The Concept of Unschedulability Core for Optimizing Priority Assignment in Real-Time Systems

***Abstract*—In the design optimization of real-time systems, the schedulability analysis is used to define the feasibility region within which tasks meet their deadlines, so that optimization algorithms can find the best solution within the region. However, the complexity of current schedulability analysis techniques often makes it difficult to leverage existing optimization frameworks and scale to large designs. In this paper, we consider the design optimization problems for real-time systems scheduled with fixed priority, where task priority assignment is part of the decision variables. We propose the concept of unschedulability core, a compact representation of the schedulability conditions, and develop efficient algorithms for its calculation. We present a new optimization procedure based on lazy constraint paradigm that leverages such a concept. Experimental results on two case studies show that the new optimization procedure provides optimal solutions, but is a few magnitudes faster than other exact algorithms (Branch-and-Bound, Integer Linear Programming).**

I. Introduction

The design of real-time embedded systems is often subject to many requirements and objectives in addition to real-time constraints, including limited resources (e.g., memory), cost, quality of control, and energy consumption. For example, automotive industry is hard pressed to deliver products with low cost, due to the large volume and the competitive international market [1]. Similarly, the technology innovation for medical devices is mainly driven by reduced size, weight, and power (SWaP) [2]. In these application domains, it is important to perform *design optimization* in order to find the best design (i.e., optimized according to an objective function) while satisfying all the critical requirements.

Formally, a design optimization problem is defined by decision variables, constraints, and an objective function. The *decision variables* represent the set of design choices under the designers' control. The set of *constraints* forms the feasibility region, the domain of the allowed values for the decision variables. The *objective function* characterizes the optimization goal. In general, the optimal design can be obtained by solving an optimization problem where the objective function is optimized within the feasibility region. For real-time systems, the *feasibility region* (also called *schedulability region* if concerning only real-time schedulability) must only contain the designs that satisfy the *schedulability constraints* whereby tasks complete before their deadlines.

In this paper, we consider the design optimization for real-time systems scheduled with fixed priority, where priority assignment is part of the decision variables. There is a large body of work on priority assignment for real-time systems with fixed priority scheduling. In particular, Audsley's algorithm [3] is proven to be "optimal" for many task models and scheduling schemes, if the designer is only concerned to find a schedulable solution. See a recent survey by Davis et al. [4] on a complete list of applicable settings. However, if the design optimization problem contains constraints or an objective function related to other metrics (such as memory, power, thermal, etc.), Audsley's algorithm is no longer guaranteed to be optimal. In fact, such problems typically are NP-hard, including the two case studies in this paper: the optimization of Simulink models with Adaptive Mixed Criticality scheduling (Section V-A), and the memory minimization in the implementation of AUTOSAR models (Section V-B).

Related Work. In general, the current approaches for optimizing priority assignment in complex design optimization problems (i.e., those without known polynomial-time optimal algorithms) can be classified into three categories. The *first* is based on meta heuristics such as simulated annealing (e.g., [5], [6]) and genetic algorithm (e.g., [7]). The *second* is to develop problem specific heuristics (e.g., [8], [9]). These two categories do not have any guarantee on optimality.

The *third* category is to search for the exact optimum, often applying existing optimization frameworks such as branch-and-bound (BnB) (e.g., [10]), or integer linear programming (ILP) (e.g., [11]). However, this approach typically suffers from scalability issues and may have difficulty to handle large industrial designs. For example, automotive engine control system contains over a hundred runnables [12], but the ILP based approach can only scale up to about 40 runnables (see Section V). Furthermore, not all problems can easily be formulated in a particular framework due to the complexity of schedulability conditions. For example, the exact schedulability analysis for tasks with non-preemptive scheduling requires to check all the task instances in the busy period, but the number of instances is unknown a priori. Hence, it is difficult to formulate the exact schedulability constraints in ILP [13].

Our Contributions. In this paper, instead of directly reusing the standard techniques (BnB, ILP, etc.), we aim at developing techniques for optimizing priority assignment that can guarantee the optimality of the solution while drastically improving the scalability. The observation is that schedulability conditions are often inefficient or even impossible to be directly formulated in these generic optimization frameworks. We develop a set of new techniques and make the following contributions:

- We propose the concept of **unschedulability core**, an abstraction of the schedulability condition in real-time systems scheduled with fixed priority. It can be represented by a set of new and compact constraints to be learned efficiently during the execution of the optimization procedure (i.e., at runtime).
- We devise an optimization procedure based on **lazy constraint paradigm** that judiciously utilizes the unschedulability cores to drastically improve the scalability.
- We use two design optimization problems to illustrate the benefit of the proposed approach. The new unschedulability core guided optimization algorithm runs one or more magnitudes faster than other optimal algorithms (BnB, ILP) while maintaining the optimality of the solutions.

The rest of the paper is organized as follows. Section II describes the task models that are suitable for the proposed

approach. Section III proposes the concept of unschedulability core and studies its efficient calculation. Section IV presents the optimization procedure that leverages the unschedulability cores for optimizing priority assignment. Section V demonstrates the effectiveness of the proposed approach with two example problems. Finally, Section VI concludes the paper.

II. Preliminary

We consider a real-time system scheduled by fixed priority. It consists of a set of periodic or sporadic tasks $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$. Each task τ_i is assumed to have a *unique* priority π_i (the higher the number, the higher the priority) to be assigned by the designer. The concept of unschedulability core applies to any systems scheduled with fixed priority. However, its application in design optimization is most effective when there is a simple algorithm to determine the existence of a schedulable priority assignment for a given task set. Hence, we consider a list of task models and scheduling schemes where some simple priority assignment policy (such as Rate Monotonic or Deadline Monotonic) or Audsley's algorithm [3] is applicable (i.e., it can find a schedulable priority assignment if there exists one). The list, as summarized in [4], includes:

- The periodic task model, where independent tasks are scheduled on a single-core platform with preemptive scheduling. Each task is characterized by a tuple of parameters: T_i denotes the period; $D_i = T_i$ represents the implicit relative deadline; C_i denotes the worst case execution time (WCET).
- Sporadic tasks with arbitrary deadlines, or static offsets.
- Probabilistic real-time systems where task WCETs are described by independent random variables [14].
- Systems scheduled with deferred preemption [15].
- Tasks modeled as arbitrary digraphs [16], where vertices represent different kinds of jobs, and edges represent the possible flows of control.
- Tasks access shared resources that may be protected by semaphore locks to ensure mutual exclusion.

The set of binary variables denoting task priority assignment is defined as $\mathbf{P} = \{p_{i,j} | i \neq j, \tau_i \in \Gamma, \tau_j \in \Gamma\}$, where

$$p_{i,j} = \begin{cases} 1 & \pi_i > \pi_j, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The priority assignment shall satisfy the antisymmetric and transitive properties: If τ_i has a higher priority than τ_j ($p_{i,j} = 1$), then τ_j has a lower priority than τ_i ($p_{j,i} = 0$); If τ_i has a higher priority than τ_j ($p_{i,j} = 1$) and τ_j has a higher priority than τ_k ($p_{j,k} = 1$), then τ_i must have a higher priority than τ_k ($p_{i,k} = 1$). These properties can be formally formulated as

$$\begin{cases} \text{Antisymmetry:} & p_{i,j} + p_{j,i} = 1, & \forall i \neq j \\ \text{Transitivity:} & p_{i,j} + p_{j,k} \leq 1 + p_{i,k}, & \forall i \neq j \neq k \end{cases} \tag{2}$$

We focus on a design optimization problem where the decision variables $\mathbf{X}$ include the task priority assignment, i.e., $\mathbf{P} \subseteq \mathbf{X}$.

$$\begin{aligned} & \min C(\mathbf{X}) \\ s.t. \quad & \mathbf{F}(\mathbf{X}) \leq 0 \end{aligned} \tag{3}$$

Here $C(\mathbf{X})$ is the objective function to be minimized, $\mathbf{F}(\mathbf{X}) \leq 0$ defines the set of constraints that the solutions in the feasibility region shall satisfy, including those in Equation (2).

III. The Concept of Unschedulability Core

Central to our technique is the concept of unschedulability core. Intuitively, it is an irreducible representation of the priority assignment that causes the system unschedulable. In this section, we establish its formal definition, and study its properties and efficient calculation. We use a running example system Γ_e configured as in Table I to illustrate, where all tasks are assumed to be *independent and preemptive*.

TABLE I: An Example Task System Γ_e

τ_i	T_i	D_i	C_i	τ_i	T_i	D_i	C_i
τ_1	10	10	2	τ_2	20	20	3
τ_3	40	40	16	τ_4	100	100	3
τ_5	200	200	17	τ_6	400	400	32

Definition 1. A **partial priority order (PPO)**, denoted as $r_{i,j} \equiv (p_{i,j} = 1)$, defines a priority order that τ_i has a higher priority than τ_j. A **PPO set** $\mathcal{R} = \{r_{i_1,j_1}, r_{i_2,j_2}, ..., r_{i_m,j_m}\}$ is a collection of one or more partial priority orders that are *consistent with the properties in Equation (2)*. The number of elements in $\mathcal{R}$ is defined as its **cardinality**, denoted as $|\mathcal{R}|$.

Definition 2. Let Γ be a task system and $\mathcal{R}$ be a PPO set on Γ. Γ is **$\mathcal{R}$-schedulable** if and only if there exists a feasible priority assignment $\mathcal{P}$ that respects the partial priority order corresponding to each element in $\mathcal{R}$.

Example 1. Consider the example task system Γ_e in Table I and two PPO sets $\mathcal{R}_1 = \{r_{1,2}, r_{2,3}\}$, $\mathcal{R}_2 = \{r_{5,4}, r_{4,3}\}$. Γ_e is $\mathcal{R}_1$-schedulable, since the system is schedulable under rate-monotonic priority assignment which respects $\mathcal{R}_1$. However, Γ_e is not $\mathcal{R}_2$-schedulable: τ_1 must have a higher priority than τ_3 (due to $C_3 > D_1$), hence assigning τ_4 and τ_5 with higher priority than τ_3 will result in deadline miss for τ_3.

The following theorem intuitively states that if the system is schedulable for a PPO set, then the system is also schedulable for any of its subset.

Theorem 1. Let $\mathcal{R}$ and $\mathcal{R}'$ be two PPO sets on Γ such that $\mathcal{R}' \subseteq \mathcal{R}$. The following always holds

$$\Gamma \text{ is } \mathcal{R}\text{-schedulable} \Rightarrow \Gamma \text{ is } \mathcal{R}'\text{-schedulable} \tag{4}$$

The proof is straightforward as any priority assignment satisfying $\mathcal{R}$ must also satisfy $\mathcal{R}'$. Applying contrapositive law on Theorem 1, we have

$$\Gamma \text{ is not } \mathcal{R}'\text{-schedulable} \Rightarrow \Gamma \text{ is not } \mathcal{R}\text{-schedulable} \tag{5}$$

We now give the definition of unschedulability core. Intuitively, it is an irreducible representation of the reason why the system is unschedulable, in the sense that removing any element from it will allow schedulable priority assignment.

Definition 3. Let Γ be a task system and $\mathcal{R}$ be a PPO set on Γ. $\mathcal{R}$ is an **unschedulability core** for Γ if and only if $\mathcal{R}$ satisfies the following two conditions:

- Γ is not $\mathcal{R}$-schedulable;
- $\forall \mathcal{R}' \subset \mathcal{R}$, Γ is $\mathcal{R}'$-schedulable.

Remark 1. By Theorem 1, the second condition in Definition 3 can be replaced by

- $\forall \mathcal{R}' \subset \mathcal{R}$ s.t. $|\mathcal{R}'| = |\mathcal{R}| - 1$, Γ is $\mathcal{R}'$-schedulable.

Example 2. Consider the PPO set $\mathcal{R}_3 = \{r_{5,4}, r_{4,3}, r_{3,6}\}$, which equivalently defines the priority order $\pi_5 > \pi_4 > \pi_3 > \pi_6$. Obviously, Γ_e is not $\mathcal{R}_3$-schedulable as it is not schedulable for the subset $\mathcal{R}_2 = \{r_{5,4}, r_{4,3}\}$ of $\mathcal{R}_3$ (see Example 1). However, $\mathcal{R}_3$ is not an unschedulability core since it has a proper subset $\mathcal{R}_2$ for which Γ_e is not schedulable.

$\mathcal{R}_2$ is a valid unschedulability core, as for each of its proper subset, there exists a respecting feasible priority assignment: $\mathcal{P} = [\pi_1 > \pi_2 > \pi_3 > \pi_5 > \pi_4 > \pi_6]$ respects $\mathcal{R}_2^{(1)} = \{r_{5,4}\}$ and $\mathcal{R}_2^{(2)} = \emptyset$, and $\mathcal{P}' = [\pi_1 > \pi_2 > \pi_4 > \pi_3 > \pi_5 > \pi_6]$ respects $\mathcal{R}_2^{(3)} = \{r_{4,3}\}$.

Let $\mathcal{U}$ denote an unschedulability core for Γ. The constraint that the PPOs in $\mathcal{U}$ cannot be simultaneously satisfied is:

$$\sum_{r_{i,j} \in \mathcal{U}} p_{i,j} \leq |\mathcal{U}| - 1 \tag{6}$$

Remark 2. Constraint (6) are more friendly to ILP solver. Its coefficients on the left hand side are all small integral (0 or 1), which in many cases makes the ILP solver more efficient [17].

We now prove that the set of *all* unschedulability cores is a necessary and sufficient condition that makes the system unschedulable.

Theorem 2. Let Γ be a *schedulable* task system and $\mathcal{R}$ be a PPO set on Γ. Γ is not $\mathcal{R}$-schedulable if and only if $\mathcal{R}$ contains at least one unschedulability core.

Proof. **Necessity:** It is straightforward by the definition of unschedulability core and the result in Equation (5).
Sufficiency: Proof by induction on the cardinality of $\mathcal{R}$.

Base case. Let $\mathcal{R}$ be any PPO set such that $|\mathcal{R}| = 1$ and Γ is not $\mathcal{R}$-schedulable. The only proper subset of $\mathcal{R}$ is $\mathcal{R}' = \emptyset$. Since Γ is schedulable, $\mathcal{R}$ itself is an unschedulability core.

Inductive step. Assume any PPO set of cardinality from 1 to $k-1$ such that Γ is not schedulable contains an unschedulability core. We prove that any $\mathcal{R}$ of cardinality k such that Γ is not $\mathcal{R}$-schedulable shall contain an unschedulability core.

By Definition 3, there must exist $\mathcal{R}' \subset \mathcal{R}$ such that Γ is not $\mathcal{R}'$-schedulable (otherwise, $\mathcal{R}$ itself is an unschedulability core). Now we consider $\mathcal{R}'$, which has a cardinality smaller than k. By the assumption for the inductive step, $\mathcal{R}'$ contains an unschedulability core, so does $\mathcal{R}$. □

Theorem 2 implies that if all the unschedulability cores for the system are known, then we can formulate the exact schedulability region by adding Constraint (6) for each unschedulability core. However, the number of unschedulability cores may be exponential to the number of tasks. Hence, it is inefficient to rely on the complete knowledge of the unschedulability cores. In the following, we develop procedures that judiciously and efficiently add a selective subset of unschedulability cores to gradually form the schedulability region. In the rest of the section, we present a procedure (Algorithm 1) that, starting from an unschedulable priority assignment, efficiently calculates an unschedulability core. In the next section, we propose an unschedulability core guided optimization algorithm.

Algorithm 1 takes as input the task set Γ and a PPO set $\mathcal{R}$, where Γ is not $\mathcal{R}$-schedulable. It leverages Remark 1 and checks if those subset of $\mathcal{R}$ with cardinality $|\mathcal{R}| - 1$ (i.e., one less element) can allow Γ schedulable. Hence, it iterates through and tries to remove each element r in $\mathcal{R}$. If the resulted PPO set still does not allow Γ to be schedulable, then r is removed. In the end, it will return one unschedulability core. Since the cardinality of $\mathcal{R}$ is $O(n^2)$, the number of iterations in Algorithm 1 is $O(n^4)$.

Remark 3. Note that $\mathcal{R}$ may contain more than one unschedulability cores. To compute a different core than known ones, a straightforward way is to start with a subset $\mathcal{R}' \subset \mathcal{R}$ such that $\mathcal{R}'$ is not a superset of any known core.

Algorithm 1 Algorithm for Computing Unschedulability Core

```
1: function UNSCHEDCORE(Task set Γ, PPO set R)
2:     for each r ∈ R do
3:         if Γ is not R\{r}-schedulable then
4:             remove r from R
5:             restart the for loop
6:         end if
7:     end for
8:     return R
9: end function
```

The above algorithm depends on an efficient way to check, given a PPO set $\mathcal{R}$, whether Γ is $\mathcal{R}$-schedulable (Line 3 in the algorithm). Depending on the type of system, the forms of such procedures vary. In this paper, we assume that Audsley's algorithm is applicable to the task system. For such systems, a *revised Audsley's algorithm* can check if Γ is $\mathcal{R}$-schedulable. Similar to Audsley's algorithm, it iteratively picks a task that can be assigned at a particular priority level starting from the lowest priority. However, when choosing the candidate task, it shall guarantee that assigning the priority does not violate any partial priority order in $\mathcal{R}$. This is done by checking if the current task is a legal candidate: A task τ_i is a legal candidate if and only if (a) it has not been assigned with a priority; and (b) all the tasks that should have a lower priority than τ_i according to $\mathcal{R}$ have already been assigned.

IV. Unschedulability Core Guided Optimization

In this section, we develop a new optimization algorithm, inspired by three observations. *First*, the schedulability condition is often difficult to be explicitly included in optimization, as the optimization procedure often needs to check the feasibility of a large number of solutions. For example, the problem presented in Section V-A essentially is to assign priority orders to n mixed-criticality function blocks under Adaptive Mixed Criticality (AMC) scheduling [18], resulting $n!$ different solutions. For a decent sized problem, e.g., $n = 30$, the design space contains $30! = 2.7 \times 10^{32}$ solutions, 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. *Second*, the complexity of the schedulability analysis may prevent us from leveraging existing optimization frameworks. For example, the most accurate schedulability analysis for AMC, AMC-max [18], hinders a possible formulation in ILP: It requires to check, for each possible time instant s of criticality change, whether the corresponding response time is within the deadline, However, the range of s is unknown a priori as it depends on the task response time in LO mode. *Third*, the optimization objective may only be sensitive to a small set of schedulability constraints.

As discussed earlier, finding all the unschedulability cores is hardly practical as the total number of unschedulability cores may grow exponentially with the size of system. Hence, we consider the lazy constraint paradigm that only selectively adds unschedulability cores into the problem formulation. The paradigm starts with a relaxed problem containing only a selected subset of constraints that leaves out all the schedulability conditions. The rest of the constraints is temporarily put in a lazy constraint pool. A constraint from the pool is

Algorithm 2 Unsched Core Guided Optimization Algorithm

1: **function** FINDOPTIMAL(Task set Γ, Integer k)
2: Build initial problem Π as in (7)
3: **while** true **do**
4: $\mathbf{X}^* =$SOLVE(Π)
5: **if** Π is not feasible **then**
6: **return** Infeasibility
7: **end if**
8: Compute $\mathcal{R}_{\mathbf{X}^*}$ as in (8)
9: **if** Γ is not $\mathcal{R}_{\mathbf{X}^*}$-schedulable **then**
10: Compute k unschedulability cores $\mathcal{U}_1, ..., \mathcal{U}_k$ with Algorithm 1
11: Add Constraint (9) to Π
12: **else**
13: **return** priority assignment $\mathcal{P}$ respecting $\mathcal{R}_{\mathbf{X}^*}$
14: **end if**
15: **end while**
16: **end function**

added back only if it is violated by the solution returned for the relaxed problem. In addition, instead of adding the violated schedulability conditions back, we may derive some more suitable ones to be added. Specifically, when we find out the system is unschedulable because some tasks violate their deadlines, instead of adding the schedulability constraints of these tasks, we may use the concept of unschedulability core as a much more compact representation of these constraints.

The proposed procedure is summarized in Algorithm 2. It takes as input arguments a task system Γ and an integer number k which denotes the maximum number of unschedulability cores to compute for each infeasible solution (see Remark 5). The algorithm works as follows.

Step 1. Instantiate the initial problem Π as

$$\begin{array}{ll} & \min C(\mathbf{X}) \\ s.t. & \mathbf{F}'(\mathbf{X}) \leq 0 \end{array} \tag{7}$$

Different from the original problem in (3), $\mathbf{F}'(\mathbf{X}) \leq 0$ excludes all the schedulability conditions from $\mathbf{F}(\mathbf{X}) \leq 0$.

Step 2. Solve the initial problem Π in (7). If Π is not feasible, then the algorithm terminates. Otherwise, let $\mathbf{X}^*$ denote the optimal solution obtained. Construct the corresponding PPO set $\mathcal{R}_{\mathbf{X}^*}$ as follows

$$\mathcal{R}_{\mathbf{X}^*} = \{r_{i,j} | p_{i,j} = 1 \text{ in } \mathbf{X}^*\} \tag{8}$$

Apply the revised Audsley's algorithm to test $\mathcal{R}_{\mathbf{x}^*}$-schedulability. If Γ is not $\mathcal{R}_{\mathbf{X}^*}$-schedulable, go to step 3. Otherwise go to step 4.

Step 3. Apply Algorithm 1 to compute k number of unschedulability cores $\mathcal{U}_1, ..., \mathcal{U}_k$. Update problem Π by adding the following constraints, then go to step 2.

$$\sum_{r_{i,j} \in \mathcal{U}_m} p_{i,j} \leq |\mathcal{U}_m| - 1, \quad \forall m = 1, ..., k \tag{9}$$

Step 4. Return the optimal priority assignment $\mathcal{P}$ that respects $\mathcal{R}_{\mathbf{X}^*}$ with the revised Audsley's algorithm.

Remark 4. As the number of unschedulability cores is clearly bounded, Algorithm 2 shall always terminate as it either returns a solution (Line 13), or adds more constraints corresponding to the newly detected (and different from known ones) unschedulability cores to the problem (Line 11), or reports infeasibility (Line 6). If a solution is returned, Algorithm 2 guarantees that it is feasible since it confirms that the system Γ is $\mathcal{R}_{\mathbf{X}^*}$-schedulable. Algorithm 2 also guarantees the optimality of the solution since it always maintains an over-approximation of the exact feasibility region (a subset of all constraints).

Remark 5. The parameter k in Algorithm 2 does not affect the optimality of the algorithm, but influences its runtime. A good choice is $k = 5$, as in our experiments it is almost always within 10% compared to the optimal setting.

The efficiency of Algorithm 2 comes at three-fold. First, the procedure avoids modeling the complete schedulability region. Instead, it explores, in an objective-guided manner, much simpler over-approximation formulations that are sufficient to establish optimality. Second, it hides the potentially complicated system schedulability conditions by converting them into a simple form of unschedulability cores using a separate and dedicated algorithm. This also makes the framework easily adaptable to other systems whose analysis are difficult to formulate in frameworks such as ILP. Third, the conversion to unschedulability core is essentially a generalization from one infeasible solution to many, which is a key in allowing a fast convergence rate to true optimality.

We now illustrate the algorithm by applying it on the example Γ_e in Table I, where the parameter k is set to 1.

Example 3. Consider the following objective function

$$C(\mathbf{X}) = -p_{3,1} - p_{4,1} - p_{4,2} - p_{4,3} - p_{5,4} \tag{10}$$

The algorithm constructs the initial problem Π as (7), where $\mathbf{F}'(\mathbf{X}) \leq 0$ only contains the set of antisymmetry and transitivity constraints as defined in (2).

The algorithm enters the first iteration and solves Π by possibly using ILP solvers. The solution is (for simplicity, we omit those not affecting the objective function)

$$\mathbf{X}^* = [p_{3,1}, p_{4,1}, p_{4,2}, p_{4,3}, p_{5,4}] = [1, 1, 1, 1, 1]$$

The corresponding PPO set is $\mathcal{R}_{\mathbf{X}^*} = \{r_{3,1}, r_{4,1}, r_{4,2}, r_{4,3}, r_{5,4}\}$. Clearly, Γ_e is not $\mathcal{R}_{\mathbf{X}^*}$-schedulable. The algorithm computes one unschedulability core of $\mathcal{R}_{\mathbf{X}^*}$ as $\mathcal{U}_1 = \{r_{4,3}, r_{5,4}\}$. The problem Π then becomes

$$\begin{array}{ll} & \min C(\mathbf{x}) \\ s.t. & \mathbf{F}'(\mathbf{X}) \leq 0 \\ & p_{4,3} + p_{5,4} \leq 1 \end{array} \tag{11}$$

In the second iteration, solving (11) gives the solution $\mathbf{X}^* = [p_{3,1}, p_{4,1}, p_{4,2}, p_{4,3}, p_{5,4}] = [1, 1, 1, 1, 0]$. The corresponding PPO set is $\mathcal{R}_{\mathbf{X}^*} = \{r_{3,1}, r_{4,1}, r_{4,2}, r_{4,3}, r_{4,5}\}$. Since Γ is still not $\mathcal{R}_{\mathbf{X}^*}$-schedulable, the algorithm computes another unschedulability core as $\mathcal{U}_2 = \{r_{3,1}\}$. The problem Π is correspondingly updated as

$$\begin{array}{ll} & \min C(\mathbf{x}) \\ s.t. & \mathbf{F}'(\mathbf{X}) \leq 0 \\ & p_{4,3} + p_{5,4} \leq 1, \quad p_{3,1} \leq 0 \end{array} \tag{12}$$

In the third iteration, the problem in (12) is solved to obtain the solution $\mathbf{X}^* = [p_{3,1}, p_{4,1}, p_{4,2}, p_{4,3}, p_{5,4}] = [0, 1, 1, 1, 0]$. The corresponding PPO set is $\mathcal{R}_{\mathbf{X}^*} = \{r_{1,3}, r_{4,1}, r_{4,2}, r_{4,3}, r_{4,5}\}$. At this point, Γ_e becomes $\mathcal{R}_{\mathbf{X}^*}$-schedulable. The algorithm then terminates and returns the following optimal solution

$$\mathcal{P} = [\pi_4 > \pi_1 > \pi_2 > \pi_3 > \pi_5 > \pi_6]$$

V. Experimental Evaluation

In this section, we demonstrate the advantages of the proposed approach with two example problems.

A. Implementation of Mixed-Criticality Simulink Models

The first is the optimization of the semantics-preserving implementation of mixed-criticality Simulink models. We briefly describe the problem below. A Simulink model is a Directed Acyclic Graph (DAG) where nodes represent function blocks and links represent data communication between function blocks [19]. We assume each function block is implemented in a dedicated task (hence *use the terms function block and task interchangeably*). The semantics-preserving implementation of the Simulink model has to match its functional behavior. This typically requires the addition of a Rate Transition (RT) block between a reader and a writer with different but harmonic periods, which is a special type of wait-free communication buffers. However, the costs of RT blocks are additional memory overheads and in some cases, functional delays in result delivery. The latter degrades control performance. Consider a fast reader τ_r and slow writer τ_w that writes to τ_r. Assigning higher priority to τ_r generally helps schedulability as it conforms with the rate monotonic policy. However, since the reader now executes before the writer, an RT block is needed to store the data from the previous instance of the writer, which also incurs a functional delay. On the other hand, if τ_r can be assigned with a lower priority while keeping the system schedulable, then no RT block is needed and no functional delay is introduced. The software synthesis of Simulink model is to exploit *priority assignment as the design variable to minimize the functional delays introduced by the RT blocks* (hence improving control quality). We note that Audsley's algorithm is no longer optimal as schedulability is not the only constraint, and the design should minimize the weighted sum of functional delays.

We consider the problem where the Simulink model contains functional blocks with different criticality levels, scheduled with Adaptive Mixed Criticality (AMC) scheme [18]. This problem is NP-hard as the special case where all tasks are LO-critical is proven to be NP-hard [19]. AMC scheduled systems can be analyzed with two methods [18]: AMC-max and AMC-rtb. The straightforward ILP formulation of AMC-max is excluded due to its extreme high complexity (see Section IV). We also include brute-force branch-and-bound (BnB) algorithms, to evaluate the benefit from modern ILP solvers (e.g., CPLEX). The list of compared methods is:

- **UC-AMC-max**: Unschedulability core guided algorithm (Algorithm 2) with AMC-max as schedulability analysis;
- **UC-AMC-rtb**: Algorithm 2 with AMC-rtb analysis;
- **ILP-AMC-rtb**: ILP with AMC-rtb analysis, solved by CPLEX;
- **BnB-AMC-rtb**: BnB algorithm with AMC-rtb analysis;
- **BnB-AMC-max**: BnB with AMC-max analysis.

We use TGFF [20] to generate random systems. Each function block has at most an in-degree of 3 and out-degree of 2. We first randomly choose a number of sink function blocks and assign it with HI-criticality. The criticality of the remaining blocks are determined by the following rules:

- If a block is the predecessor of any HI-critical block, then it is assigned a HI-critical level as well;
- All blocks not assigned HI-critical by the above rule are assigned LO-critical level.

We first study the scalability with respect to the number of function blocks which varies from 5 to 100. The system utilization in LO-criticality mode is randomly selected from [0.5, 0.95]. For each task in the system, utilization is generated using the UUnifast-Discard algorithm [21]. Task period is randomly chosen from a predefined set of values {10, 20, 40, 50, 100, 200, 400, 500, 1000}, which contains all the periods for the real-world automotive benchmark in [22]. The criticality factor of HI-criticality task is uniformly set to 2.0 ($\frac{C_i(HI)}{C_i(LO)} = 2.0$). We generate 1000 systems and report their average for each point in the plots. k is set to 5 in Algorithm 2 as the corresponding runtime is typically within 10% of the optimal setting.

Figs. 1 and 2 illustrate the runtime and minimized functional delay of these methods, respectively. **AMC-max** based methods give better optimal solution due to the better accuracy of **AMC-max** than **AMC-rtb**, but they also run slower than their counterpart based on **AMC-rtb** (e.g., **UC-AMC-max** vs. **UC-AMC-rtb**). The superiority of branch-and-bound based algorithms in small-sized systems is mainly due to the overhead in ILP model construction in other methods, which consumes a significant portion of the runtime when the ILP formulation is rather simple. The scalability for **UC-AMC-rtb** and **UC-AMC-max** is remarkably better than that of the other methods. For example, for systems with 35 tasks, the unschedulability core guided techniques are more than 1000 times faster compared to **ILP-AMC-rtb**. In addition, **UC-AMC-rtb** and **UC-AMC-max** are quite close in their runtimes, demonstrating that Algorithm 2 is not very sensitive to the complexity of the schedulability analysis. Finally, **ILP-AMC-rtb** scales much better than **BnB-AMC-rtb**. This demonstrates that modern ILP solvers, which are equipped with various sophisticated techniques, are generally more efficient than brute force BnB.

We also evaluate the scalability of **UC-AMC-max** and **UC-AMC-rtb** with respect to different system utilization ranging from 0.05 to 0.90. The number of function blocks in a system is fixed to 70 while the other parameters remain the same. The runtime of both methods stays almost the same (around 1000ms per system) when the utilization is larger than 40%. This means that the scalability of unschedulability core guided algorithms is not sensitive to system utilization.

B. Minimizing Memory Usage of AUTOSAR Components

The second case study is to minimize the memory usage of AUTOSAR components [23], where a set of runnables (the AUTOSAR term for function blocks) communicates through shared buffers that shall be appropriately protected to ensure data integrity. We assume that each runnable is implemented in a dedicated task, and use the terms runnable and task interchangeably. We consider the problem for the optimal selection of (a) the priority assignment to runnables; (b) the selection of the appropriate mechanism for protecting shared buffers among a set of possible choices, including ensuring absence of preemption, lock-based method (priority ceiling semaphore lock), and wait-free method [23]. These mechanisms have different timing constraints and memory cost. *Ensuring absence of preemption* has no memory or timing cost, but it requires the two communicating tasks satisfy that the lower priority task always finish before the next activation of the higher priority task. *Wait-free method* imposes no extra timing constraints but incurs a memory cost equal to the size of share buffer. *Lock-based method* introduces blocking delay to higher priority tasks but reduces the memory overhead to

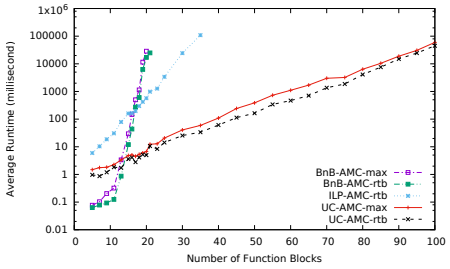

Fig. 1: Runtime vs. System Size for Implementation of Simulink Model

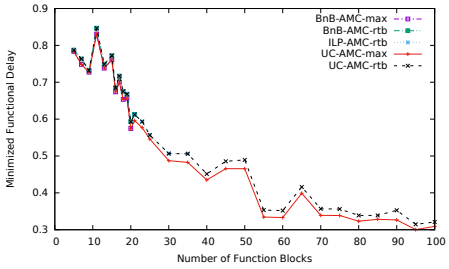

Fig. 2: Objective vs. System Size for Implementation of Simulink Model

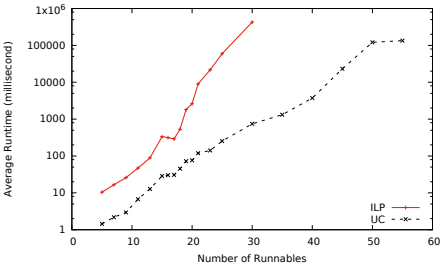

Fig. 3: Runtime vs. System Size for Memory Minimization of AUTOSAR

minimal (only one-bit for implementing semaphore locks). This problem has been demonstrated to be NP-hard [24].

The objective is to minimize the total memory usage while ensuring system schedulability. In addition to the variables of priority assignment, for each shared memory buffer v between two tasks τ_i and τ_j, we introduce a set of binary variables defined as follows:

$$\begin{cases} l_v = 1 \Leftrightarrow \text{Using semaphore lock to protect } v \\ w_v = 1 \Leftrightarrow \text{Using wait-free method to protect } v \\ l_v = 0 \wedge w_v = 0 \Leftrightarrow \text{Absence of preemption between } \tau_i \text{ and } \tau_j \end{cases}$$

Since it is sufficient to protect each shared buffer with one of the mechanisms, the following constraints are added

$$l_v + w_v \leq 1, \quad \forall v$$

We compare our technique (denoted as **UC**) with request bound function based ILP formulation [13] (denoted as **ILP**) on randomly generated synthetic task systems. We omit BnB as it is demonstrated to be less scalable than ILP. Task utilization and period are generated in the same way as Section V-A. Each task communicates with 0 to 5 other tasks. The size of the share buffer is randomly selected between 1 to 512 bytes. The WCET of the critical section for each task τ_i on each share buffer is randomly generated from $(0, 0.1 \cdot C_i]$.

Fig. 3 plots the runtime versus system sizes for the case study. As in the figure, **UC** always takes significantly smaller runtime than **ILP** while giving the same optimal results, and the difference becomes larger with larger systems. For example, for systems with 25 runnables, **UC** runs about 200 times faster than **ILP**. This demonstrates that the carefully crafted algorithm **UC** can achieve much better scalability than the other exact algorithms while maintaining optimality.

VI. Conclusions

In this work, we presented the concept of unschedulability core, a compact representation of schedulability conditions for use in design optimization of priority assignment in real-time systems. We developed efficient algorithms for calculating unschedulability cores and optimizing priority assignment. Experiments show that our unschedulability core guided optimization framework can provide optimal solutions while scaling much better than exact approaches.

References

[1] S. Chakraborty, "Keynote talk: Challenges in automotive cyber-physical systems design," in *International Conference on VLSI Design*, 2012.

[2] K. Bazaka and M. V. Jacob, "Implantable devices: issues and challenges," *Electronics*, 2(1): 1–34, 2012.

[3] N. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters*, 79(1): 39–44, 2001.

[4] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns, "A review of priority assignment in real-time systems," *J. Syst. Archit.*, 65(C): 64–82, Apr. 2016.

[5] K. W. Tindell *et al.*, "Allocating hard real-time tasks: An np-hard problem made easy," *Real-Time Syst.*, 4(2): 145–165, May 1992.

[6] I. Bate and P. Emberson, "Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[7] A. Hamann, M. Jersak, K. Richter, and R. Ernst, "Design space exploration and system optimization with symta/s - symbolic timing analysis for systems," in *IEEE Real-Time Systems Symposium*, 2004.

[8] M. Saksena and Y. Wang, "Scalable real-time system design using preemption thresholds," in *IEEE Real-Time Systems Symposium*, 2000.

[9] H. Zeng, M. Di Natale, and Q. Zhu, "Minimizing stack and communication memory usage in real-time embedded applications," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 5s, pp. 149:1–149:25, Jul. 2014.

[10] Y. Wang and M. Saksena, "Scheduling fixed-priority tasks with preemption threshold," in *International Conference on Real-Time Computing Systems and Applications*, 1999.

[11] Q. Zhu *et al.*, "Optimization of task allocation and priority assignment in hard real-time distributed systems," *ACM Trans. Embedded Comput. Syst.*, vol. 11, no. 4, pp. 85:1–85:30, 2012.

[12] M. Panić *et al.*, "RunPar: an allocation algorithm for automotive applications exploiting runnable parallelism in multicores," in *International Conference on Hardware/Software Codesign and System Synthesis*, 2014.

[13] H. Zeng and M. Di Natale, "An efficient formulation of the real-time feasibility region for design optimization," *IEEE Transactions on Computers*, vol. 62, no. 4, pp. 644–661, April 2013.

[14] S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis, "Static probabilistic timing analysis for real-time systems using random replacement caches," *Real-Time Syst.*, vol. 51, no. 1, pp. 77–123, Jan. 2015.

[15] R. I. Davis and M. Bertogna, "Optimal fixed priority scheduling with deferred pre-emption," in *IEEE Real-Time Systems Symposium*, 2012.

[16] M. Stigge *et al.*, "Combinatorial abstraction refinement for feasibility analysis of static priorities," *Real-Time Systems*, 51(6): 639–674, 2015.

[17] A. N. Letchford and A. Lodi, "Strengthening chvátal-gomory cuts and gomory fractional cuts," *Oper. Res. Lett.*, 30(2): 74–82, Apr. 2002.

[18] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *IEEE Real-Time Systems Symposium*, 2011.

[19] M. D. Natale, L. Guo, H. Zeng, and A. Sangiovanni-Vincentelli, "Synthesis of multi-task implementations of simulink models with minimum delays," *IEEE Trans. Industrial Informatics*, 6(4): 637–651, 2010.

[20] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *6th international workshop on Hardware/software codesign*, 1998.

[21] R. I. Davis and A. Burns, "Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *IEEE Real-Time Systems Symposium*, 2009.

[22] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2015.

[23] A. Ferrari, M. Di Natale, G. Gentile, G. Reggiani, and P. Gai, "Time and memory tradeoffs in the implementation of autosar components," in *Conference on Design, Automation and Test in Europe*, 2009.

[24] E. Wozniak *et al.*, "An optimization approach for the synthesis of autosar architectures," in *IEEE Conference on Emerging Technologies Factory Automation*, 2013.