbf{F}_4(50) = \left\{\mathbf{C} \in \mathbf{R}_4^+ : \frac{22}{44}C_1 + \frac{9}{44}C_2 + \frac{5.8}{44}C_3 + \frac{0.9}{44}C_4 \leq 1\right\}$$

specifies a region containing $\mathbf{F}_4(44)$ since the coefficients in the constraint are no greater than the ones in the constraint of $\mathbf{F}_4(44)$. After applying Theorem 5, the set of points is

τ_i	τ_1	τ_2	τ_3	τ_4
$\mathcal{I}_i$ Theorem 3	$\{2\}$	$\{4,5\}$	$\{5,8\}$	$\{44,45,48,50\}$
$\mathcal{I}_i$ Theorem 5	$\{2\}$	$\{4,5\}$	$\{5,8\}$	$\{45,48,50\}$

The computation of the solutions to (17) or (18) can be further simplified by checking only the subset of $he(i)$ with period less than T_i plus τ_i itself.

Theorem 5. *Equation (18) is true if the following condition is true:*

$$\begin{cases} \forall j \in (he(i) \bigcap \{l : T_l < T_i\}) \bigcup \{i\}, \gamma_i(p) \geq \sum_k \gamma_j(q_k)\alpha_k \\ \sum_k \alpha_k = 1 \\ \forall k, \alpha_k \geq 0. \end{cases} \tag{19}$$

Proof. Since T_i is larger than or equal to any point in $\mathcal{I}_i$, from (9) it is $\gamma_i(p) = \frac{1}{p}$ and $\gamma_i(q_k) = \frac{1}{q_k}$. A solution α to (19) satisfies the first constraint for τ_i, therefore

$$\gamma_i(p) = \frac{1}{p} \geq \sum_k \gamma_i(q_k)\alpha_k = \sum_k \frac{\alpha_k}{q_k}.$$

For task $\tau_j \in he(i) \bigcap \{j : T_j \geq T_i\}$, we also have $\gamma_j(p) = \frac{1}{p}$ and $\gamma_i(q_k) = \frac{1}{q_k}$. Hence, satisfaction of the constraint for τ_i also implies satisfaction of the constraint for τ_j. □

Theorems 4 and 5 require knowledge of the set of higher priority tasks. However, they can be used in a priority assignment optimization at the price of some pessimism (redundancy) by including all tasks in (17) and (18), instead of those with higher or equal priority only ($j \in he(i)$).

5 System-Level Redundancy

The set of points that are nonredundant at task level should be further reduced by finding all the points $\mathcal{I}_i^R \subseteq \mathcal{I}_i$ that are redundant considering constraints on *other tasks*. Informally, some points may define regions that are not contained in the regions of the other points for the same task, but they bring no contribution to the system feasibility region because one or more other task is infeasible in this additional portion of the solution space.

We define $\mathcal{I}_i^{NR} = \mathcal{I}_i \setminus \mathcal{I}_i^R$. We consider two cases, one in which the redundant points $\mathcal{I}_i^R$ are a strict subset of $\mathcal{I}_i$ (*system-level point redundancy*), the other case is when the entire set $\mathcal{I}_i$ is redundant (*system-level task redundancy*).

Case 1. If $\mathcal{I}_i^R \subset \mathcal{I}_i$, then we denote the regions defined by the redundant and nonredundant point sets as $\mathbb{M}_i^R = \bigcup_{t \in \mathcal{I}_i^R} \mathbf{F}_i(t)$ and $\mathbb{M}_i^{NR} = \bigcup_{t \in \mathcal{I}_i^{NR}} \mathbf{F}_i(t)$. Thus, $\mathcal{I}_i^R$ is redundant if and only if $\mathbb{M}$ is not altered by its removal

$$\mathbb{M} = \mathbb{M}_i \bigcap_{j \neq i} \mathbb{M}_j = \mathbb{M}_i^{NR} \bigcap_{j \neq i} \mathbb{M}_j. \tag{20}$$

Since $\mathbb{M}_i = \mathbb{M}_i^R \bigcup \mathbb{M}_i^{NR}$, the condition can be rewritten as

$$\begin{aligned} &\left(\mathbb{M}_i^R \bigcap_{j \neq i} \mathbb{M}_j\right) \bigcup \left(\mathbb{M}_i^{NR} \bigcap_{j \neq i} \mathbb{M}_j\right) = \mathbb{M}_i^{NR} \bigcap_{j \neq i} \mathbb{M}_j \\ &\Leftrightarrow \mathbb{M}_i^R \bigcap_{j \neq i} \mathbb{M}_j \subseteq \mathbb{M}_i^{NR} \bigcap_{j \neq i} \mathbb{M}_j \\ &\Leftrightarrow \mathbb{M}_i^R \bigcap \overline{\mathbb{M}}_i^{NR} \bigcap_{j \neq i} \mathbb{M}_j = \emptyset, \end{aligned} \tag{21}$$

where $\overline{\mathbb{M}}_i^{NR}$ denotes the complementary region of $\mathbb{M}_i^{NR}$.

Case 2. If $\mathcal{I}_i^R = \mathcal{I}_i$ (the feasibility of τ_i is implied by that of other tasks), then $\mathcal{I}_i^R$ is redundant if and only if

$$\mathbb{M}_i \bigcap_{j \neq i} \mathbb{M}_j = \bigcap_{j \neq i} \mathbb{M}_j \Leftrightarrow \bigcap_{j \neq i} \mathbb{M}_j \subseteq \mathbb{M}_i \Leftrightarrow \overline{\mathbb{M}}_i \bigcap_{j \neq i} \mathbb{M}_j = \emptyset. \tag{22}$$

5.1 System-Level Point Redundancy

A point $p \in \mathcal{I}_i$ is *system-level redundant* if and only if the region $\mathbf{F}_i(p)$ is contained in the union of the regions for the other points in $\mathcal{I}_i$ where *all the other tasks are schedulable*.

Theorem 6. *A point $p \in \mathcal{I}_i$ is nonredundant at system level if and only if there exists a solution $\mathbf{C}$ such that*

- *p is necessary for τ_i to be feasible at $\mathbf{C}$, i.e.*
  - $\mathbf{C} \in \mathbf{F}_i(p)$
  - $\forall q \neq p \in \mathcal{I}_i, \mathbf{C} \notin \mathbf{F}_i(q)$
- *all other tasks τ_j are schedulable at $\mathbf{C}$.*

Proof. It follows the definition of system-level point redundancy. If $\mathcal{I}_i^R = \{p\}$, the conjunction of the above conditions is the same as the intersection on (21). The solution $\mathbf{C}$ demonstrates that this intersection is not empty. □

In practice, we could search for a feasible solution $\mathbf{C}$ to the following problem:

$$\begin{cases} \mathbf{C} \in \mathbf{F}_i(p): & \sum_{k \in he(i)} \gamma_k(p) C_k \leq 1 \\ \forall q_i \neq p \in \mathcal{I}_i, \mathbf{C} \notin \mathbf{F}_i(q_i): & \sum_{k \in he(i)} \gamma_k(q_i) C_k > 1 \\ \forall j \neq i, \mathbf{C} \in \mathbf{M}_j: & \bigvee_{t \in \mathcal{I}_j} \sum_{k \in he(j)} \gamma_k(t) C_k \leq 1 \\ \text{the disjunction is encoded in} \\ \quad \text{an MILP formulation using (16)} & \mathbf{C} \geq 0. \end{cases} \tag{23}$$

If such a problem has no solution, the point is redundant, otherwise it is nonredundant.

The constraints $\mathbf{C} \geq 0$ can be tightened if additional knowledge is available. For example, bounds on $\mathbf{C}$ can be obtained considering all the possible task implementations. If the feasibility region is defined for the optimal assignment of priorities, then the $\mathbf{C}$ values may be fully determined with a drastic reduction in the number of points.

Actually, only the subset of tasks with indices in $\{j : D_j < D_i\}$ is needed to check the system-level point redundancy for task τ_i (as the third constraint in (23)).

Theorem 7. *A point $p \in \mathcal{I}_i$ is nonredundant if and only if there exists a solution to the following problem:*

$$\begin{cases} \mathbf{C} \in \mathbf{F}_i(p): & \sum_{k \in he(i)} \gamma_k(p) C_k \leq 1 \\ \forall q_i \neq p \in \mathcal{I}_i, \mathbf{C} \notin \mathbf{F}_i(q_i): & \sum_{k \in he(i)} \gamma_k(q_i) C_k > 1 \\ \forall j \neq i, D_j < D_i, \mathbf{C} \in \mathbf{M}_j: \\ \quad \textit{Formulated as in (23)} & \mathbf{C} \geq 0. \end{cases} \tag{24}$$

Proof. Only if. If $p \in \mathcal{I}_i$ is nonredundant, there exists a solution $\mathbf{C}$ to (23) which is also a feasible solution to (24).

If. Assume there exists a solution $\mathbf{C}$ to (24). Consider another solution $\mathbf{C}'$ where $C'_j = C_j, \forall j \in he(i)$, and $C'_j = 0$, $\forall j \notin he(i)$. $\mathbf{C}'$ satisfies the first two set of constraints in (23) as they only depend on the values of $C_j, \forall j \in he(i)$.

We now prove $\mathbf{C}'$ satisfies the third constraint of (23). $\forall j$ where $D_j < D_i$, $\mathbf{C}' \in \mathbf{M}_j$ because $\mathbf{C} \in \mathbf{M}_j$ and $\mathbf{C}'$ is componentwise $\leq \mathbf{C}$. $\forall j$ where $D_j \geq D_i$, since $\mathbf{C}' \in \mathbf{F}_i(p)$ and $C'_j = 0$, $\forall j \geq i$, task τ_j with a deadline $D_j \geq D_i$ must be schedulable at p, which implies $\mathbf{C}' \in \mathbf{M}_j$. □

Example. The points to check in our example are further reduced. Point 45 in the test set of τ_4 is redundant since the corresponding problem (24) has no feasible solution.

τ_i	τ_1	τ_2	τ_3	τ_4
$\mathcal{I}_i$ Theorem 5	$\{2\}$	$\{4, 5\}$	$\{5, 8\}$	$\{45, 48, 50\}$
$\mathcal{I}_i$ Theorem 7	$\{2\}$	$\{4, 5\}$	$\{5, 8\}$	$\{48, 50\}$

Unfortunately, Theorems 6 and 7 cannot be applied for priority optimization without changes, given that they require knowledge of the priority order of tasks. In Section 6, we will show how to overcome this limitation.

5.2 System-Level Task Redundancy

From (22), since $\mathbf{M} = \bigcap_{\tau_j \in \Gamma} \mathbf{M}_j$, the region $\mathbf{M}_i$ does not contribute to $\mathbf{M}$ if and only if the feasibility region of the other tasks is a subset of $\mathbf{M}_i$. We first provide some simple results to simplify the detection of redundant tasks.

Theorem 8. *The lowest priority task τ_n is nonredundant.*

Proof. Consider $\mathbf{C} = (0, \ldots, 0, D_n + \epsilon)$ where $\epsilon > 0$. At $\mathbf{C}$, τ_n is not schedulable ($\mathbf{C} \notin \mathbf{M}_n$), but all other tasks are schedulable ($\mathbf{C} \in \bigcap_{j \neq n} \mathbf{M}_j$). Thus, $\bigcap_{j \neq n} \mathbf{M}_j \not\subseteq \mathbf{M}_n$. □

Theorem 9. *Task τ_i is redundant if a lower priority task has a smaller or equal deadline ($\exists j, i \in hp(j)$ with $D_j \leq D_i$).*

Proof. If task τ_j is schedulable, then τ_i is also schedulable. Thus, $\mathbf{M}_j \subseteq \mathbf{M}_i$ and τ_i is redundant. □

Theorem 10. *A task τ_i is redundant if it is not the lowest priority task and $D_i = T_i = lcm_{j \in he(i)}(T_j)$.*

Proof. $\forall j \in he(i)$, $\gamma_j(T_i) = \lceil \frac{T_i}{T_j} \rceil \frac{1}{T_i} = \frac{T_i}{T_j} \frac{1}{T_i} = \frac{1}{T_j}$. Also, $\forall p < T_i$, p and $q = T_i$ satisfy the condition (17); thus, the feasibility region of τ_i is $\mathbf{M}_i = \mathbf{F}_i(T_i)$, which is $\sum_{j \in he(i)} \frac{C_j}{T_j} \leq 1$. This is a trivial necessary condition for feasibility and defines a superset of the feasibility region of any task with priority lower than τ_i. □

The main result that allows to define the procedure for the removal of task redundancy is the following.

Theorem 11. *A task τ_i is nonredundant at the system level if and only if there exists a solution $\mathbf{C}$ such that*

- *τ_i is not schedulable at $\mathbf{C}$;*
- *all the other tasks are schedulable at $\mathbf{C}$.*

Proof. The proof follows immediately from the definition of system-level task redundancy. From (22), a task is redundant if and only if $\overline{\mathbf{M}}_i \bigcap_{j \neq i} \mathbf{M}_j = \emptyset$. □

The existence of such a solution $\mathbf{C}$ can be checked by the feasibility of the following MILP problem:

$$\begin{cases} \mathbf{C} \notin \mathbf{M}_i: & \sum_{k \in he(i)} C_k \gamma_k(q_i) > 1, \forall q_i \in \mathcal{I}_i \\ \forall j \neq i, \mathbf{C} \in \mathbf{M}_j: \\ \quad \text{Formulation as in (23)} & \mathbf{C} \geq 0. \end{cases} \tag{25}$$

Example. For the example in Table 1, τ_1 is redundant since it satisfies the condition in Theorem 10 and τ_2 is redundant since there is no feasible solution for (25).

τ_i	τ_1	τ_2	τ_3	τ_4
$\mathcal{I}_i$ Theorem 7	$\{2\}$	$\{4, 5\}$	$\{5, 8\}$	$\{48, 50\}$
$\mathcal{I}_i$ Theorems 10 and 11	$\emptyset$	$\emptyset$	$\{5, 8\}$	$\{48, 50\}$

5.3 Overall Procedure

The previous conditions allow the removal of redundant points without changing the feasibility region. The following Theorem 15 proves that the feasibility region after removing all the redundant points does not depend on the removal order of the individual redundant points or tasks, as long as *point* redundancy (task- and system-level) is removed before *task* redundancy.

The following lemma is used to prove that the feasibility region does not change if point redundancy is removed before task redundancy (**step 1** in Theorem 15). This lemma

(with the other lemmas and Theorem 15) is based on set theory. For our purposes, the sets are feasibility regions.

Lemma 12. *Given the feasibility regions* $\mathbf{M}_1$, $\mathbf{F}_{1R}$, $\mathbf{M}_R$, *and* $\mathbf{M}_2$, *if* $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap \mathbf{M}_R \bigcap \mathbf{M}_2 = \mathbf{M}_1 \bigcap \mathbf{M}_2$, *then* $\mathbf{M}_1 \bigcap \mathbf{M}_R \bigcap \mathbf{M}_2 = \mathbf{M}_1 \bigcap \mathbf{M}_2$.

Proof. $\mathbf{M}_1$ can be rewritten as $\mathbf{M}_1 \bigcap (\mathbf{M}_R \bigcup \overline{\mathbf{M}_R})$. Thus, $\mathbf{M}_1 \bigcap \mathbf{M}_2 = (\mathbf{M}_1 \bigcap \mathbf{M}_R \bigcap \mathbf{M}_2) \bigcup (\mathbf{M}_1 \bigcap \overline{\mathbf{M}_R} \bigcap \mathbf{M}_2)$. Therefore, it must also be $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap \mathbf{M}_R \bigcap \mathbf{M}_2 = \mathbf{M}_1 \bigcap \mathbf{M}_2 = (\mathbf{M}_1 \bigcap \mathbf{M}_R \bigcap \mathbf{M}_2) \bigcup (\mathbf{M}_1 \bigcap \overline{\mathbf{M}_R} \bigcap \mathbf{M}_2)$.

The intersection of $\mathbf{M}_1 \bigcap \overline{\mathbf{M}_R} \bigcap \mathbf{M}_2$ (on the right-hand side) with the left-hand side $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap \mathbf{M}_R \bigcap \mathbf{M}_2$ is clearly empty. Hence, $\mathbf{M}_1 \bigcap \overline{\mathbf{M}_R} \bigcap \mathbf{M}_2 = \emptyset$, and $\mathbf{M}_1 \bigcap \mathbf{M}_2 = \mathbf{M}_1 \bigcap \mathbf{M}_R \bigcap \mathbf{M}_2$. □

In Lemma 12, $\mathbf{F}_{1R}$ and $\mathbf{M}_R$ represent the feasibility regions for a redundant point and a redundant task, respectively, in $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap \mathbf{M}_R \bigcap \mathbf{M}_2$, it is safe to remove $\mathbf{F}_{1R}$ before $\mathbf{M}_R$.

The following lemma shows that point redundancy can be removed in any order (**step 2** in Theorem 15).

Lemma 13. *Given the feasibility regions* $\mathbf{M}_1$, $\mathbf{F}_{1R}$, $\mathbf{M}_2$, $\mathbf{F}_{2R}$, *and* $\mathbf{M}_3$, *if* $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap (\mathbf{M}_2 \bigcup \mathbf{F}_{2R}) \bigcap \mathbf{M}_3 = \mathbf{M}_1 \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3$, *then* $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3 = \mathbf{M}_1 \bigcap (\mathbf{M}_2 \bigcup \mathbf{F}_{2R}) \bigcap \mathbf{M}_3 = \mathbf{M}_1 \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3$.

Proof. $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap (\mathbf{M}_2 \bigcup \mathbf{F}_{2R})$ can be expanded as

$$(\mathbf{M}_1 \bigcap \mathbf{M}_2) \bigcup (\mathbf{M}_1 \bigcap \mathbf{F}_{2R}) \bigcup (\mathbf{F}_{1R} \bigcap \mathbf{M}_2) \bigcup (\mathbf{F}_{1R} \bigcap \mathbf{F}_{2R}).$$

The equality between $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap (\mathbf{M}_2 \bigcup \mathbf{F}_{2R}) \bigcap \mathbf{M}_3$ and $\mathbf{M}_1 \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3$ implies $\mathbf{F}_{1R} \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3 \subseteq \mathbf{M}_1 \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3$, thus $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3 = (\mathbf{M}_1 \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3) \bigcup (\mathbf{F}_{1R} \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3) = \mathbf{M}_1 \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3$. Similarly, $\mathbf{M}_1 \bigcap (\mathbf{M}_2 \bigcup \mathbf{F}_{2R}) \bigcap \mathbf{M}_3 = \mathbf{M}_1 \bigcap \mathbf{M}_2 \bigcap \mathbf{M}_3$. □

If $\mathbf{F}_{1R}$ and $\mathbf{F}_{2R}$ are redundant (point) regions in the representation of the system feasibility region $(\mathbf{M}_1 \bigcup \mathbf{F}_{1R}) \bigcap (\mathbf{M}_2 \bigcup \mathbf{F}_{2R}) \bigcap \mathbf{M}_3$, then they can be removed in arbitrary order.

The following lemma proves that task redundancy can be removed in any order (**step 3** in Theorem 15).

Lemma 14. *For any regions* $\mathbf{M}_{1R}$, $\mathbf{M}_{2R}$, *and* $\mathbf{M}$, *if* $\mathbf{M} = \mathbf{M}_{1R} \bigcap \mathbf{M}_{2R} \bigcap \mathbf{M}$, *then* $\mathbf{M} = \mathbf{M}_{1R} \bigcap \mathbf{M} = \mathbf{M}_{2R} \bigcap \mathbf{M}$.

Proof. Since $\mathbf{M}_{1R} \bigcap \mathbf{M}_{2R} \bigcap \mathbf{M} \subseteq \mathbf{M}_{1R} \bigcap \mathbf{M} \subseteq \mathbf{M}$, then $\mathbf{M}_{1R} \bigcap \mathbf{M}_{2R} \bigcap \mathbf{M} = \mathbf{M}_{1R} \bigcap \mathbf{M} = \mathbf{M}$. Likewise, $\mathbf{M}_{2R} \bigcap \mathbf{M} = \mathbf{M}$. □

We now present the main result in this section.

Theorem 15. *Given a set of redundant points* $\mathcal{I}^R = \{\mathcal{I}_1^R \subseteq \mathcal{I}_1, \ldots, \mathcal{I}_n^R \subseteq \mathcal{I}_n\}$, *we partition the indices of its subsets:*

- $A_1 = \{i : \mathcal{I}_i^R = \emptyset\}$, *i.e., no redundancy in* $\mathcal{I}_i$;
- $A_2 = \{i : \mathcal{I}_i^R = \mathcal{I}_i\}$, *i.e., remove all points in* $\mathcal{I}_i$ *(all points are redundant, that is, the task is redundant);*
- $A_3 = \{i : \mathcal{I}_i^R \subset \mathcal{I}_i, \mathcal{I}_i^R \neq \emptyset\}$ *(a subset of the points is redundant).*

The set $\mathcal{I}^R$ *can be removed by first checking* $\mathcal{I}_i^R$ *for* $i \in A_3$ *in any order, then* $\mathcal{I}_i^R$ *for* $i \in A_2$ *in any order.*

Proof. For any subset $B \subseteq \{1, \ldots, n\}$ of task indices, we denote the corresponding feasibility region as $\mathbf{M}_B = \bigcap_{i \in B} \mathbf{M}_i$. Thus, we denote the feasibility regions $\mathbf{M}_{A_1} = \bigcap_{i \in A_1} \mathbf{M}_i$, $\mathbf{M}_{A_2} = \bigcap_{i \in A_2} \mathbf{M}_i$, $\mathbf{M}_{A_3} = \bigcap_{i \in A_3} \mathbf{M}_i$.

Step 1. The set $\mathcal{I}^R$ can be removed by first checking $\mathcal{I}_i^R$ for $i \in A_3$, then $\mathcal{I}_i^R$ for $i \in A_2$. This step follows Lemma 12 by extending it to the general case of an arbitrary number (possibly more than two) of tasks.

$\forall i \in A_3$, the set $\mathcal{I}_i^R$ is a strict subset of $\mathcal{I}_i$. Hence, $\mathbf{M}_{A_3} = \bigcap_{i \in A_3} (\mathbf{M}_i^R \bigcup \mathbf{M}_i^{NR})$. The conjunction can be expanded resulting in $2^{|A_3|}$ terms. We denote $\mathbf{M}_{A_3}^{NR} = \bigcap_{i \in A_3} \mathbf{M}_i^{NR}$, and the union of the other $2^{|A_3|} - 1$ terms as $\mathbf{M}_{A_3}^{R}$. The union of the two gives $\mathbf{M}_{A_3} = \mathbf{M}_{A_3}^{R} \bigcup \mathbf{M}_{A_3}^{NR}$. By definition, the simultaneous removal of all redundant points $\mathcal{I}_i^R, i = 1, \ldots, n$ does not change the feasibility region

$$\mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3} = \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_3}^{NR}. \tag{26}$$

By (26) and Lemma 12 where we set $\mathbf{M}_1 = \mathbf{M}_{A_3}^{NR}$, $\mathbf{F}_{1R} = \mathbf{M}_{A_3}^{R}$, $\mathbf{M}_R = \mathbf{M}_{A_2}$, and $\mathbf{M}_2 = \mathbf{M}_{A_1}$,

$$\begin{aligned} \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3} &= \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3}^{NR} \\ &= \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_3}^{NR}. \end{aligned} \tag{27}$$

This means that the point redundancies in $\mathcal{I}_i^R$ for $i \in A_3$ can be removed before the task redundancies, and the feasibility region remains the same during the process, including the intermediate result $\mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3}^{NR}$.

Step 2. The points in the sets $\mathcal{I}_i^R, i \in A_3$ can be removed in any order. This step follows by generalizing Lemma 13 to any combination of point redundancies.

For any subset $\mathcal{B} \subseteq \mathcal{I}_i$, we denote the feasibility region resulting from the disjunction of the regions defined by the points $p \in \mathcal{B}$ as $\mathbf{F}_i(\mathcal{B}) = \bigcup_{p \in \mathcal{B}} \mathbf{F}_i(p)$. Suppose we have removed an arbitrary combination $\mathcal{I}_i^{R_1} \subseteq \mathcal{I}_i^R$ of the redundant points for tasks in A_3. We denote $\mathcal{I}_i^R \setminus \mathcal{I}_i^{R_1}$ as $\mathcal{I}_i^{R_2}$. The feasibility region is $\mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3} = \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap_{i \in A_3} (\mathbf{M}_i^{NR} \bigcup \mathbf{F}_i(\mathcal{I}_i^{R_1}) \bigcup \mathbf{F}_i(\mathcal{I}_i^{R_2}))$.

The conjunction on the right-hand side can be expanded. One of the terms is $\mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap_{i \in A_3} (\mathbf{M}_i^{NR} \bigcup \mathbf{F}_i(\mathcal{I}_i^{R_2}))$, thus, it is a subset of $\mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3}$. Likewise,

$$\begin{aligned} \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3}^{NR} &= \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap_{i \in A_3} \mathbf{M}_i^{NR} \\ &\subseteq \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap_{i \in A_3} (\mathbf{M}_i^{NR} \bigcup \mathbf{F}_i(\mathcal{I}_i^{R_2})). \end{aligned}$$

However, $\mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3}^{NR}$ and $\mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3}$ are the same by the definition of redundant points, hence

$$\begin{aligned} \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3} &= \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3}^{NR} \\ &= \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap_{i \in A_3} (\mathbf{M}_i^{NR} \bigcup \mathbf{F}_i(\mathcal{I}_i^{R_2})). \end{aligned} \tag{28}$$

Now, if we remove another arbitrary redundant point p in any task in A_3, i.e., $\forall p \in \mathcal{I}_j^{R_2}, j \in A_3$, by applying the same reasoning used to derive (28), we can prove that

$$\begin{aligned} \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3} &= \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}_{A_3}^{NR} \\ &= \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap_{i \in A_3, i \neq j} (\mathbf{M}_i^{NR} \bigcup \mathbf{F}_i(\mathcal{I}_i^{R_2})) \\ &\quad \bigcap (\mathbf{M}_j^{NR} \bigcup \mathbf{F}_i(\mathcal{I}_i^{R_2} \setminus \{p\})) \\ &= \mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap_{i \in A_3} (\mathbf{M}_i^{NR} \bigcup \mathbf{F}_i(\mathcal{I}_i^{R_2})). \end{aligned} \tag{29}$$

The last equivalence of (29) says that the region does not change by removing a combination of point redundancies for tasks in A_3, then removing the redundant point p for τ_j. Because (29) holds for an arbitrary p and any task $j \in A_3$, we can remove point redundancies one by one in any order.

Step 3. The sets $\mathcal{I}_i, i \in A_2$ can be removed in any order. This step follows by extending Lemma 14 to the case that any number of tasks is system-level redundant.

Suppose the redundancy from A_3 is removed, and the resulting representation of the feasibility region is $\mathbf{M}_{A_1} \bigcap \mathbf{M}_{A_2} \bigcap \mathbf{M}^{NR}_{A_3}$. $\forall i \in A_2$, we consider an arbitrary subset $A'_2 \subseteq A_2 \setminus \{i\}$, and denote $A''_2 = A_2 \setminus (\{i\} \bigcup A'_2)$. By (27) and Lemma 14, if we define $\mathbf{M}_{1R} = \mathbf{M}_{A'_2}$, $\mathbf{M}_{2R} = \mathbf{M}_{A''_2} \bigcap \mathbf{M}_i$, and $\mathbf{M} = \mathbf{M}_{A_1} \bigcap \mathbf{M}^{NR}_{A_3}$,

$$\begin{aligned}&\mathbf{M}_{A_1} \bigcap \mathbf{M}^{NR}_{A_3} \bigcap \mathbf{M}_{A''_2} \bigcap \mathbf{M}_i \bigcap \mathbf{M}_{A'_2} \\ &= \mathbf{M}_{A_1} \bigcap \mathbf{M}^{NR}_{A_3} \bigcap \mathbf{M}_{A''_2} \bigcap \mathbf{M}_i.\end{aligned}$$

Similarly, if we define $\mathbf{M}_{1R} = \mathbf{M}_{A''_2}$, $\mathbf{M}_{2R} = \mathbf{M}_i$, and $\mathbf{M} = \mathbf{M}_{A_1} \bigcap \mathbf{M}^{NR}_{A_3}$, by Lemma 14, we obtain

$$\mathbf{M}_{A_1} \bigcap \mathbf{M}^{NR}_{A_3} \bigcap \mathbf{M}_{A''_2} \bigcap \mathbf{M}_i = \mathbf{M}_{A_1} \bigcap \mathbf{M}^{NR}_{A_3} \bigcap \mathbf{M}_{A''_2}. \quad (30)$$

Therefore, the feasibility region is not altered by removing task redundancy for the tasks with index in A'_2, then removing redundancy for task τ_i. Because (30) holds for an arbitrary i and any set $A'_2 \subseteq A_2 \setminus \{i\}$, we can remove task redundancy for all tasks in A_2 in any order. □

The procedure to find a nonredundant set of points is summarized in Algorithm 1. First, we reduce the task-level point redundancy as in Theorem 3 (lines 2-6) and 5 (lines 7-11). Then, the system-level point redundancy is exhaustively explored for each task τ_i and each point $p \in \mathcal{I}_i$ by checking the feasibility of (24) (lines 13-19). After applying Theorems 9 and 10 (lines 20-24), the system-level task redundancy is removed by checking the feasibility of (25) (lines 25-29).

Algorithm 1. Remove redundancy from the test set for Γ

```
 1: for each task τ_i do
 2:     for each p ∈ I_i do
 3:         if ∃q > p which satisfies (17) then
 4:             I_i = I_i \ {p}
 5:         end if
 6:     end for
 7:     for each p ∈ I_i do
 8:         if (19) is feasible then
 9:             I_i = I_i \ {p}
10:         end if
11:     end for
12: end for
13: for each task τ_i do
14:     for each p ∈ I_i do
15:         if ∄C which satisfies (24) then
16:             I_i = I_i \ {p}
17:         end if
18:     end for
19: end for
20: for each task τ_i except the lowest priority one do
21:     if ∃j, (i ∈ hp(j) and D_j ≤ D_i) or D_i = T_i = lcm_{j∈he(i)} T_j
        then
22:         I_i = ∅
23:     end if
24: end for
25: for each task τ_i do
26:     if ∄C which satisfies (25) then
27:         I_i = ∅
28:     end if
29: end for
```

Our algorithm gives a set of nonredundant points by starting from an initial (redundant) set, such as the set $\mathcal{P}_i$ [2]. There is currently no guarantee that the produced set is unique or consequently it contains the minimum number of points. For a full-dimensional convex polytope, the minimal halfspace representation is in fact unique and is given by the set of facet-defining halfspaces [9]. Our conjecture is that the nonredundant set of the (nonconvex) feasibility region provided here is unique. As a support to this conjecture, we did not find any counterexample in the experiments.

Unfortunately, there is no decisive answer whether the problem of finding a nonredundant representation for the feasibility region is NP-complete. The problem of finding a nonredundant representation for a nonconvex polytope is known to be strongly NP-hard. The complexity of our algorithm depends on the number of points in the initial set, and in the worst case the number of points in $\mathcal{P}_i$ [2] is exponential to the number of tasks/periods. But, of course, there might be a more efficient algorithm than ours.

6 Optimizing the Priority Assignment

The definition of the set $\mathcal{P}_i$ in [2] and the identification of system-level redundancy presented in the previous section require knowledge of the task priority assignment, which makes them unsuitable when the assignment of priorities is among the optimization variables. In this section, we discuss how to remove some, but not necessarily all the system-level redundancy ***regardless of the priority assignment***. The method is based on the observation that, because of the optimality of the deadline-monotonic priority assignment when $D_i \leq T_i$, the feasibility region defined by any priority order is a subset of the feasibility region defined by the DM assignment.

Algorithm 2 describes the overall procedure to find a sufficient test set for task τ_i. We consider the set of tasks $\Gamma_{D_i} = \{\tau_m : D_m < D_i\} \bigcup \{\tau_i\}$ with deadline smaller than τ_i plus itself. First, a starting set $\mathcal{I}_m$ is identified for each task $\tau_m \in \Gamma_{D_i}$ by following the rules for $\mathcal{S}_m$ in (8) assuming that all the tasks with smaller period than τ_m (the only ones that can generate points in $\mathcal{S}_m$) have higher priority than τ_m. Then, task-level point redundancy is removed for τ_m by checking the conditions in (17) and (18). This is done also with the pessimistic assumption that all the tasks with smaller period than τ_m have higher priority than τ_m. Finally, system-level point redundancy from $\mathcal{I}_i$ is removed by checking the feasibility of (24) where $\Gamma = \Gamma_{D_i}$. We assume priorities are assigned to tasks $\tau_m \in \Gamma_{D_i}$ by DM. Since the condition that a point is nonredundant includes that all tasks $\in \Gamma_{D_i}$ other than τ_i are schedulable, a DM assignment to Γ_{D_i} avoids removing a point that is possibly nonredundant in any priority assignment.

Algorithm 2. Find a sufficient test set for task τ_i for any priority assignment

1: $\Gamma_{D_i} = \{\tau_m : D_m < D_i\} \bigcup \{\tau_i\}$
2: **for** each task $\tau_m \in \Gamma_{D_i}$ **do**
3: $\quad \mathcal{I}_m = \{kT_j : T_j < T_m, k = 1, \ldots, \lfloor \frac{D_m}{T_j} \rfloor\} \bigcup \{D_m\}$
4: $\quad$ **for** each $p \in \mathcal{I}_m$ **do**
5: $\quad\quad$ **if** $\exists q \neq p$ which satisfies (17) assuming $hp(m) = \{\tau_j : T_j < T_m\}$ **then**
6: $\quad\quad\quad \mathcal{I}_m = \mathcal{I}_m \setminus \{p\}$
7: $\quad\quad$ **end if**
8: $\quad\quad$ **if** (18) is feasible assuming $hp(m) = \{\tau_j : T_j < T_m\}$ **then**
9: $\quad\quad\quad \mathcal{I}_m = \mathcal{I}_m \setminus \{p\}$
10: $\quad\quad$ **end if**
11: $\quad$ **end for**
12: **end for**
13: Assign priorities to tasks $\in \Gamma_{D_i}$ using deadline-monotonic policy
14: **for** each $p \in \mathcal{I}_i$ **do**
15: $\quad$ **if** $\nexists \mathbf{C}$ which satisfies (24) where $\Gamma = \Gamma_{D_i}$ **then**
16: $\quad\quad \mathcal{I}_i = \mathcal{I}_i \setminus \{p\}$
17: $\quad$ **end if**
18: **end for**

The correctness of Algorithm 2 in the sense that the resulting test set $\mathcal{I}_i$ for task τ_i is sufficient for any fixed priority assignment can be formally demonstrated.

Theorem 16. *For any priority assignment to Γ, Algorithm 2 computes a sufficient test set $\mathcal{I}_i$ for τ_i.*

Proof. Step 1. For any priority assignment, $\forall \tau_m \in \Gamma_{D_i}$, $\mathcal{I}_m = \{kT_j : T_j < T_m, k = 1, \ldots, \lfloor \frac{D_m}{T_j} \rfloor\} \bigcup \{D_m\}$ is a sufficient test set. It is an immediate consequence of the derivation of the set $\mathcal{S}_m$ in [11], also referred here as (8). The request bound function only changes at multiples of the task periods and only needs to be considered up to D_m. If $T_j \geq T_m$, then $\lfloor \frac{D_m}{T_j} \rfloor \leq \lfloor \frac{T_m}{T_j} \rfloor \leq 1$, i.e., any integer multiple of T_j either equals D_m or is too large to be considered.

Step 2. The task-level redundancy removal from $\mathcal{I}_m$ does not change $\mathbb{M}_m$; thus, the resulting test set is still sufficient to define $\mathbb{M}_m$. This directly follows Theorem 4. Moreover, we reduce the task-level redundancy from $\mathcal{I}_m$ based on the set of tasks in $\{\tau_j : T_j < T_m\}$, which is a superset of $\{\tau_j : \tau_j \in hp(m), T_j < T_m\}$ for any priority assignment in Theorem 5; thus, the satisfaction of (17) and (18) with $hp(m) = \{\tau_j : T_j < T_m\}$ is a sufficient condition to prove $p \in \mathcal{I}_m$ is task-level redundant.

Step 3. By the optimality of the DM priority assignment, assignment, if any static priority scheduling algorithm can meet all the deadlines, so will DM. Hence, the feasibility region $\bigcap_{j \in \Gamma_{D_i} \setminus \{\tau_i\}} \mathbb{M}_j$ obtained by a DM assignment to $\Gamma_{D_i} \setminus \{\tau_i\}$ is a superset of the feasibility region under any other priority assignment.

Step 4. As in (24) (third constraint), to check whether a point $p \in \mathcal{I}_i$ is redundant, it is sufficient to check the schedulability of tasks $\{\tau_j : D_j < D_i\}$. For the conditions in (24) where $\Gamma = \Gamma_{D_i}$, if τ_i is infeasible for $p \in \mathcal{I}_i$ with a DM assignment to $\{\tau_j : D_j < D_i\}$, it is also infeasible under any other priority assignment. Thus, it is safe to remove p from $\mathcal{I}_i$ when (24) is infeasible under DM. □

The resulting test set from Algorithm 2 represents the *exact feasibility region* regardless of the priority assignment, but is possibly redundant.

7 Scheduling with Blocking Times

Our definition of feasibility region applies to fully preemptive periodic task sets executed independently. However, tasks may share resources such as physical devices, or logical resources like data structures for communication using critical sections protected by mechanisms with bounded blocking time like the Priority Inheritance protocol or the Priority Ceiling protocol [20], included (in its immediate form) in the OSEK operating system standard [18].

In the priority ceiling protocol, each resource is assigned with a *ceiling priority* as the highest priority among tasks accessing it. The worst case blocking time B_i for τ_i with priority π_i is the duration of the longest critical section from a lower priority task τ_j on a resource shared with τ_i or a task with priority higher than π_i (a resource with a ceiling priority no lower than π_i)

$$B_i = \max_{j \in lp(i)} \max_k \beta_{j,k}, \tag{31}$$

where $lp(i)$ is the set of tasks with lower priority than τ_i, and $\beta_{j,k}$ is the duration of the critical section of task τ_j on the shared resource R_k with a ceiling no lower than π_i.

Once the worst case blocking time is formulated, the exact schedulability condition can be extended by adding the blocking term B_i to the task execution time [20]

$$\exists t \leq D_i, B_i + C_i + \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t. \tag{32}$$

Theorem 1 can be extended to consider blocking times [2].

Theorem 17 (from [2]). *Assuming task indices are assigned according to their priorities, the task set Γ where $D_i \leq T_i$ is schedulable* ***if and only if***

$$\bigwedge_{\tau_i \in \Gamma} \bigvee_{t \in \mathcal{P}_i(D_i)} B_i + \sum_{j \in he(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t. \tag{33}$$

Our main results can be easily extended to the case with blocking times. When the design variable only includes $\mathbf{C}$, the B_i terms (as computed in (31)) are constant when the size and type of the critical sections are not affected by the optimization result (the selected task implementation). For example, critical sections may be library functions, or their implementation only depends on the communication data and is independent from the versions of the control algorithms. If the blocking times depend on the optimization variables, these dependencies need to be encoded as additional constraints in the problems (24) and (25).

When also optimizing the priority assignment, assignments, B_i depends on the priority level. This dependency must be encoded with additional constraints and used during the point/task reduction for the definition of the feasibility region. An MILP formulation of the B_i term can be done as follows: A binary **parameter** $U_{j,k}$ is defined as 1 if task τ_j uses resource R_k and 0 otherwise. A binary **variable** $p_{i,j}$ is defined as 1 if τ_i has a higher priority than τ_j. Also, a set of real-valued constants $\beta_{j,k}$ indicates the length of the critical section executed by τ_j on R_k. By these definitions,

$$B_i = \max_{h\neq i, j\neq i,k} \beta_{j,k} \text{ where } (U_{j,k} \wedge p_{i,j}) \wedge (U_{h,k} \wedge p_{h,i}) = 1.$$

Given that the U values are constant, the terms on which the maximum is computed can be 0 or the product $\beta_{j,k}p_{i,j}p_{h,i}$. This term can be directly handled by some solvers, but can also be linearized as follows.

The product xb of a real variable x and a binary variable b can be reformulated with a real variable y and the additional constraints $x - M(1-b) \leq y \leq x$ and $0 \leq y \leq Mb$ (M is a constant greater than the upper bound of x).

As for the max function, the generic definition $Y = \max_i\{y_i\}$ can be simply translated to a set of constraints $Y \geq y_i$ for all the possible y_i, when Y only appears on the left-hand side of "$\leq$" constraints, and the cost function (to be minimized) is monotonically nondecreasing with Y. Otherwise, a more complex formulation is required: an additional binary variable a_i is associated with each y_i, which is defined as 1 if $Y = y_i$, and 0 otherwise. The relation $Y = \max_i\{y_i\}$ is formulated by the constraints

$$\forall i, y_i \leq Y \leq y_i + M(1-a_i)$$
$$\sum_i a_i = 1.$$

Finally, the term B_i in (32) and (33) can be replaced by an upper bound B^{ub} (the largest critical section in the system) to achieve a sufficient-only condition.

8 Preemption Threshold Scheduling

Preemption may be selectively disabled among tasks using several methods. The most popular is the preemption threshold [24] supported by the ThreadX kernel [22]. A preemption threshold scheduling model may have several practical motivations, including improving the schedulability of lower priority tasks, reducing the required amount of stack space, and improving predictability of the worst case execution time by preventing context switches in places that heavily affect cache line reuse.

8.1 Analysis of Tasks with Preemption Thresholds

For a task τ_i with priority π_i (*higher values correspond to higher priorities*), the preemption threshold is defined as $\theta_i \geq \pi_i$. A task τ_i can preempt another task τ_j only if its priority π_i is higher than θ_j. This means that a higher priority task may wait for a lower priority (but higher preemption threshold) task to finish execution. Therefore, the worst case blocking time for task τ_i is the maximum execution time of the lower priority tasks ($\pi_j < \pi_i$) with a preemption threshold ($\pi_i \leq \theta_j$)

$$B_i = \max_{\pi_j<\pi_i\leq\theta_j} C_j. \tag{34}$$

The response time analysis for tasks with preemption threshold was proposed in [24] and later fixed in [19]. Tasks are scheduled with two different modes. When ready, they are scheduled with their priority, but as soon as they start execution, they behave as if their priority is raised to the preemption threshold. Accordingly, the analysis is performed in two steps. First, the worst case start time is computed. Then, the worst case finish time. The worst case start time S_i and finish time F_i of the first instance of τ_i can be calculated by the following [24]:

$$\begin{aligned} S_i &= B_i + \sum_{j\in hp(i)} \left(1 + \left\lfloor \frac{S_i}{T_j} \right\rfloor\right) C_j \\ F_i &= S_i + C_i + \sum_{\pi_j>\theta_i} \left(\left\lceil \frac{F_i}{T_j} \right\rceil - 1 - \left\lfloor \frac{S_i}{T_j} \right\rfloor \right) C_j. \end{aligned} \tag{35}$$

Unfortunately (as explained in [19]), the first instance is not necessarily the one with the largest response time. All instances in the busy period of level π_i need to be considered. The start time and finish time for the instances $q = 0, 1, \ldots, q^*$ in the busy period are

$$\begin{aligned} S_i(q) &= B_i + \sum_{j\in hp(i)} \left(1 + \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor\right) C_j + qC_i \\ F_i(q) &= S_i(q) + C_i + \sum_{\pi_j>\theta_i} \left(\left\lceil \frac{F_i(q)}{T_j} \right\rceil - 1 - \left\lfloor \frac{S_i(q)}{T_j} \right\rfloor \right) C_j. \end{aligned} \tag{36}$$

The worst case response time of τ_i is computed as $R_i = \max_{q=0,\ldots,q^*}\{R_i(q) = F_i(q) - qT_i\}$, where $q^* = \lfloor \frac{L_i}{T_i} \rfloor$ is the largest index in the level-i busy period L_i. L_i can be calculated by the iterative formula $L_i = B_i + \sum_{j\in he(i)} \lceil \frac{L_i}{T_i} \rceil C_j$.

From (36), the amount of workload from higher priority tasks before the completion of τ_i depends on the finish time F_i as well as the start time S_i.

The need to consider the first q^* instances, together with the fact that the number q^* is not known a priori, results in excessive complexity for optimization purposes. Given that the linearization of the exact feasibility region is exceedingly complex, we look for lower and upper bounds corresponding, respectively, to sufficient-only (pessimistic) and necessary-only (optimistic) conditions.

Pessimistic and optimistic tests can be used for the definition of lower and upper bounds to the feasibility region, which can be used inside procedures for finding good quality solutions to the optimization problem. For example, the optimum solution can be found with a local search starting from the two optimum points (on the upper and lower bounds). With luck, the optimum on the upper bound could be feasible so that it would be the optimal solution to the problem. With respect to this possibility, it is important to estimate how optimistic the necessary test is, and how far apart are the three region boundaries. The experimental section will discuss this issue.

There is an additional limitation to the study on preemption thresholds. The definition of the region in the domain of the priority assignment is complicated due to the dependency of the preemption thresholds on the priority levels. We do not have a solution for this problem, and the only optimization variable discussed for this model is $\mathbf{C}$.

8.1.1 A (Pessimistic) Lower Bound on the Feasibility Region in a Preemption Threshold Scheduling Model

A *sufficient condition* for the schedulability of τ_i is that τ_i is schedulable assuming it is fully preemptive, i.e., its preemption threshold is the same as its priority.

Theorem 18. *Assuming task indices are assigned according to their priorities, the task set Γ where $D_i \leq T_i$ is schedulable* ***if***

$$\bigwedge_{\tau_i\in\Gamma} \bigvee_{t\in\mathcal{P}_i(D_i)} B_i + \sum_{j\in he(i)} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t, \tag{37}$$

where B_i is computed as in (34). The set $\mathcal{P}_i(D_i)$ can be reduced using the same method as in the fully preemptive case. The formulation of the blocking time requires the additional constraints $B_i \geq C_j, \forall j \in \{j : \pi_j < \pi_i \leq \theta_j\}$. These constraints do not need additional binary variables, since the term B_i is only involved in the left side of the "$\leq$" constraints in (37) with positive coefficients.

8.1.2 An (Optimistic) Upper Bound on the Feasibility Region in a Preemption Threshold Scheduling Model

A *necessary condition* for the schedulability of task τ_i is that its first instance in the busy period is schedulable.

The representation of the feasibility region requires computing pairs of points (s, f), where s and f are possible start and finish times. The following theorem extends (8) to preemption threshold scheduling, and defines conditions for the feasibility of the first instance in the busy period.

Theorem 19. *The feasibility region $\mathbb{M}_i$ for the first instance of τ_i is expressed as $\mathbf{C} \in \mathbb{R}_n^+ : \bigvee_{(s,f)\in\mathcal{I}_i}$*

$$\begin{cases} B_i + \sum_{j\in hp(i)} \left\lceil \frac{s}{T_j} \right\rceil C_j < s \\ B_i + \sum_{j\in hnt(i)} \left\lceil \frac{s}{T_j} \right\rceil C_j + \sum_{j\in ht(i)} \left\lceil \frac{f}{T_j} \right\rceil C_j + C_i \leq f, \end{cases} \tag{38}$$

where the sets $\mathcal{I}_i$, $\mathcal{S}_i$, and $\mathcal{F}_i$ are defined as the set of candidate point pairs, start and finish times for τ_i, respectively.

$$\mathcal{I}_i = \{(s, f) : s \in \mathcal{S}_i, f \in \mathcal{F}_i, s \leq f\}$$
$$\mathcal{S}_i = \left\{ kT_j : j \in hp(i), k = 1, \ldots, \left\lfloor \frac{D_i}{T_j} \right\rfloor \right\} \bigcup \{D_i\}$$
$$\mathcal{F}_i = \left\{ kT_j : j \in ht(i), k = 1, \ldots, \left\lfloor \frac{D_i}{T_j} \right\rfloor \right\} \bigcup \{D_i\}.$$

The set $ht(i) = \{j : \pi_j > \theta_i\}$ represents the set of tasks with priority higher than the preemption threshold of τ_i, and $hnt(i) = \{j : \pi_i < \pi_j \leq \theta_i\} = hp(i) \backslash ht(i)$ represents the complement of $ht(i)$ with respect to $hp(i)$.

Proof. In order for the first instance of τ_i to be schedulable, there must exist a pair of points $s \leq f \leq D_i$ such that the instance can start before or at s and finish before or at f

$$\begin{aligned} &B_i + \sum_{j\in hp(i)} \left(1 + \left\lfloor \frac{s}{T_j} \right\rfloor\right) C_j \leq s \\ &B_i + \sum_{j\in hnt(i)} \left(1 + \left\lfloor \frac{s}{T_j} \right\rfloor\right) C_j + \sum_{j\in ht(i)} \left\lceil \frac{f}{T_j} \right\rceil C_j + C_i \leq f. \end{aligned} \tag{39}$$

We define the following functions:

$$\Sigma_i(s) = \frac{B_i + \sum_{j\in hp(i)} \left(1 + \lfloor \frac{s}{T_j} \rfloor\right) C_j}{s}$$
$$\Phi_i(s, f) = \frac{B_i + \sum_{j\in hnt(i)} \left(1 + \lfloor \frac{s}{T_j} \rfloor\right) C_j + \sum_{j\in ht(i)} \lceil \frac{f}{T_j} \rceil C_j + C_i}{f}.$$

$\Sigma_i(s)$ and $\Phi_i(s, f)$ are strictly decreasing with respect to s except for points in $\mathcal{S}_i$. $\Phi_i(s, f)$ is strictly decreasing with respect to f except for points in $\mathcal{F}_i^+ = \{(kT_j)^+ | j \in ht(i), k = 1, \ldots, \lfloor \frac{D_i}{T_j} \rfloor\} \bigcup \{D_i^+\}$, where $(kT_j)^+$ denotes a number infinitely close, but still larger than kT_j. Thus, to satisfy (39), there must be at least one local minimum of $\Sigma_i(s)$ and $\Phi_i(s, f)$ less than or equal to 1.

Local minima of Σ_i belong to the set $\mathcal{S}_i^- = \{(kT_j)^- | j \in hp(i), k = 1, \ldots, \lfloor \frac{D_i}{T_j} \rfloor\} \bigcup \{D_i^-\}$. Hence,

$$\bigvee_{s\in\mathcal{S}_i^-} B_i + \sum_{j\in hp(i)} \left(1 + \left\lfloor \frac{s}{T_j} \right\rfloor\right) C_j \leq s. \tag{40}$$

Local minima of Φ_i belongs to the set $\mathcal{F}_i$, thus

$$\bigvee_{\substack{f\in\mathcal{F}_i \\ f\geq s}} B_i + \sum_{j\in hnt(i)} \left(1 + \left\lfloor \frac{s}{T_j} \right\rfloor\right) C_j + \sum_{j\in ht(i)} \left\lceil \frac{f}{T_j} \right\rceil C_j + C_i \leq f. \tag{41}$$

The equivalence $1 + \lfloor \frac{s^-}{T_j} \rfloor = \lceil \frac{s}{T_j} \rceil$ holds true for all points $s = kT_j$. Therefore, (40) and (41) can be replaced by (38), which concludes the proof. □

For convenience, we denote the halfspace defined by the linear inequality in (38) associated with the request bound function for the start and finish times as

$$\mathbb{S}_i(\Gamma, s) = \left\{ \mathbf{C} \in \mathbb{R}_n^+ : B_i + \sum_{j\in hp(i)} \left\lceil \frac{s}{T_j} \right\rceil C_j < s \right\}$$
$$\mathbb{F}_i(\Gamma, s, f) = \left\{ \mathbf{C} \in \mathbb{R}_n^+ : B_i + \sum_{j\in hnt(i)} \left\lceil \frac{s}{T_j} \right\rceil C_j + \sum_{j\in ht(i)} \left\lceil \frac{f}{T_j} \right\rceil C_j + C_i \leq f \right\}.$$

Their intersection is $\mathbb{I}_i(\Gamma, s, f) = \mathbb{S}_i(\Gamma, s) \bigcap \mathbb{F}_i(\Gamma, s, f)$. The short notations $\mathbb{S}_i(s)$, $\mathbb{F}_i(s, f)$, and $\mathbb{I}_i(s, f)$ are used for $\mathbb{S}_i(\Gamma, s)$, $\mathbb{F}_i(\Gamma, s, f)$, and $\mathbb{I}_i(\Gamma, s, f)$, respectively. Thus, the feasibility region of τ_i is

$$\mathbb{M}_i = \bigcup_{(s,f)\in\mathcal{I}_i} \mathbb{I}_i(s, f) = \bigcup_{(s,f)\in\mathcal{I}_i} \left(\mathbb{S}_i(s) \bigcap \mathbb{F}_i(s, f)\right). \tag{42}$$

8.1.3 Removing Redundancy in the Point Pairs Defining Start and Finish Times

The reduction of point pairs involves a more complicated definition of the possible types of redundancy:

- *Task-level finish time redundancy*. For any finish time $f \in \mathcal{F}_i$, if the resulting region $\mathbb{M}_i^-$ after removing f from $\mathcal{F}_i$ is the same as $\mathbb{M}_i$.
- *Task-level start time redundancy*. For any start time $s \in \mathcal{S}_i$, if the resulting region $\mathbb{M}_i^-$ after removing s from $\mathcal{S}_i$ is the same as $\mathbb{M}_i$.
- *System-level start time redundancy*. For any start time $s \in \mathcal{S}_i$, if the resulting region $\mathbb{M}^-$ after removing s from $\mathcal{S}_i$ is the same as $\mathbb{M}$.
- *Task-level point pair redundancy*. For any start/finish time pair $(s, f) \in \mathcal{I}_i$, if the resulting region $\mathbb{M}_i^-$ after removing (s, f) from $\mathcal{I}_i$ is the same as $\mathbb{M}_i$.
- *System-level point pair redundancy*. For any start/finish time pair $(s, f) \in \mathcal{I}_i$, if the resulting region $\mathbb{M}^-$ after removing (s, f) from $\mathcal{I}_i$ is the same as $\mathbb{M}$.
- *System-level task redundancy*: for any task τ_i, if the resulting region $\mathbb{M}^-$ after removing all points in $\mathcal{I}_i$ is the same as $\mathbb{M}$.

In the following, we develop theorems to find these redundancies. The proofs of some theorems are omitted as they are similar to those provided for the preemptive case in Sections 4 and 5. Proof details can be found in [25].

Finding task-level finish-time redundancy. Each candidate finish time $f \in \mathcal{F}_i$ only defines the second linear constraint as in $\mathbb{F}_i(s,f)$. Since the regions $\mathbb{F}_i(s,f)$ for each pair (s,f) are OR-ed to define the feasibility region of τ_i, to prove f is redundant, it is sufficient to prove that for any $s \in \mathcal{S}_i$, the region $\mathbb{F}_i(s,f)$ is contained in the union of all other regions $\mathbb{F}_i(s,g)$, $g \in \mathcal{F}_i, g \neq f$.

Theorem 20. *A point $f \in \mathcal{F}_i$ is redundant if for all other points $g_k \in \mathcal{F}_i, g_k \neq f$, there exists a solution $\alpha = (\alpha_1, \ldots, \alpha_{|\mathcal{F}_i|-1})$ which satisfies the constraints*

$$\begin{cases} \forall j \in ht(i) \bigcup \{i\}, \quad \gamma_j(f) \geq \sum_k \gamma_j(g_k)\alpha_k \\ \sum_k \alpha_k = 1 \\ \forall k, \alpha_k \geq 0. \end{cases} \tag{43}$$

Proof. $\forall s \in \mathcal{S}_i$, we define $G_i = B_i + \sum_{j \in hnt(i)} \lceil \frac{s}{T_j} \rceil C_j + C_i$. Given that $f \leq D_i \leq T_i \Longrightarrow \gamma_i(f) = \frac{1}{f}\lceil \frac{f}{T_i} \rceil = \frac{1}{f}$, the second inequality in (38), which defines the halfspace $\mathbb{F}_i(s,f)$, is then rewritten as $\gamma_i(f)G_i + \sum_{j \in ht(i)} \gamma_j(f)C_j \leq 1$.

As in Theorem 4, if (43) admits a solution, then the region $\mathbb{F}_i(s,f)$ is contained inside the union of the $\mathbb{F}_i(s,g_k)$ regions defined by all other points $g_k \in \mathcal{F}_i$, i.e., $\forall s, \mathbb{F}_i(s,f) \subseteq \bigcup_k \mathbb{F}_i(s,g_k)$.

Hence, $\forall s \in \mathcal{S}_i$, the region $\mathbb{I}_i(s,f)$ is contained by the union of the regions $\mathbb{I}_i(s,g_k)$ defined by the other pairs (s,g_k), and f can be safely removed from $\mathcal{F}_i$. □

A theorem similar to Theorem 3 can be derived for the reduction of the finish time candidate points.

Theorem 21. *The set of redundant points in $\mathcal{F}_i$ includes all points f for which there exists another point g such that*

$$\forall j \in ht(i) \bigcup \{i\} \quad \gamma_j(f) \geq \gamma_j(g). \tag{44}$$

Finding task-level start-time redundancy. Each start time $s \in \mathcal{S}_i$ defines both linear constraints in (38). Consequently, the condition for establishing the redundancy of s are more stringent than those for finish times. To safely remove s from $\mathcal{S}_i$, not only the region $\mathbb{S}_i(s)$ must be contained in the union of the halfspaces $\mathbb{S}_i(t)$ for a subset of points t in $\mathcal{S}_i$, but $\mathbb{F}_i(s,f)$ should be contained in the regions $\mathbb{F}_i(t,f)$ for the same set of points t.

Theorem 22. *A point $s \in \mathcal{S}_i$ is redundant if for all other points $t_k \leq t_{k_{\max}}$, $t_k \neq s$, there exists a solution $\alpha = (\alpha_1, \ldots, \alpha_{t_{k_{\max}}})$ which satisfies the following constraints:*

$$\begin{cases} \forall j \in he(i) \quad \gamma_j(s) \geq \sum_{k=1}^{k_{\max}} \gamma_j(t_k)\alpha_k \\ \sum_{k=1}^{k_{\max}} \alpha_k = 1 \\ \forall k, \alpha_k \geq 0, \end{cases} \tag{45}$$

where $t_k \in \mathcal{S}_i$ are indexed by increasing order, and

$$k_{\max} = \max_k \left(\forall j \in hnt(i), \left\lceil \frac{t_k}{T_j} \right\rceil \leq \left\lceil \frac{s}{T_j} \right\rceil \right). \tag{46}$$

Finding system-level start-time redundancy. The set of start times that are nonredundant at task level can be further reduced by removing all the points that are made redundant by the feasibility region of *other tasks*.

Theorem 23. *A point $s \in \mathcal{S}_i$ is system-level redundant if there is no solution $\mathbf{C}$ such that ($k_{\max}$ is defined as in (46))*

- $\mathbf{C} \in \mathbb{S}_i(s)$
- $\forall t_k \leq t_{k_{\max}}, t_k \neq s, \mathbf{C} \notin \mathbb{S}_i(t_k)$
- *all other tasks τ_j are schedulable at $\mathbf{C}$.*

The feasibility region $\mathbb{M}_j$ of task τ_j can be formulated as in the fully preemptive case. $k_j = \lceil \log_2 m_j \rceil$ binary variables are used to encode the $m_j = |\mathcal{I}_j|$ constraints (one for each point pair in $\mathcal{I}_j$). The function $E_j(m,k)$ in (15) associates the values of the variables $b_{j,k}$ with each constraint on a point pair $(s_{j,m}, f_{j,m})$.

$$\forall (s_{j,m}, f_{j,m}) \in \mathcal{I}_j,$$
$$\begin{cases} B_j + \sum_{i \in hp(j)} \left\lceil \frac{s_{j,m}}{T_i} \right\rceil C_i < s_{j,m} + M \sum_{k=0}^{k_j-1} E_j(m,k) \\ B_j + \sum_{i \in hnt(j)} \left\lceil \frac{s_{j,m}}{T_i} \right\rceil C_i + \sum_{i \in ht(j)} \left\lceil \frac{f_{j,m}}{T_i} \right\rceil C_i + C_j \leq f_{j,m} \\ \qquad\qquad + M \sum_{k=0}^{k_j-1} E_j(m,k) \\ \sum_{k=0}^{k_j-1} (2^k \times b_{j,k}) \leq m_j - 1 \\ \forall i \in \{i : \pi_i < pi_j \leq \theta_i\}, B_j \geq C_i. \end{cases} \tag{47}$$

Finding task-level point pair redundancy. A point pair (s,f) is task-level redundant if the removal of the associated feasibility region $\mathbb{I}_i(s,f)$ does not change the feasibility region $\mathbb{M}_i$ of τ_i or, alternatively

$$\mathbb{I}_i(s,f) \subseteq \mathbb{M}_i^- \text{where } \mathbb{M}_i^- = \bigcup_{\substack{(s',f') \in \mathcal{I}_i \\ (s',f') \neq (s,f)}} \mathbb{I}_i(s',f').$$

Theorem 24. *A point pair $(s,f) \in \mathcal{I}_i$ is nonredundant at task level if and only if there exists a solution $\mathbf{C}$ such that (s,f) is necessary for τ_i to be feasible at $\mathbf{C}$, i.e.,*

- $\mathbf{C} \in \mathbb{I}_i(s,f)$
- $\forall (s_{i,m}, f_{i,m}) \neq (s,f) \in \mathcal{I}_i, \mathbf{C} \notin \mathbb{I}_i(s_{i,m}, f_{i,m})$.

The first condition can be formulated using the following two constraints (according to the definition of $\mathbb{I}_i(s,f)$).

$$\begin{cases} B_i + \sum_{j \in hp(i)} \left\lceil \frac{s}{T_j} \right\rceil C_j < s \\ B_i + \sum_{j \in hnt(i)} \left\lceil \frac{s}{T_j} \right\rceil C_j + \sum_{j \in ht(i)} \left\lceil \frac{f}{T_j} \right\rceil C_j + C_i \leq f. \end{cases} \tag{48}$$

The second condition is equivalent to $\mathbf{C} \notin \mathbb{I}_i(s_{i,m}, f_{i,m}) \Longleftrightarrow \mathbf{C} \in \bar{\mathbb{I}}_i(s_{i,m}, f_{i,m}) = \bar{\mathbb{S}}_i(s_{i,m}) \bigcup \bar{\mathbb{F}}_i(s_{i,m}, f_{i,m})$. This can be verified by setting up a linear programming problem as follows: A binary variable $b_{i,m}$ is associated with each point pair $(s_{i,m}, f_{i,m}) \neq (s,f) \in \mathcal{I}_i$

$$b_{i,m} = \begin{cases} 1 & \text{if } \mathbf{C} \in \bar{\mathbb{S}}_i(s_{i,m}) \\ 0 & \text{if } \mathbf{C} \in \bar{\mathbb{F}}_i(s_{i,m}, f_{i,m}). \end{cases} \tag{49}$$

Thus, $\forall (s_{i,m}, f_{i,m}) \neq (s,f) \in \mathcal{I}_i$

$$\begin{cases} B_i + \sum_{j \in hp(i)} \left\lceil \frac{s_{i,m}}{T_j} \right\rceil C_j + M(1 - b_{i,m}) \geq s_{i,m} \\ B_i + \sum_{j \in hnt(i)} \left\lceil \frac{s_{i,m}}{T_j} \right\rceil C_j + \sum_{j \in ht(i)} \left\lceil \frac{f_{i,m}}{T_j} \right\rceil C_j + C_i + M b_{i,m} > f_{i,m}. \end{cases} \tag{50}$$

A total of $m_i = |\mathcal{I}_i| - 1$ binary variables are required to encode the infeasibility of these m_i point pairs.[1]

Contrary to the previous case (34), the constraints in (50) define conditions for nonfeasibility and the terms B_i appear on the left-hand side of "$\geq$" constraints with positive coefficient. We use a binary variable $a_{i,j}$ for each task τ_j with $\pi_j < \pi_i \leq \theta_j$: $a_{i,j}$ is defined as 1 if $B_i = C_j$, and 0 otherwise. The blocking time B_i can be formulated as

$$\begin{cases} \forall j \in \{j : \pi_j < \pi_i \leq \theta_j\}, \begin{cases} B_i \geq C_j \\ B_i \leq C_j + M(1 - a_{i,j}) \end{cases} \\ \sum_{j : \pi_j < \pi_i \leq \theta_j} a_{i,j} = 1. \end{cases} \tag{51}$$

Thus, the task-level point pair redundancy can be checked by the feasibility of the following MILP problem:

$$\begin{cases} \mathbf{C} \in \mathbb{I}_i(s,f) : & \text{Formulated as in (48)} \\ \forall (s_{i,m}, f_{i,m}) \neq (s,f) \in \mathcal{I}_i, & \\ \quad \mathbf{C} \notin \mathbb{I}_i(s_{i,m}, f_{i,m}) : & \text{Formulated as in (50)} \\ B_i = \max_{j : \pi_j < \pi_i \leq \theta_j} C_j : & \text{Formulated as in (51).} \end{cases}$$

Finding system-level point pair redundancy. For the point pair (s,f), the feasibility region $\mathbb{I}_i(s,f)$ is system-level redundant if its removal does not change the system feasibility region $\mathbb{M}$. It is equivalent to prove that

$$\mathbb{I}_i(s,f) \bigcap_{j \neq i} \mathbb{M}_j \subseteq \mathbb{M}^- \text{where } \mathbb{M}^- = \bigcup_{\substack{(s',f') \in \mathcal{I}_i \\ (s',f') \neq (s,f)}} \mathbb{I}_i(s',f') \bigcap_{j \neq i} \mathbb{M}_j.$$

Theorem 25. *$(s,f) \in \mathcal{I}_i$ is nonredundant at system level if and only if there exists a solution $\mathbf{C}$ such that*

- *(s,f) is necessary for τ_i to be feasible at $\mathbf{C}$, i.e.,*
  - $\mathbf{C} \in \mathbb{I}_i(s,f)$
  - $\forall (s_{i,m}, f_{i,m}) \neq (s,f) \in \mathcal{I}_i, \mathbf{C} \notin \mathbb{I}_i(s_{i,m}, f_{i,m})$.
- *all other tasks τ_j are schedulable at $\mathbf{C}$.*

The existence of such a solution $\mathbf{C}$ can be checked by the feasibility of the following MILP problem:

$$\begin{cases} \mathbf{C} \in \mathbb{I}_i(s,f) : & \text{Formulated as in (48)} \\ \forall (s_{i,m}, f_{i,m}) \neq (s,f) \in \mathcal{I}_i, & \\ \quad \mathbf{C} \notin \mathbb{I}_i(s_{i,m}, f_{i,m}) : & \text{Formulated as in (50)} \\ B_i = \max_{j : \pi_j < \pi_i \leq \theta_j} C_j : & \text{Formulated as in (51)} \\ \forall j \neq i, \mathbf{C} \in \mathbb{M}_j : & \text{Formulated as in (47).} \end{cases}$$

Finding system-level task redundancy. Since $\mathbb{M} = \bigcap_{\tau_j \in \Gamma} \mathbb{M}_j$, the region $\mathbb{M}_i$ does not contribute to the definition of $\mathbb{M}$ if and only if the feasibility region of the other tasks is a subset of $\mathbb{M}_i$.

Theorem 26. *τ_i is nonredundant at the system level if and only if there exists a set of execution times $\mathbf{C}$ such that*

- *τ_i is not schedulable at $\mathbf{C}$.*
- *all other tasks τ_j are schedulable at $\mathbf{C}$.*

The existence of such a solution $\mathbf{C}$ can be checked by the feasibility of the following MILP problem:

$$\begin{cases} \forall (s,f) \in \mathcal{I}_i, \mathbf{C} \notin \mathbb{I}_i(s,f) : & \text{Formulated as in (50)} \\ B_i = \max_{j : \pi_j < \pi_i \leq \theta_j} C_j : & \text{Formulated as in (51)} \\ \forall j \neq i, \mathbf{C} \in \mathbb{M}_j : & \text{Formulated as in (47).} \end{cases}$$

9 Experimental Results

We performed experiments to assess the effectiveness and efficiency of our algorithm in reducing the set of points. First, we discuss the case of fully preemptive systems by applying our method to randomly generated task sets, then to an automotive case study. Next, we provide experimental results for the extension to task sets with blocking times and with preemption thresholds.

9.1 Random Task Sets, Preemptive Case

We generate applications consisting of random sets with $n = 5$ to 33 tasks. The systems are assumed to be fully preemptive. One thousand sets are generated and then examined for schedulability for each n. The periods of the tasks are generated by the product of one to three factors, each randomly drawn from three harmonic sets (2, 4), (6, 12), (5,10). This generates periods belonging to at most three harmonic sets, as is the case for most applications of practical interest. In one set of experiments, deadlines are assumed equal to periods. In another set, deadlines are randomly selected between $T/2$ and T for each task. Priorities are assigned to tasks according to the deadline-monotonic policy, since it has the largest feasibility region among all the fixed priority scheduling policies, and it is likely to require the largest number of points. We compare the number of points to be checked for the lowest priority task and for the entire set.

We start from the set $\mathcal{I}_i = \mathcal{S}_i$ in (8). Of course, if we start from the set $\mathcal{I}_i = \mathcal{P}_i$, the number of points at each intermediate step of redundancy reduction is smaller and the runtime of Algorithm 1 is shorter. The resulting nonredundant set is the same regardless of the starting set, or the ordering of the tasks τ_i or points $p \in \mathcal{I}_i$ in the algorithm.

In the case of the 1,000 runs on 10-task systems with deadline equal to period, these are the results. For the lowest priority task, only one point needed to be checked in 14.3 percent of the cases, $\leq$2 in 24.8 percent of the cases, $\leq$4 points in 48.3 percent of the cases, and $\leq$8 points in 80.8 percent of the cases. As a comparison, when using $\mathcal{P}_i$ [2], no more than four and eight points provide an exact test in 29.2 and 49.6 percent of the cases, respectively. For the schedulability of the whole task set, the total number of points are compared in Fig. 4. On average, the number of points in the nonredundant set is 54 percent less than what is required by the method in [2].

When considering sets from 5 to 33 tasks, Fig. 5 shows the amount of redundancy detected by our method with respect to the sets $\mathcal{P}_n$ and $\mathcal{P}$ of [2] for the lowest priority task (dark line) and the entire task set (light line), respectively. The nonredundant sets computed by Algorithm 1 are $\mathcal{I}_n$ and $\mathcal{I}$. For sets of more than 20 tasks, we limited the running time of the solver for each MILP problem to 60 seconds, indicated

1. It can be improved by grouping together the point pairs with the same start time.

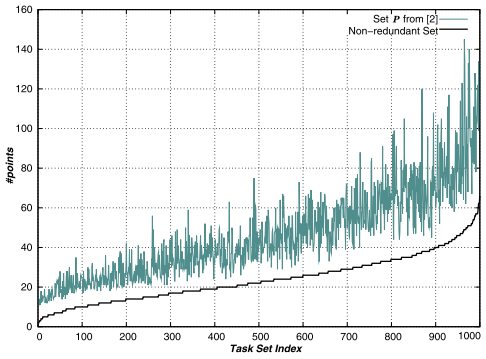


Fig. 4. Number of points for checking feasibility of all tasks (10-tasks sets).

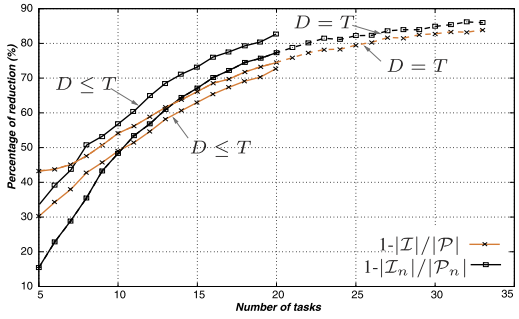


Fig. 5. Percentage of redundancy found in $\mathcal{P}_n$ and $\mathcal{P}$.

by the dashed lines in Fig. 5. In general, the percentage of detected redundancy increases with the number of tasks. For the first set of experiments where $D = T$, the percentage of redundancy in $\mathcal{P}$ increases from 43.24 percent for $n = 5$ to 74.49 percent for $n = 20$. Despite the runtime limitation, up to 84 percent redundancy is found for sets with 33 tasks. The second set of experiments, with $D \leq T$, has similar percentage of reduction to $\mathcal{P}_n$ and $\mathcal{P}$. In all cases, $\mathcal{S}_n$ is found to contain more than 99 percent redundant points and $\mathcal{S}$ is more than 97 percent redundant.

9.2 Algorithm Complexity

Our algorithm consists of a first stage, in which task-level redundancy is removed, and a second stage, for removing system-level redundancy. The first stage is based on the checking of a set of simple inequalities (Theorem 3) or the solution of linear programming problems (Theorem 5). This can be checked in a very short time (linear-programming problems are solvable in polynomial time). The second stage requires solving a very large number of problems in (24) and (25). To check the feasibility of (24) and (25) for task τ_i, it requires an exact formulation of the feasibility region of other tasks τ_j. Thus, having a first stage which reduces many of the redundant points allows a compact formulation of the feasibility region in the second stage. These problems are solved using CPLEX 11.0 on a machine with an Intel Pentium 3.2 GHz CPU and 1 GB memory.

The runtime information for the experiment where $D = T$ is shown in Fig. 6, where the light/orange line shows the total runtime of Algorithm 1 for the 1,000 samples, and the black line denotes the maximum runtime among them. The runtime grows faster than exponentially with respect to the number of tasks. However, most of the time is spent on a few cases where solving (24) or (25) is time consuming. We performed another set of experiments by setting a 60-second

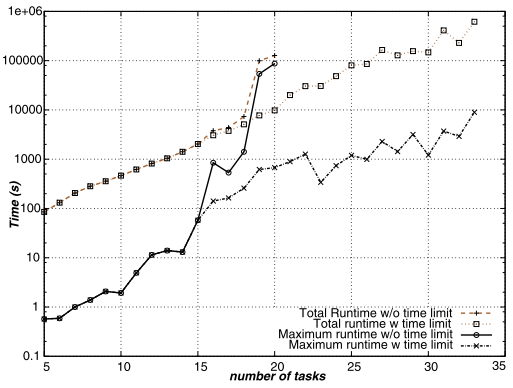


Fig. 6. Runtime to find the test set with/without timeout.

time limit to the MILP solver. For each instance of (24) or (25), if the solver fails to find a feasible solution or prove infeasibility in 60 seconds, we leave the points in the test set (a safe, conservative assumption). The resulting runtimes are shown for sets up to 33 tasks with dashed lines in Fig. 6. The time limit to the solver may result in some residual redundancy in the test set $\mathcal{I}'$. However, the difference is very small: up to $n = 15$, $\mathcal{I}'$ and $\mathcal{I}$ are exactly the same; for $n = 16$ to 20, $\mathcal{I}'$ contains 3, 21, 31, 30, and 38 more points, respectively. The maximum percentage of residual redundancy in $\mathcal{I}'$ is 0.04 percent with $n = 18$. Thus, enforcing a time limit to the MILP solver allows avoiding long runtimes at the price of a very small redundancy.

Also, to prove that a point is system-level redundant we need to prove that (24) is infeasible. This is in general much harder than to find a feasible $\mathbf{C}$. For $n = 16, 17, 18, 19, 20$, the average runtimes for the cases where (24) is infeasible are 490, 509, 1,021, 20,618, and 21,615 ms, respectively. Even if we do not count the outliers (runtime ≥ 60 seconds), the average runtime is 246, 321, 483, 834, 1,030 ms, respectively. If (24) is feasible, the average runtime is much shorter: 15, 19, 23, 41, 67 ms, respectively.

9.3 An Automotive Case Study

We applied our algorithm to a case study consisting in the optimization of multitask implementations of a Simulink models with 90 function blocks executed at seven different periods [13]. Functional blocks must be mapped for execution into tasks and priorities must be assigned to tasks so that all blocks meet their deadlines. Code generation tools enforce the addition of extra buffering and delays whenever there is a rate transition among functional blocks with a possible preemption. The optimization objective is to minimize additional buffers and delays by preventing preemptions between communicating blocks with different rates. The overall system utilization is 94.1 percent. The function blocks communicate through 106 links, 37 from high rate to low rate blocks which could require a rate transition block with an additional memory buffer, 31 from low to high rate blocks possibly requiring a rate transition block with two additional memory buffers and an additional functional delay. Since task priority assignment is among the optimization variables, Algorithm 2 is used to find the test set for the exact definition of the feasibility region.

The computation time of our reduced set of points takes only *3.7 seconds*. Using Algorithm 2, the total number of points that are required to define the feasibility region for this problem is 113, and the number of additional binary

variables to encode the disjunctive constraints is zero. Indeed, knowledge on the worst case execution times in (24) allows to check the feasibility of each task using only one constraint. Under the hypothesis of DM assignment, the method in [2] would require 561 points and 290 additional binary variables for the definition of the feasibility region. We then performed a comparison of the optimization runtimes using the set in [2] and our set. When using the set from [2], *the solver cannot find a feasible solution in more than 2 hours*. Using our reduced set, *the optimal solution can be found in 6.8 seconds [14]. Hence, the 3.7 seconds spent in removing redundancy in the set of points are worth hours of saved optimization time.*

To better show the effectiveness and the time advantage of the proposed method over conventional search techniques, we also implement a branch-and-bound method to solve the problem. The problem essentially is to assign priority orders to 90 function blocks; thus, there are $90!(\approx 1.5 \times 10^{138})$ different solutions. By preassigning the priority orders according to simple schedulability conditions and precedence constraints as in [14], there are about 2.1×10^{50} solutions remaining. This is too large a space to explore with simple branch-and-bound method, even with the most powerful computer today regardless how fast the feasibility analysis is at each step. Therefore, we define a subset of the original case study consisting of 18 functional blocks and 17 communication links. To compensate for the reduction in the load, the utilization is scaled up to 95.7 percent. Among the $18!(\approx 2.8 \times 10^{16})$ possible priority assignments, 8.5×10^6 remain after the preprocessing stage where obvious nonfeasible solutions are pruned and some priority orders are preassigned [14]. The schedulability analysis routine is then called for 2.0×10^7 times. The total runtime is 26.7 seconds for the branch-and-bound optimization using the set of points from Algorithm 2, and 29.6 seconds using the set of points from [2]. The runtime is 38.1 seconds with response time-based analysis. As a comparison, the same problem can be solved in less than 0.1 second using CPLEX. For larger (realistic) systems, branch and bound with analysis at each step may simply be inapplicable. A powerful solver like CPLEX with many modern optimization techniques can solve the problem much quicker, as in the full version of the case study. It also shows that a small improvement on the feasibility analysis can result in a huge difference in the runtime of the solver.

9.4 Scheduling with Blocking Times

In this section, we present the results on task sets with blocking times. The periods of the task sets are generated similarly to the ones in Section 9.1; the deadlines are assumed to be equal to the periods. We examine two cases. In the first, blocking times are small, with each critical section taking at most 5 percent of the period ($B \leq 5\%T$). In the second set, blocking times are larger (up to $40\%T$).

Fig. 7 shows the amount of redundancy detected by our method with respect to the sets $\mathcal{P}_n$ and $\mathcal{P}$ of [2] for the lowest priority task as well as the entire task set. For the first set of experiments ($B \leq 5\%T$), the percentage of redundancy in $\mathcal{P}$ increases from 14.79 percent for $n = 5$ to 65.76 percent for $n = 20$. The second set of experiments, with $B \leq 40\%T$, results in an even higher percentage of redundancy, from 23.23 percent for $n = 5$ to 72.26 percent for $n = 20$.

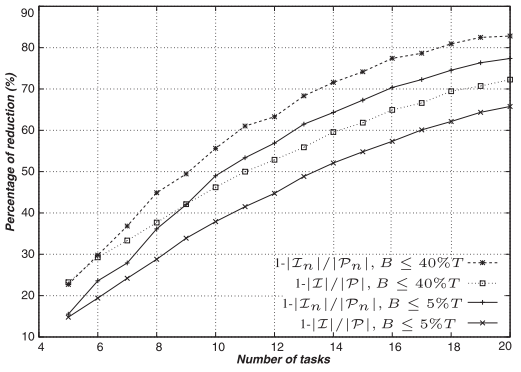


Fig. 7. Percentage of redundancy found in $\mathcal{P}_n$ and $\mathcal{P}$ for task sets with blocking times.

9.5 Preemption Threshold Scheduling Model

In this section, we present the results on systems scheduled with a preemption threshold model. In Section 8, we argue that an exact characterization of the feasibility region will be exceedingly complex, and we define lower and upper bounds to the region. Here, we attempt a quantification of the difference between the exact region and its upper and lower bounds, which corresponds to necessary-only (optimistic) and sufficient-only (pessimistic) conditions for feasibility. Also, for the necessary-only condition, we perform experiments to assess the effectiveness of our algorithm in finding the set of point pairs and to estimate the number of points that are needed for the definition of the system feasibility region. The sufficient-only condition is based on the finish time only, and the effectiveness and efficiency of the algorithm are similar to fully preemptive scheduling.

We generate applications consisting of random sets with $n = 5$ to 15 tasks. One thousand task configurations are tried for each n. Priorities are assigned to tasks according to the deadline-monotonic policy, and preemption thresholds are uniformly distributed between the task priority and the highest priority level in the system. The task periods are randomly generated in the same way as in Section 9.1. Two deadline settings are considered, $D = T$, and D randomly distributed between $T/2$ and T. All problems are solved using CPLEX 12.1 on a machine with an Intel Xeon 3.0 GHz CPU and 2 GB memory.

To quantify the difference between the exact feasibility region and its approximation, for each of the 1,000 task configuration, a number of task utilizations have been generated by selecting execution times such that the system utilization is uniformly distributed between 0 and 100 percent (using the *UUniFast* algorithm in [3]). We count the numbers N^{suf}, N^{exact}, and N^{nec} that are feasible by the sufficient, exact, and necessary tests respectively, until N^{exact} reached 10^5. The difference between the exact and sufficient (necessary) conditions as an indication of the amount of pessimism (optimism) can be quantified by ρ^{suf} (ρ^{nec}) as below

$$\rho^{\text{suf}} = \frac{N^{\text{exact}} - N^{\text{suf}}}{N^{\text{exact}}} \times 100\%, \ \rho^{\text{nec}} = \frac{N^{\text{nec}} - N^{\text{exact}}}{N^{\text{exact}}} \times 100\%.$$

Fig. 8 plots the number of feasible utilization sets for the necessary-only and sufficient-only tests in the case of 1,000 task configurations of a 10-task system with $D = T$. The difference between the sufficient and exact conditions is significant: of the 1,000 runs, 937 have $N^{\text{suf}} < 10^5$, with a minimum of 45,077, and an average of 88,034. However, the

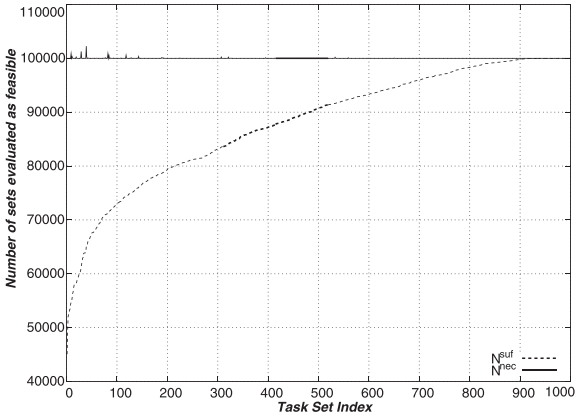


Fig. 8. Number of task sets feasible for necessary and sufficient conditions (10-tasks sets).

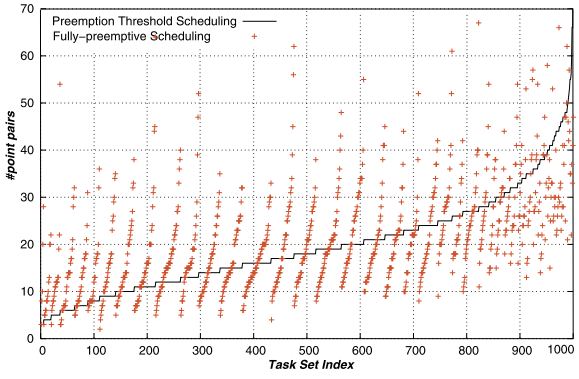


Fig. 9. Number of points for checking feasibility of all tasks with preemption thresholds (10-tasks).

necessary-only test fares much better. The number of feasible task sets for the necessary-only test is very close to the ideal line at 100,000. Only in 109 out of 1,000 task configurations, the optimistic test provides false results for at least one task set, i.e., $N^{\text{nec}} > 10^5$. In 59 of these configurations, there are less than 10 false positives, and only in two configurations, the test fails for more than 1,000 utilization sets ($N^{\text{nec}} - N^{\text{exact}} > 1{,}000$).

The small difference between N^{nec} and N^{exact} is also confirmed for task configurations with $n \neq 10$. We tried n from 5 to 15 for a total of 16,000 configurations. The average value ρ^{nec} is lower than 0.021 percent, and the maximum ρ^{nec} is 6.1 percent. The sufficient condition provides a much easier formulation, but its use may easily result in a suboptimal solution. The necessary condition is definitely more complex, but the small difference between the necessary and exact feasibility regions indicate a very good chance that the solution obtained using the necessary-only condition is also feasible with respect to the exact test.

To evaluate the effectiveness of the algorithm, Fig. 9 shows the number of point pairs that are needed to evaluate the sufficient-only condition for 1,000 configurations of 10 tasks with $D = T$. At most eight point pairs need to be checked in 10.9 percent of the cases, $\leq$16 in 43.2 percent of the cases, $\leq$32 points in 89.5 percent of the cases, and $\leq$64 points in 99.8 percent of the cases. Fig. 9 also plots the number of points that are required for the same task set assuming fully preemptive scheduling. Although the number of initial points for the preemption threshold scheduling is approximately the square of those for the

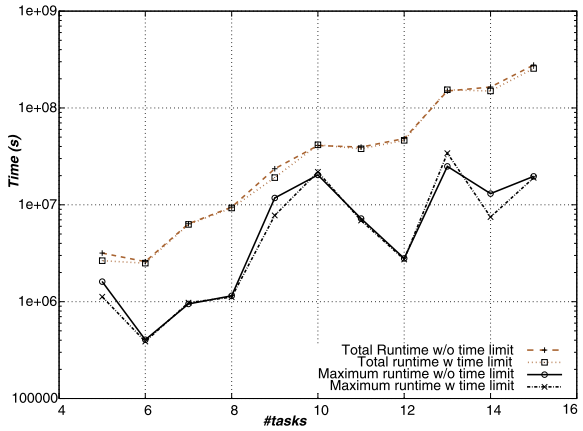


Fig. 10. Runtime of the algorithm (with and without the time limit) to find the nonredundant set.

fully preemptive case, their average numbers of nonredundant points are roughly the same. In 452 task sets, fully preemptive scheduling requires more points than preemption threshold scheduling, and in 492 cases it requires less. The average redundancy with respect to the set $\mathcal{I}_n$ in Theorem 19 is always very high (>99.99%) for both $D = T$ and $D \leq T$.

The runtime of the algorithm for the case $D = T$ is shown in Fig. 10, where the light line shows the total runtime for the 1,000 configurations, and the dark line is the maximum runtime among them. The runtime grows more than exponentially with respect to n. Similar to the preemptive case, we tried to limit the runtime of the solver for each MILP problem to 60 seconds, trading a possible redundancy for a shorter runtime. We were only able to process sets with $n \leq 15$. The difficulty does not originate from the MILP problems (the time saved by the timeout is very small), but from the huge number of initial point pairs (quadratically more than the fully preemptive case). For $n = 9, 10, 12, 13, 14, 15$, the set $\mathcal{I}'$ obtained using a timeout in the solver contains 52, 57, 1, 361, 28, and 143 more points than the nonredundant set, respectively. For all the other n, $\mathcal{I}'$ and $\mathcal{I}$ are identical. The maximum percentage of residual redundancy in $\mathcal{I}'$ is 1.0 percent when $n = 13$.

10 Conclusions

In the design of time-critical applications, schedulability analysis results are used to define the task feasibility region so that optimization techniques can be used to compute the best design that satisfies the deadlines. In this paper, we summarize possible approaches to the problem and provide an improved feasibility analysis that opens the possibility for an exact definition of the feasibility region which requires an acceptable number of binary variables. We demonstrate the efficiency and effectiveness of our algorithm by random task sets and industrial-size problems.

Acknowledgments

This work is supported by NSERC Discovery Grant RGPIN 418741-12 and NSFC Project Grant #61070002.

REFERENCES

[1] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings, "Applying New Scheduling Theory to Static Priority Pre-Emptive Scheduling," *Software Eng. J.*, vol. 8, no. 5, pp. 284-292, Sept. 1993.

[2] E. Bini and G.C. Buttazzo, "Schedulability Analysis of Periodic Fixed Priority Systems," *IEEE Trans. Computers*, vol. 53, no. 11, pp. 1462-1473, Nov. 2004.

[3] E. Bini and G.C. Buttazzo, "Measuring the Performance of Schedulability Tests," *Real-Time Systems*, vol. 30, nos. 1/2, pp. 129-154, May 2005.

[4] E. Bini, T.H. Châu Nguyen, P. Richard, and S.K. Baruah, "A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines," *IEEE Trans. Computers*, vol. 58, no. 2, pp. 279-286, Feb. 2009.

[5] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.E. Arzen, "How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime," *IEEE Control Systems Magazine*, vol. 23, no. 3, pp. 16-30, June 2003.

[6] A. Davare, Q. Zhu, M.D. Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period Optimization for Hard Real-Time Distributed Automotive Systems," *Proc. 44th Design Automation Conf.*, pp. 278-283, 2007.

[7] L. George, N. Riviere, and M. Spuri, "Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling," technical report, INRIA, 1996.

[8] R. Ghattas and A.C. Dean, "Preemption Threshold Scheduling: Stack Optimality, Enhancements and Analysis," *Proc. 13th Real-Time and Embedded Technology and Applications Symp.*, pp. 147-157, 2007.

[9] B. Grünbaum, *Convex Polytopes*, second ed., p. 52a. Springer, 2003.

[10] A. Hamann, R. Racu, and R. Ernst, "Multi-Dimensional Robustness Optimization in Heterogeneous Distributed Embedded Systems," *Proc. 13th Real-Time and Embedded Technology and Applications Symp.*, pp. 269-280, 2007.

[11] J.P. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proc. 10th Real-Time Systems Symp.*, pp. 166-171, 1989.

[12] M. Di Natale, "Optimizing the Multitask Implementation of Multirate Simulink Models," *Proc. 12th Real-Time and Embedded Technology and Applications Symp.*, pp. 335-346, 2006.

[13] M. Di Natale and V. Pappalardo, "Buffer Optimization in Multitask Implementations of Simulink Models," *ACM Trans. Embedded Computing Systems*, vol. 7, no. 3, pp. 1-32, May 2008.

[14] M. Di Natale, L. Guo, H. Zeng, and A. Sangiovanni-Vincentelli, "Synthesis of Multi-Task Implementations of Simulink Models with Minimum Delays," *IEEE Trans. Industrial Informatics*, vol. 6, no. 4, pp. 637-651, Nov. 2010.

[15] A. Metzner and C. Herde, "RTSAT—An Optimal and Efficient Approach to the Task Allocation Problem in Distributed Architectures," *Proc. 27th Real-Time Systems Symp.*, pp. 147-158, 2006.

[16] A.K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," PhD thesis, MIT, 1983.

[17] M. Oral and O. Kettani, "Linearization Procedure for Quadratic and Cubic Mixed-Integer Problems," *Operations Research*, vol. 40, no. S1, pp. 109-116, Jan./Feb. 1992.

[18] "OSEK/VDX Operating System Specification Version 2.2.3," http://www.osek-vdx.org, Feb. 2005.

[19] J. Regehr, "Scheduling Tasks with Mixed Preemption Relations for Robustness to Timing Faults," *Proc. 23rd Real-Time Systems Symp.*, pp. 315-326, 2002.

[20] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. Computers*, vol. 39, no. 9, pp. 1175-1185, Sept. 1990.

[21] M. Sjödin and H. Hansson, "Improved Response Time Analysis Calculations," *Proc. 19th Real-Time Systems Symp.*, pp. 399-408, 1998.

[22] W. Lamie, "Preemption-Threshold," White Paper, Express Logic, Inc., http://rtos.com/articles/18833, 2012.

[23] M. Velasco, P. Marti, and E. Bini, "Control-Driven Tasks: Modeling and Analysis," *Proc. 29th Real-Time Systems Symp.*, pp. 280-290, 2008.

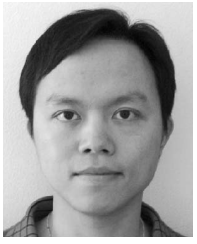
[24] Y. Wang and M. Saksena, "Scheduling Fixed-Priority Tasks with Preemption Threshold," *Proc. Sixth Real-Time Computing Systems and Applications*, pp. 328-335, 1999.

[25] H. Zeng and M.D. Natale, "An Efficient Formulation of the Real-Time Feasibility Region for Design Optimization," TECIP technical report, Scuola Superiore S. Anna, Pisa, Aug. 2011.

[26] W. Zheng, M.D. Natale, C. Pinello, P. Giusto, and A. Sangiovanni-Vincentelli, "Synthesis of Task and Message Activation Models in Real-Time Distributed Automotive Systems," *Proc. Conf. Design, Automation & Test in Europe*, pp. 93-98, 2007.

[27] W. Zheng, Q. Zhu, M. Di Natale, and A. Sangiovanni-Vincentelli, "Definition of Task Allocation and Priority Assignment in Hard Real-Time Distributed System," *Proc. Real-Time System Symp.*, pp. 161-170, 2007.

[28] Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, and A. Sangiovanni-Vincentelli, "Optimizing Extensibility in Hard Real-Time Distributed Systems," *Proc. 15th Real-Time and Embedded Technology and Applications Symp.*, pp. 275-284, 2009.



Haibo Zeng received the BE and ME degrees in electrical engineering from Tsinghua University, Beijing, China. He received the PhD degree in electrical engineering and computer sciences from University of California at Berkeley. He is currently an assistant professor at McGill University, Canada. He was a senior researcher at General Motors R&D until October 2011. His research interests are design methodology, analysis, and optimization for embedded systems, real-time systems, and cyber-physical systems. He is a member of the IEEE.

Marco Di Natale received the PhD degree from Scuola Superiore Sant'Anna in 1991 and has been visiting researcher at the University of California, Berkeley, in 2006 and 2008/2009. He is currently an associate professor at the Scuola Superiore Sant'Anna of Pisa, Italy, in which he held a position as director of the Real-Time Systems (ReTiS) Laboratory from 2003 to 2006. He has been selected in 2006 by the Italian Ministry of Research as the national representative in the mirror group of the ARTEMIS European Union Technology platform. He has been a researcher in the area of real-time systems and embedded systems for more than 15 years, being author or coauthor of more than 100 scientific papers. He has been winner of four best paper awards and the Archie T. Colwell award. He has served as program committee member and has been an organizer of tutorials and special sessions for the main conferences in the area, including the Real-time Systems Symposium, the IEEE/ACM Design Automation Conference (DAC), the Design Automation and Test in Europe (DATE) and the Real-Time Application Symposium. He also served as program chair and track chair for the RTAS conference, Automotive and Transportation track chair of the 2010 DATE Conference and program chair for the ICESS conference. He has been an associate editor for the *IEEE Transactions on CAD* and the *IEEE Embedded Systems Letters* and is currently in the editorial board of the *IEEE Transactions on Industrial Informatics*. He is currently a senior member of the IEEE.