Extending Periodic Task Sets with Sporadic Tasks: Computational Complexity and Exact Feasibility Tests[⋆]

Mohammad Ibrahim Alkoudsi[1] [0000−0002−1787−3284],
Sanjoy Baruah[4] [0000−0002−4541−3445],
Pontus Ekberg[2] [0009−0009−5282−9463],
Gerhard Fohler[1] [0000−0001−6162−2653],
Joël Goossens[3] [0000−0001−9524−8911], and
Maryam Moslehi[2]

[1] RPTU Kaiserslautern-Landau in Kaiserslautern, Germany
`alkoudsi@rptu.de`
`gerhard.fohler@rptu.de`
[2] Uppsala University in Uppsala, Sweden
`pontus.ekberg@it.uu.se`
`maryam.moslehi.baharanchi@it.uu.se`
[3] Université Libre de Bruxelles in Brussels, Belgium
`joel.goossens@ulb.be`
[4] Washington University in St. Louis, USA
`baruah@wustl.edu`

Abstract. Given a periodic task system that is guaranteed to be schedulable upon a preemptive uniprocessor, this paper addresses the problem of determining whether schedulability is preserved upon the addition of a set of sporadic tasks. The precise computational complexity of the problem is established, and exact tests are designed, formally proven correct, and evaluated experimentally to assess their practicality for runtime deployment.

Keywords: Real-Time Scheduling · Periodic and sporadic Tasks · Preemptive Uniprocessors · Exact Feasibility Analyses

1 Introduction and Problem Statement

This paper investigates the following real-time schedulability analysis problem:

> *Given a task system comprising a collection of asynchronous periodic tasks that is known to be schedulable upon a preemptive uniprocessor*

⋆ The work presented in this paper was supported in part by the Fonds National de la Recherche Luxembourg (FNR) and the Deutsche Forschungs Gemeinschaft (DFG) through the Core-Inter project ReSAC, the Swedish Research Council through grant 2023-04586, and the US National Science Foundation (CNS-2141256, CNS-2229290 (CPS), and CNS-2502855).

platform, is schedulability preserved upon adding a collection of sporadic tasks to the task system?

Below we first explain the reasons that have motivated us to study this particular problem, and then define the problem more formally.

Motivation. One of the earliest (and still one of the most common) use-cases for periodic tasks [1,2] is in implementing control loops upon a common computing platform[5]. The Time-Triggered Architecture (TTA) [3–6] offers a disciplined approach for scheduling such periodic tasks in safety-critical distributed systems ensuring reliability and predictability. It is particularly well-suited for applications with complex timing constraints, e.g., precedence, mutual exclusion, and distributed end-to-end deadlines, as reflected in the successful commercialization of TTA by the company TTTech.

The strict adherence of conventional time-triggered (TT) implementations to offline defined schedules prevents runtime flexibility: workloads and scheduling decisions are completely determined offline. At runtime, scheduling reduces to a straightforward dispatching process guided by these pre-computed schedules. This inherent rigidity hinders the direct application of the TT approach for implementing applications exhibiting a certain degree of runtime uncertainty.

Introducing flexibility into TT systems requires careful analysis to avoid compromising system properties [7]. Several methods have been proposed in [8–12] to mitigate the rigidity of conventional TT scheduling. These methods dynamically adjust the offline schedules to enable the efficient integration of event-triggered tasks while maintaining timing guarantees and preserving TT properties.

Our goal is to further extend prior work and enable incorporating even greater flexibility into the TT approach, by allowing for the admission, *during runtime*, of an entire collection of sporadic tasks if it is possible to do so without missing any deadlines. Further, we do not restrict adding sporadic tasks to a predetermined time, i.e., they can be added at their initial arrival time, as long as feasibility is maintained. Since such admission control is being done during runtime, it is imperative that it be performed efficiently. This capability is essential for emerging domains such as real-time cloud computing and autonomous safety-critical systems. In these environments, services are provisioned dynamically, and applications and their timing requirements may vary in response to changing conditions. Predefining operational modes for every conceivable scenario is both resource-inefficient and, in many cases, infeasible, particularly when not all system variations can be anticipated at design time.

The problem considered. We now provide a formal definition of the scheduling problem that is studied in this paper. We assume that we have a set $\mathit{\Gamma}_{\mathrm{P}}$ of periodic tasks, that is *a priori* known to be schedulable upon a preemptive uniprocessor. We seek to determine whether, if we were to now add a set $\mathit{\Gamma}_{\mathrm{S}}$ of

[5] E.g., the widely-cited Liu and Layland paper [1] opens with the words "The use of computers for control and monitoring of industrial processes has expanded greatly in recent years [. . .] Often, the computer used in such an application is shared between a certain number of time-critical control and monitor functions. . . "

sporadic tasks to the processor, all deadlines of all the tasks in both Γ_P and Γ_S will continue to always be met; i.e., is the set of tasks $\Gamma \stackrel{\text{def}}{=} \Gamma_\text{P} \cup \Gamma_\text{S}$ schedulable? In greater detail,

1. Γ_P is a set of $\underline{\text{four-parameter periodic tasks}}$ [2,13]; each task $\tau_i \in \Gamma_\text{P}$ is a 4-tuple (ϕ_i, C_i, D_i, T_i) where
   - ϕ_i is the *initial offset* of τ_i: the first job of τ_i arrives at time-instant ϕ_i;
   - C_i is the *worst-case execution time* (wcet) of τ_i: each job of τ_i needs to execute for no more than C_i time units;
   - D_i is the *relative deadline* of τ_i: each job must complete execution within a duration D_i of its arrival; and
   - T_i is the *period* of τ_i: successive jobs arrive exactly T_i time units apart. We assume *constrained deadlines*, i.e., that $D_i \le T_i$.
2. Γ_S is a set of $\underline{\text{three-parameter sporadic tasks}}$ [14]: each task $\tau_i \in \Gamma_\text{S}$ is a 3-tuple (C_i, D_i, T_i) where
   - C_i and D_i are the wcet and relative deadline respectively of τ_i, with the same interpretations as above for periodic tasks; and
   - T_i is the *period* of τ_i, with the interpretation (different from above) that successive jobs arrive at least T_i time units apart. We assume $D_i \le T_i$.
3. We are considering the scheduling of $\Gamma \stackrel{\text{def}}{=} \Gamma_\text{P} \cup \Gamma_\text{S}$ upon a preemptive uniprocessor platform. Given the optimality of the Earliest Deadline First scheduling algorithm (EDF) upon preemptive uniprocessors [1,15], we will, for the most part, assume that EDF is the runtime scheduling that is used.

Known complexity results. Schedulability verification is known to be strongly coNP-complete both for task systems comprising four-parameter periodic tasks [13] and for task systems comprising three-parameter sporadic tasks [16] – this implies that polynomial or pseudo-polynomial time algorithms for verifying the schedulability of either of these task set systems are not likely to exist.

Additional results are known for *bounded-utilization systems* — task systems for which the total utilization $\text{U}(\Gamma)$, defined as the sum of each task's utilization $U_i \stackrel{\text{def}}{=} C_i/T_i$, is a constant fraction strictly smaller than one, i.e., $\text{U}(\Gamma) < 1$.

Under bounded-utilization conditions, the schedulability verification problem for three-parameter sporadic task systems is weakly coNP-complete [17]. This follows from the fact that the length of the analysis interval - the critical window within which all potential deadline violations must be checked - is bounded by a value that grows only pseudo-polynomially with respect to the numerical magnitudes of the task parameters. Consequently, pseudo-polynomial time algorithms are applicable in this setting [14]. In contrast, for four-parameter periodic task systems with unequal offsets, the length of the analysis interval grows exponentially with the size of task set parameters. As a result, these systems remain coNP-complete even under bounded-utilization conditions.

Implications to our problem. The strong coNP-hardness result for general three-parameter sporadic task systems [16] implies that we are unlikely to be able to solve our problem in polynomial or pseudo-polynomial time (to see this, suppose that the periodic task system Γ_{P} is empty, determining whether $\Gamma \stackrel{\text{def}}{=} \Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is schedulable is equivalent to determining whether Γ_{S} is schedulable). However, the above argument does not apply for bounded-utilization systems — since the sporadic task system Γ_{S} also necessarily has bounded utilization, an empty periodic task system Γ_{P} would allow our problem to be solved in pseudo-polynomial time using the processor-demand analysis [14]. While determining whether $\Gamma \stackrel{\text{def}}{=} \Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is schedulable is strongly coNP-hard even for bounded-utilization systems (since determining whether Γ_{P} is schedulable is strongly coNP-complete [13, Corollary 3.1]), notice that our problem only requires us to determine Γ's schedulability *given* that Γ_{P} is schedulable. That is, our problem is stated as a ***promise problem*** [18]: determine the schedulability of $\Gamma \stackrel{\text{def}}{=} \Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ under the "promise" that Γ_{P} is schedulable.

Our Results. Our first major result is a negative one: we prove that determining whether $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is schedulable under any work-conserving scheduler remains strongly coNP-hard for bounded-utilization systems despite the promise that Γ_{P} is schedulable (Theorem 1). This hardness result persists even when Γ_{S} comprises a single implicit-deadline sporadic task ($D_i = T_i$) and the total utilization of $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is bounded by an arbitrarily small constant. In essence, verifying whether a schedulable Γ_{P} remains feasible after adding a single sporadic task is computationally as hard as establishing the schedulability of Γ_{P} from scratch.

On the brighter side, we observe that since the problem for EDF is in coNP, it follows from prior results in [13] that it can be represented in polynomial time as the feasibility problem for an Integer Linear Program (ILP)[6], and thereby solved by using an ILP-solver. We do, however, have a negative result to accompany this positive one: we show (Lemma 3) that if we were to use Fixed-Priority scheduling [2] rather than EDF, then it is unlikely that we can reduce schedulability verification to the feasibility problem for an ILP in polynomial time.

We have also developed some feasibility tests for the problem under consideration. First, we propose an exact feasibility test based on *processor demand analysis (PDA)* [14], that exploits some properties identified in Theorem 2 to significantly reduce the number of test cases that are typically used for such PDA analysis when asynchronous periodic tasks are involved.

Recognizing PDA's exponential time and polynomial space complexity, we introduce PDA*, a novel approach that shifts a substantial portion of the computation to the pre-runtime phase. This is achieved by pre-computing the processor demand of periodic tasks, which reduces the runtime complexity to pseudo-polynomial at the cost of pseudo-polynomial space. We further optimize runtime with QPDA* by incorporating the quick convergence technique of [19]. QPDA* achieves average-case speedups of 2-3× over PDA* while maintaining the same memory footprint. In contrast, our ILP formulation of the problem exhibits complexity comparable to the original PDA. Empirical evaluation on synthetic task sets and automotive benchmarks, including a comparison with Slot-Shifting's

[6] Technically, EDF-schedulability can be represented as the *in*-feasibility problem of an ILP.

offline test, demonstrates the efficiency of the QPDA* for runtime deployment. Under the constraints of contemporary real-time systems, QPDA* achieves a 2-3 order of magnitude improvement in runtime performance over the original PDA, while requiring less than a few hundred kilobytes to store preprocessed demand.
Organization. The remainder of this manuscript is organized in the following manner. Section 2 reviews related work that addresses similar instances of the problem under both fixed and dynamic priority scheduling. Section 3 presents our computational complexity results; Section 4, our proposed feasibility tests, PDA and its optimized variant PDA*, followed by the formulation of the problem as an ILP model in Section 5. Section 6 reports the results of our experimental evaluation of the feasibility tests. Finally, Section 7 summarizes our contributions and outlines directions for future research.

2 Related Work

Existing closely related works assume TT activation, where the scheduler is invoked at the start of predefined time slots in a scheduling table. This section categorizes these works based on the timing of TT schedule modifications into:
Offline schedule modification. In [20], a hierarchical approach for handling sporadic tasks in a TT schedule is proposed. A periodic server abstraction, e.g., one or multiple time slots in the TT schedule, is reserved for handling sporadic tasks based on a second-level fixed-priority-based scheduler. The advantage of this method is that it decouples the schedulability of sporadic tasks from the TT schedule generation problem. The system designer, however, has to solve the non-trivial server design problem, i.e., the problem of deriving suitable server parameters under which the sporadic tasks are schedulable. The authors propose two methods for implementing this hierarchical approach: simple and advanced polling. Simple polling assigns each sporadic task to its own periodic server, which, while simple, it can cause over-utilization when deadlines and periods diverge, resulting in suboptimal schedulability performance. In contrast, advanced polling involves assigning multiple sporadic tasks to the same periodic server, where schedulability verification is based on the response time of sporadic tasks. The method *Burst Limiting Least Laxity First* (B3LF) [21] has shown the best scheduling results under this category considering both sporadic and periodic tasks. It assumes that periodic tasks have the highest priority and are scheduled offline, whereas sporadic tasks are assigned fixed unique priorities and serviced dynamically in the free time slots of the offline-generated TT schedule. Since B3LF assumes periodic tasks have zero offsets, the worst-case arrival time for sporadic tasks is unambiguous: it occurs at the start of the longest synchronous busy period of periodic tasks, i.e., at $t = 0$. However, since such a long interference can lead to low schedulability, the method derives a *Burst Limiting Constraint (BLC)* based on real-time calculus. The BLC constrains how big and where busy periods from TT tasks can be in order to reposition idle slots in between them, s.t. the sporadic task set can be guaranteed. However, even optimal dynamic priority algorithms s.a. EDF or LLF lead to suboptimal schedulability

when used with the BLC to generate the TT schedule. This is because these algorithms make decisions based on the current set of ready task, and with no view on future released tasks. To alleviate the issue, the authors added a virtual task with dynamic laxity to the Least Laxity First (LLF) algorithm, which when chosen, an idle slot is inserted in the schedule, effectively delaying the execution of periodic tasks.

Online schedule modification methods. This category includes methods that adjusts the allocation of time slots to tasks at runtime based on dynamic priority rules, with the end goal of servicing event-triggered tasks more efficiently. *Slot-Shifting* [11] is a well-known method within this category and operates in two distinct phases. In the offline phase, Slot-Shifting supplants the fixed time slot allocation in the scheduling table with less rigid target execution windows, defined for each task on each node. It divides each scheduling table into disjoint capacity intervals created from the deadlines of the target execution windows of tasks. Each interval is annotated with the available spare processing capacity, computed based on the assumption of as-late-as-possible execution of TT tasks, while still preserving their timing constraints. In the online phase, the local dispatcher on each node uses and continuously updates spare capacities to safely accommodate aperiodic tasks. This is done on a slot-by-slot basis following EDF. In [10], Slot-Shifting was extended from guaranteeing individual aperiodic tasks to sporadic task sets. The extension includes an offline guarantee test and online scheduling algorithm. The offline test pessimistically assumes that TT tasks are always executed as late as possible, which can adversely impact its schedulability results. In contrast, the feasibility tests proposed in this paper are exact and can be easily integrated into Slot-Shifting and the more recent *Dynamic Task Set Scheduler (DTSS)* [12] with minimal effort.

3 Computational Complexity Results

We will prove that determining if $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is schedulable is generally a hard problem (strongly coNP-hard), even if we a priori know that Γ_{P} is schedulable in isolation (the promise) and when Γ_{S} contains a single implicit-deadline sporadic task and the utilization of $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is bounded by an arbitrarily small constant. Then we will show that the decision version of this problem is in coNP for EDF scheduling, also without any of the restrictions mentioned above, and can therefore be reduced to (the complement of) Integer Linear Programming (ILP) and solved with an ILP solver. In contrast, for FP scheduling we show that it is impossible to reduce in polynomial time to ILP under commonly held complexity-theoretic conjectures.

First we prove hardness of a restricted version of *Simultaneous Congruences Problem* (SCP) that will be useful in our later reduction. SCP was introduced by Leung and Whitehead [2] and was shown to be strongly NP-complete by Baruah et al. [13].

Definition 1 (The Simultaneous Congruences Problem [2]). *An instance (A, k) of SCP is a set $A = \{(a_1, b_1), \dots, (a_n, b_n)\}$ of n pairs of integers and an integer k, such that $2 \leq k \leq n$ and $a_i < b_i$ for all $(a_i, b_i) \in A$.*
The simultaneous congruences problem asks whether there exists a subset $A' \subseteq A$, where $|A'| \geq k$, and an $x \in \mathbb{N}$, such that $x \equiv a_i \pmod{b_i}$ for all $(a_i, b_i) \in A'$.

In other words, SCP asks if at least k of the congruence classes $(a_i \bmod b_i)$ overlap at any single value. Note that given any set A of pairs of integers as above, it is easy to determine if *all* the corresponding congruence classes overlap at some value using the Generalized Chinese Remainder Theorem (see, e.g., Knuth [22]).

Proposition 1 (Generalized Chinese Remainder Theorem). *Given natural numbers $a_1, \dots, a_n$ and $b_1, \dots, b_n$, where $a_i < b_i$, there exists an x such that $x \equiv a_i \pmod{b_i}$ for all $1 \leq i \leq n$ if and only if $a_i \equiv a_j \pmod{\gcd(b_i, b_j)}$ for all $1 \leq i < j \leq n$.*

Since recognizing overlapping congruence classes is easy (the condition in the Generalized Remainder Theorem can be checked in polynomial time) an important aspect of why SCP is hard is the fact that k can be significantly different from $|A|$, so that there can be exponentially many choices for the combination of k congruence classes to consider. Indeed, an instance (A, k) of SCP with $k = |A|$ is easily solved in polynomial time by a single application of the Generalized Chinese Remainder Theorem, and any instance where $|A| - k$ is bounded by some constant can be solved with a polynomial number of applications of the theorem. However, we show below that if we restrict k to be at least some constant fraction of $|A|$, then SCP remains strongly NP-complete.

Lemma 1 (λ-restricted SCP). *SCP is strongly NP-complete even when restricted to instances (A, k) with $k \geq \lambda |A|$, for any constant λ where $\lambda < 1$.*

Proof. Let the constant λ be given. We will reduce from unrestricted SCP (Definition 1) to the λ-restricted variant by means of a pseudo-polynomial transformation (as defined by Garey and Johnson [23]).
Given an instance (A, k) of unrestricted SCP, where $A = \{(a_1, b_1), \dots, (a_n, b_n)\}$, we construct an instance (A_λ, k_λ) of λ-restricted SCP as follows. First, let

$$m \overset{\text{def}}{=} \left\lceil \frac{\lambda n - k}{1 - \lambda} \right\rceil, \quad k_\lambda \overset{\text{def}}{=} k + m.$$

We then create A_λ by adding m extra pairs of numbers to A,

$$A_\lambda \overset{\text{def}}{=} A \cup \{(0, b_{n+1}), \dots, (0, b_{n+m})\},$$

where $b_{\max} = \max_{1 \leq i \leq n} b_i$ and b_{n+j} denotes the jth smallest prime number larger than $b_{\max}$.
First we note that the value of m is polynomially bounded in the size of instance (A, k). Then we need to show that values b_{n+j}, for $1 \leq j \leq m$, are bounded by

a two-variable polynomial in the size of instance (A, k) and its largest numerical value. Let p_r denote the r'th prime, and note that $p_r = O(r \log r)$ by the Prime Number Theorem. Clearly we have $b_{n+j} < p_{b_{\max}+j}$ for all $1 \le j \le m$, but then we also have $b_{n+j} = O((b_{\max} + j) \log(b_{\max} + j))$, and the value of b_{n+j} is indeed bounded by a polynomial in the size and largest numerical value of (A, k). Note that we can easily generate all the b_{n+j} in time bounded by a two-variable polynomial in the size of instance (A, k) and its largest numerical value, for example by simply considering consecutive values larger than $b_{\max}$ and testing each for primality. Then we observe that

$$k_\lambda \;=\; k + m \;=\; k + (1-\lambda)m + \lambda m \;\ge\; k + (1-\lambda)\frac{\lambda n - k}{1-\lambda} + \lambda m \;=\; \lambda |A_\lambda|,$$

and so (A_λ, k_λ) fulfills the requirement of λ-restricted SCP.
Finally we have to show that the reduction preserves the answer: that (A_λ, k_λ) is a positive instance of λ-restricted SCP if and only if (A, k) is a positive instance of SCP. By construction, the b_{n+j}, for $1 \le j \le m$, are all coprime to each other and to all b_i for $1 \le i \le n$. A congruence class $(0 \bmod b_{n+j})$ will therefore overlap with any set of other overlapping congruence classes taken from A_λ by the Generalized Chinese Remainder Theorem. More specifically, if A' is a maximal subset of A such that there exists an x where $x \equiv a_i \pmod{b_i}$ for all $(a_i, b_i) \in A'$, then $A'_\lambda = A' \cup \{(0, b_{n+1}), \ldots (0, b_{n+m})\}$ is a maximal subset of A_λ such that there exists an x where $x \equiv a_i \pmod{b_i}$ for all $(a_i, b_i) \in A'_\lambda$. Since $|A'_\lambda| = |A'| + m$, it follows that $|A'_\lambda| \ge k + m = k_\lambda$ if and only if $|A'| \ge k$, and therefore (A_λ, k_λ) is a positive instance of λ-restricted SCP if and only if (A, k) is a positive instance of SCP. $\square$

We can now show strong coNP-hardness of the considered scheduling problem with any work-conserving scheduler by reducing from (the complement of) λ-restricted SCP.

Theorem 1. *The promise problem of determining if $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is schedulable by work-conserving scheduler $\mathcal{A}$ on a single preemptive processor is strongly* coNP-*hard, where Γ_{P} is a set of asynchronous periodic constrained-deadline tasks, Γ_{S} is a set of sporadic constrained-deadline tasks, and the promise is that Γ_{P} is $\mathcal{A}$-schedulable if executed without Γ_{S}.*
The strong coNP-*hardness holds even when $|\Gamma_{\mathrm{S}}| = 1$, the (single) sporadic task has an implicit deadline, and $\mathrm{U}(\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}) \le c$ for any constant c such that $0 < c < 1$.*

Proof. We will reduce from λ-restricted SCP with $\lambda = 1 - \frac{c}{3}$. Given an instance (A, k) of λ-restricted SCP we create an instance $(\Gamma_{\mathrm{P}}, \Gamma_{\mathrm{S}})$ of (the complement of the) scheduling problem as follows. Let

$$\Gamma_{\mathrm{P}} \overset{\mathrm{def}}{=} \{(\phi_i, C_i, D_i, T_i) \mid (a_i, b_i) \in A\},$$

where

$$\phi_i \overset{\mathrm{def}}{=} \sigma^2 a_i, \quad C_i \overset{\mathrm{def}}{=} \sigma, \quad D_i \overset{\mathrm{def}}{=} \sigma n, \quad T_i \overset{\mathrm{def}}{=} \sigma^2 b_i, \quad \sigma \overset{\mathrm{def}}{=} \left\lceil \frac{3n}{c} \right\rceil$$

Let

$$\Gamma_{\mathrm{S}} \overset{\mathrm{def}}{=} \{(C_{\mathrm{S}}, D_{\mathrm{S}}, T_{\mathrm{S}})\},$$

where

$$C_{\mathrm{S}} \overset{\mathrm{def}}{=} \sigma(n-k)+1, \quad D_{\mathrm{S}} \overset{\mathrm{def}}{=} \sigma n, \quad T_{\mathrm{S}} \overset{\mathrm{def}}{=} \sigma n.$$

First, we note that tasks in Γ_{P} only can release jobs at time points t that are multiples of σ^2 and they all have relative deadline $D_i = \sigma n < \sigma^2$, so jobs from tasks in Γ_{P} have overlapping scheduling windows with other jobs from Γ_{P} only if they are released at the same time t. But the number of jobs from Γ_{P} that can be released at any time point t is clearly no more than n, and all those jobs therefore fit into the interval $[t, t+\sigma n)$. Since all such jobs have the same absolute deadline, it does not matter which order they are executed in, and the promise that Γ_{P} is schedulable by itself by work-conserving scheduler $\mathcal{A}$ is fulfilled.
Next we show that if the sporadic task $(C_{\mathrm{S}}, D_{\mathrm{S}}, T_{\mathrm{S}})$ is added, the combined task set $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is $\mathcal{A}$-schedulable if and only if (A, k) is a negative instance of λ-restricted SCP. Consider first the case where (A, k) is a positive instance. Then there exists a subset $A' \subseteq A$ such that $|A'| \geq k$ and there exists an $x \in \mathbb{N}$, such that $x \equiv a_i \pmod{b_i}$ for all $(a_i, b_i) \in A'$, and therefore at least k tasks from Γ_{P} release a job at time $\sigma^2 x$. If also the sporadic task releases a job at the same time, then the total execution time demand in time interval $[\sigma^2 x, \sigma^2 x + \sigma n)$ is at least $\sigma k + C_{\mathrm{S}} = \sigma k + \sigma(n-k) + 1 > \sigma n$, and a deadline miss is unavoidable.
Consider instead the case where (A, k) is a negative instance. In this case, at any time point $t = \sigma^2 x$ for some $x \in \mathbb{N}$ when tasks in Γ_{P} can release jobs, there are less than k jobs from Γ_{P}, with a total execution time of at most $\sigma(k-1)$. Even if $\mathcal{A}$ chooses to execute all jobs from Γ_{P} before jobs from the sporadic task, there is not enough interference from jobs in Γ_{P} to cause the sporadic task to miss its deadline, since the slack of the sporadic task is $D_{\mathrm{S}} - C_{\mathrm{S}} = \sigma k - 1$, which is greater than $\sigma(k-1)$. Similarly, even if $\mathcal{A}$ chooses to always execute jobs from the sporadic task first, it does not offer enough interference for any job from Γ_{P} to miss its deadline. At most two jobs from the sporadic task can overlap the scheduling window $[\sigma^2 x, \sigma^2 x + \sigma n)$ of a job from Γ_{P}, but since scheduling windows of jobs from the periodic tasks are far apart and since $\mathcal{A}$ is work-conserving the total amount of execution of the sporadic task inside $[\sigma^2 x, \sigma^2 x + \sigma n)$ is never more than C_{S}. This leaves $\sigma n - C_{\mathrm{S}} = \sigma k - 1$ time units free for the jobs from Γ_{P} inside their scheduling window, which is enough no matter which order they are executed in since their total execution time is at most $\sigma(k-1)$.
Finally, we have

$$\mathrm{U}(\Gamma_{\mathrm{P}}) = \sum_{\tau_i \in \Gamma_{\mathrm{P}}} \frac{C_i}{T_i} = \sum_{\tau_i \in \Gamma_{\mathrm{P}}} \frac{\sigma}{\sigma^2 b_i} \leq \frac{\sigma n}{\sigma^2} \leq \frac{c}{3},$$

$$\mathrm{U}(\Gamma_{\mathrm{S}}) = \frac{C_{\mathrm{S}}}{T_{\mathrm{S}}} = \frac{\sigma(n-k)+1}{\sigma n} \leq \frac{\sigma(n-\lambda n)+1}{\sigma n} = 1 - \lambda + \frac{1}{\sigma n} \leq \frac{2c}{3},$$

and therefore total utilization is bounded by c. □

From Theorem 1 we see that the promise of the schedulability of the asynchronous periodic tasks in Γ_{P} does not help from a complexity-theoretic perspective: re-determining schedulability of a known-to-be schedulable Γ_{P} after adding just a single implicit-deadline task is of similar hardness as determining if Γ_{P} is schedulable in the first place. This is true even if utilization is bounded by an arbitrarily small constant and with any work-conserving scheduler.
On the positive side, with EDF the problem is in coNP.[7]

Lemma 2. *The decision problem of determining if $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is EDF-schedulable on a single preemptive processor is in* coNP*, where Γ_{P} is a set of asynchronous periodic constrained-deadline tasks, Γ_{S} is a set of sporadic constrained-deadline tasks.*

Proof. A counterexample to the EDF-schedulability of $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is a pair of numbers t_1 and t_2 for which the inequality in Eq. 1 does not hold. This counterexample is of polynomial size (since t_1 and t_2 are bounded by Eq. 4) and can be verified in polynomial time. □

With EDF scheduling we can therefore reduce the schedulability problem to (the complement of) ILP in polynomial time and use an ILP solver to answer it. This works for the decision problem and therefore trivially for the promise problem as well.
However, for Fixed-Priority (FP) scheduling this is unlikely to be possible even for the promise problem.

Lemma 3. *The promise problem of determining if $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ is FP-schedulable on a single preemptive processor is* NP*-hard, where Γ_{P} is a set of asynchronous periodic constrained-deadline tasks, Γ_{S} is a set of sporadic constrained-deadline tasks, and the promise is that Γ_{P} is FP-schedulable if executed without Γ_{S}.*

Proof. A trivial reduction from the ordinary FP-schedulability problem for sporadic tasks simply sets $\Gamma_{\mathrm{P}} = \emptyset$ and copies the input instance to Γ_{S}. Since FP-schedulability for sporadic tasks is NP-complete [24], our promise problem is NP-hard. □

Since the promise problem with FP scheduling is both coNP-hard (Theorem 1) and NP-hard (Lemma 3) it cannot be reduced to ILP (or to the complement of ILP) in polynomial time unless NP = coNP and the polynomial hierarchy collapses.

[7] Technically, since coNP is a class of *decision* problems, it is the decision version of the problem (where we have no promise) that is in coNP. If the decision version is in coNP, the promise problem is trivially in the corresponding class promise-coNP.

4 Exact Feasibility Tests Based on PDA

It follows from [13, 14] that we can decide EDF-schedulability of $\Gamma = \Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ using the *processor demand analysis (PDA)* by verifying that the inequality

$$\sum_{\tau_i \in \Gamma_{\mathrm{P}}} \mathrm{dbf}_{\mathrm{P}}(\tau_i, t_1, t_2) + \sum_{\tau_i \in \Gamma_{\mathrm{S}}} \mathrm{dbf}_{\mathrm{S}}(\tau_i, t_2 - t_1) \;\leq\; t_2 - t_1 \tag{1}$$

holds for all $t_1 < t_2$, where

$$\mathrm{dbf}_{\mathrm{P}}(\tau_i, t_1, t_2) \overset{\mathrm{def}}{=} C_i \cdot \max\left(0, \left\lfloor \frac{t_2 - \phi_i - D_i}{T_i} \right\rfloor - \max\left(0, \left\lceil \frac{t_1 - \phi_i}{T_i} \right\rceil\right) + 1\right), \tag{2}$$

$$\mathrm{dbf}_{\mathrm{S}}(\tau_i, t) \overset{\mathrm{def}}{=} C_i \cdot \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \tag{3}$$

are the standard demand bound functions for periodic and sporadic tasks, respectively.
As shown in [13], the values of t_1 and t_2 can be confined to release times and absolute deadlines of tasks, respectively. However, since the synchronous arrival time of sporadic tasks is not known a priori, Eq.1 accumulates their maximum requested demand within the interval $[t_1, t_2]$ by assuming they arrive synchronously at t_1. Consequently, t_1 is restricted to the release times of periodic tasks only, whereas t_2 can be the absolute deadline of any task in the combined task set where for the sporadic tasks their absolute deadlines are those resulting from their synchronous arrival sequence starting at t_1.
For plain periodic tasks sets (i.e., not mixed with sporadic tasks), it was also shown in [13] that the ranges on the values of t_1 and t_2 to consider for the PDA test can be constrained as

$$0 \leq t_1 < t_2 \leq \max_{\tau_i \in \Gamma_{\mathrm{P}}} (\phi_i) + 2\,\mathrm{HP}(\Gamma_{\mathrm{P}}), \tag{4}$$

which is commonly repeated in the literature (see, e.g., [13,25–27]) – here $\mathrm{HP}(\Gamma_{\mathrm{P}})$ denotes the hyper-period of the periodic taskset Γ_{P}. However, as the following theorem shows, it is in fact possible to reduce those ranges significantly, if total utilization is below 1, even for our more general problem with mixed task sets.

Theorem 2. *A combined constrained-deadline task set $\Gamma = \Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ of N tasks, where Γ_{P} is a set of asynchronous periodic tasks, Γ_{S} is a set of sporadic tasks and $\mathrm{U}(\Gamma) < 1$, is EDF-schedulable on a single preemptive processor if and only* *if the inequality in Eq. 1 holds for all* $t_1 < t_2$*, where*[8]

$$\max_{\tau_i \in \Gamma_\mathrm{P}} (\phi_i) \leq t_1 < \max_{\tau_i \in \Gamma_\mathrm{P}} (\phi_i) + \mathrm{HP}(\Gamma_\mathrm{P}) \tag{5}$$

$$t_2 < t_1 + \mathrm{B}(\Gamma) \tag{6}$$

$$\mathrm{B}(\Gamma) \overset{def}{=} \frac{\sum_{\tau_i \in \Gamma} (T_i - D_i) U_i}{1 - \mathrm{U}(\Gamma)} \tag{7}$$

Proof. Again, it follows from [13, 14] that Γ is EDF-schedulable if and only if Eq. 1 holds for all $t_1 < t_2$. We will show that the reduced ranges for t_1 and t_2 given in Eqs. 5 and 6 are sufficient to consider.

First, we note that the range for t_1 in Eq. 5 is the length of one hyper-period of the periodic tasks, $\mathrm{HP}(\Gamma_\mathrm{P})$, and that by starting at the largest offset $\max_{\tau_i \in \Gamma_\mathrm{P}} (\phi_i)$ the interval for t_1 will include all possible patterns of job releases from the periodic tasks, which are then repeated in the same way in the next interval of length $\mathrm{HP}(\Gamma_\mathrm{P})$ and so on. It follows that for any $t'_1, \delta \in \mathbb{N}$, there exists $t_1 \in [\max_{\tau_i \in \Gamma_\mathrm{P}} (\phi_i), \max_{\tau_i \in \Gamma_\mathrm{P}} (\phi_i) + \mathrm{HP}(\Gamma_\mathrm{P}))$ such that $\sum_{\tau_i \in \Gamma_\mathrm{P}} \mathrm{dbf_P}(\tau_i, t'_1, t'_1 + \delta) = \sum_{\tau_i \in \Gamma_\mathrm{P}} \mathrm{dbf_P}(\tau_i, t_1, t_1 + \delta)$. Trivially, we also have $\sum_{\tau_i \in \Gamma_\mathrm{S}} \mathrm{dbf_S}(\tau_i, t'_1 + \delta - t'_1) = \sum_{\tau_i \in \Gamma_\mathrm{S}} \mathrm{dbf_S}(\tau_i, t_1 + \delta - t_1)$ for the sporadic tasks. Hence, if there are any values of t'_1 and t'_2, where t'_1 does not fall within the constraints of Eq. 5 and for which the inequality in Eq. 1 does not hold, then the inequality in Eq. 1 also does not hold for some other values of t_1 and t_2 where t_1 is constrained as in Eq. 5.

Second, it follows directly from Eqs. 2 and 3 that $\mathrm{dbf_P}(\tau_i, t_1, t_2) \leq \mathrm{dbf_S}(\tau_i, t_2 - t_1)$ for any periodic task τ_i and any $t_1 < t_2$. (Note that we are here applying $\mathrm{dbf_S}$ to a periodic task. While $\mathrm{dbf_S}$ is *not* an exact demand bound function for a periodic task, it is still a well-defined function over three of the task's parameters.) From this it follows that for our combined task set $\Gamma = \Gamma_\mathrm{P} \cup \Gamma_\mathrm{S}$ and for any $t_1 < t_2$ we have

$$\sum_{\tau_i \in \Gamma_\mathrm{P}} \mathrm{dbf_P}(\tau_i, t_1, t_2) + \sum_{\tau_i \in \Gamma_\mathrm{S}} \mathrm{dbf_S}(\tau_i, t_2 - t_1) \leq \sum_{\tau_i \in \Gamma} \mathrm{dbf_S}(\tau_i, t_2 - t_1) \tag{8}$$

It was shown in [14, 28] that for any Γ', where $\mathrm{U}(\Gamma') < 1$, if $\sum_{\tau_i \in \Gamma'} \mathrm{dbf_S}(\tau_i, t) > t$, then we must also have $t < \mathrm{B}(\Gamma')$. Combining this with Eq. 8, it follows that if $\sum_{\tau_i \in \Gamma_\mathrm{P}} \mathrm{dbf_P}(\tau_i, t_1, t_2) + \sum_{\tau_i \in \Gamma_\mathrm{S}} \mathrm{dbf_S}(\tau_i, t_2 - t_1) > t_2 - t_1$, then $t_2 - t_1 < \mathrm{B}(\Gamma)$. This matches the bound on t_2 in Eq. 6 and completes the proof. $\square$

The commonly-used ranges in Eq. 4 give $\Theta(\mathrm{HP}(\Gamma)^2)$ combinations of t_1 and t_2, while the ranges in Eqs. 5 and 6 give only $\Theta(\mathrm{HP}(\Gamma_\mathrm{P})\,\mathrm{B}(\Gamma))$ combinations. The latter would typically be much smaller, which could improve the running time of checking schedulablity greatly.

Theorem 2 is also applicable when $\Gamma_\mathrm{S} = \emptyset$, and so as a **corollary** the reduced ranges of t_1 and t_2 can be used instead of the ranges in Eq. 4 when checking schedulability of plain asynchronous periodic task sets as well.

[8] In the corner case where $\Gamma_\mathrm{P} = \emptyset$, and Eq. 5 would be undefined, simply let $t_1 = 0$ instead. We note that in this corner case the proposed test simply reduces to the standard PDA test for sporadic tasks described in [14].

4.1 Pre-Processing for a Runtime-Tailored Feasibility Test

Despite reducing the values of t_1 and t_2 to consider, the runtime complexity for the test in Theorem 2 is still dominated by the periodic task set for which separate combinations of t_1 and t_2 matter, and not just their difference.

However, in the setting considered in this paper, where Γ_{P} is fixed and we want to efficiently determine if adding a given Γ_{S} will make the system unschedulable, it is possible to shift much of this complexity to pre-runtime by pre-processing the sum $\sum_{\tau_i \in \Gamma_{\mathrm{P}}} \mathrm{dbf}_{\mathrm{P}}(\tau_i, t_1, t_2)$ in Eq. 1 as follows.

For each $t \in \{0, 1, \ldots, \mathrm{B}(\Gamma) - 1\}$, pre-compute values $\mathbf{dbf}_{\Gamma_{\mathrm{P}}}(t)$ as

$$\mathbf{dbf}_{\Gamma_{\mathrm{P}}}(t) \;\leftarrow\; \max_{t_1 \in R} \left(\sum_{\tau_i \in \Gamma_{\mathrm{P}}} \mathrm{dbf}_{\mathrm{P}}(\tau_i, t_1, t_1 + t) \right), \tag{9}$$

where $R = [\max_{\tau_i \in \Gamma_{\mathrm{P}}}(\phi_i), \max_{\tau_i \in \Gamma_{\mathrm{P}}}(\phi_i) + \mathrm{HP}(\Gamma_{\mathrm{P}}))$ is the range of values for t_1 from Eq. 5. Then, Theorem 3 describes our runtime-tailored feasibility test (referred to as *PDA** in Section 6).

Theorem 3. *A combined constrained-deadline task set $\Gamma = \Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$, where Γ_{P} is a set of asynchronous periodic tasks, Γ_{S} is a set of sporadic tasks and $\mathrm{U}(\Gamma) < 1$, is EDF-schedulable on a single preemptive processor if and only if the inequality*

$$\mathbf{dbf}_{\Gamma_{\mathrm{P}}}(t) + \sum_{\tau_i \in \Gamma_{\mathrm{S}}} \mathrm{dbf}_{\mathrm{S}}(\tau_i, t) \;\leq\; t \tag{10}$$

holds for all $0 < t < \mathrm{B}(\Gamma)$, where $\mathrm{B}(\Gamma)$ is calculated as per Eq. 7

Proof. Let t_1 be any release time of a periodic task, and consider any interval $[t_1, t_2]$ of length $t = t_2 - t_1$, with $0 < t < \mathrm{B}(\Gamma)$. Since sporadic tasks' demand is maximized in $[t_1, t_2]$ if they all arrive synchronously at t_1, their total demand depends only on t and equals $\sum_{\tau_i \in \Gamma_{\mathrm{S}}} \mathrm{dbf}_{\mathrm{S}}(\tau_i, t)$. In contrast, periodic tasks' demand $\mathrm{dbf}_{\mathrm{P}}(\tau_i, t_1, t_2)$ varies with the interval alignment, i.e., two intervals of the same length t but with different starting points t_1 can yield different demand. However, $\mathbf{dbf}_{\Gamma_{\mathrm{P}}}(t)$ is exactly the maximum of periodic tasks' demand over all intervals $[t_1, t_2]$ of length t, and so if the inequality in Eq. 1 fails for some values t_1 and t_2, then the inequality in Eq. 10 will also fail considering $t = t_2 - t_1$ and vice versa. $\square$

Theorem 3 allows us to check EDF-schedulability of $\Gamma_{\mathrm{P}} \cup \Gamma_{\mathrm{S}}$ for arbitrary Γ_{S}, having pre-computed the more costly analysis for Γ_{P}. The issue with this is that bound $\mathrm{B}(\Gamma)$ depends on the particular Γ_{S}, which we by assumption do not know at the time of pre-computing $\mathbf{dbf}_{\Gamma_{\mathrm{P}}}$. But if we reasonably assume upper bounds on allowed values of $\mathrm{U}(\Gamma)$ and of $\max_{\tau_i \in \Gamma}(T_i - D_i)$, then we can directly calculate an upper bound on $\mathrm{B}(\Gamma)$ to be used for the pre-processing. When evaluating Eq. 10 at runtime the exact $\mathrm{B}(\Gamma)$ is used instead.

Indeed, with a predefined constant upper bound on allowed $\mathrm{U}(\Gamma)$, say $\mathrm{U}(\Gamma) \leq 0.99$, evaluating the schedulability with Eq. 10 takes only pseudo-polynomial time (not counting pre-computing $\mathbf{dbf}_{\Gamma_\mathrm{P}}$). Storing the values of $\mathbf{dbf}_{\Gamma_\mathrm{P}}$ takes pseudo-polynomial space though. In contrast, evaluating the test in Theorem 2 is not in pseudo-polynomial time, but it requires only polynomial space.

Note that it suffices to check only those values of t at which the left-hand side of Eq. 10 is discontinuous. The discontinuity points for $\mathbf{dbf}_{\Gamma_\mathrm{P}}$ are computed during pre-processing[9]. For dbf_S, the discontinuities occur at the absolute deadlines of sporadic tasks from a synchronous arrival sequence in the interval $[0, \mathrm{B}(\Gamma)]$.

To further optimize runtime performance, we apply the *quick convergence processor demand analysis (QPA)* technique [19], which enables the safe skipping of certain values of t using a simple backward iterative procedure. We adapt QPA to our problem by setting the $h(t)$ function from Theorem 5 in [19] to be equal to the left-hand side of Eq. 10, and substituting the list of absolute deadlines with the list of discontinuity points. We refer to this adapted test, which combines QPA with preprocessed values of $\mathbf{dbf}_{\Gamma_\mathrm{P}}$, as *QPDA**. The corresponding pseudocode is provided in Algorithm 1.

Algorithm 1 Pseudocode for QPDA*

```
1: Define $h(t) = \mathbf{dbf}_{\Gamma_\mathrm{P}}(t) + \sum_{\tau_i \in \Gamma_\mathrm{S}} \mathrm{dbf}_\mathrm{S}(\tau_i, t)$   (left-hand side of Eq. 10)
2: Define $\mathcal{T}_f$ = ordered list of unique discontinuity points of $h(t)$
3: $t_\mathrm{min} = \min\{t_i \mid t_i \in \mathcal{T}_f\}$
4: $t = \max\{t_i \mid t_i \in \mathcal{T}_f \wedge t_i < \mathrm{B}(\Gamma)\}$
5: while $h(t) \leq t$ and $h(t) > t_\mathrm{min}$ do
6:     $t = \begin{cases} h(t), & \text{if } h(t) < t \\ \max\{t_i \in \mathcal{T}_f \mid t_i < t\}, & \text{otherwise} \end{cases}$
7: end while
8: Output: Task set is schedulable if $h(t) \leq t_\mathrm{min}$; otherwise, not schedulable
```

5 Exact Feasibility Test Based on ILP

From Section 4, the EDF-schedulability of the task set $\Gamma = \Gamma_\mathrm{P} \cup \Gamma_\mathrm{S}$ can be determined using Eqs 1–3. In this section, we show how to reformulate these inequalities into an Integer Linear Programming (ILP) model by expressing them as linear constraints with integer variables.

Reformulating dbf_P **.** The demand-bound function for periodic tasks in Eq. 2 includes floor, ceiling, and max functions, which must be reformulated using integer variables and linear constraints to ensure compatibility with the ILP

[9] In our evaluation in Section 6, we simply store the pre-processed values as pairs $(t, \mathbf{dbf}_{\Gamma_\mathrm{P}}(t))$ for only the points t where $\mathbf{dbf}_{\Gamma_\mathrm{P}}(t)$ is discontinuous.

framework. First we encode the floor and ceiling parts of Eq. 2 as

$$\begin{aligned} \frac{t_2 - \phi_i - D_i}{T_i} \;&\geq\; X_{i,1} \;\geq\; \frac{t_2 - \phi_i - D_i}{T_i} - 1 + \epsilon \\ \frac{t_1 - \phi_i}{T_i} \;&\leq\; X_{i,2} \;\leq\; \frac{t_1 - \phi_i}{T_i} + 1 - \epsilon \qquad (X_{i,1}, X_{i,2} \in \mathbb{N}), \end{aligned}$$

where $\epsilon < 1/\max_{\tau_i \in \mathit{\Gamma}}(T_i)$ is some sufficiently small constant.
By substituting the new variables, $X_{i,1}$ and $X_{i,2}$, we have

$$\mathrm{dbf_P}(\tau_i, t_1, t_2) \;=\; C_i \cdot \max(0, X_{i,1} - \max(0, X_{i,2}) + 1)$$

Now we consider the max functions. The inner max function $\max(0, X_{i,2})$ can be ignored assuming $t_1 \geq 0$, and $\phi_i < T_i$ for all $\tau_i \in \mathit{\Gamma}_\mathrm{P}$. To reformulate the outer max function, we use an auxiliary variable $X_{i,3}$

$$X_{i,3} \;=\; \max(0, X_{i,1} - X_{i,2} + 1)$$

We further need a binary variable to handle the two cases of the max function and a large positive constant M_i, which we give the value: $M_i = \frac{C_i}{T_i} \cdot M$ where $M = \max_{\tau_i \in \mathit{\Gamma}_\mathrm{P}}(\phi_i) + 2\,\mathrm{HP}(\mathit{\Gamma})$. The constraints for $X_{i,3}$ are as follows:

$$\begin{aligned} X_{i,3} \;&\geq\; 0 \\ X_{i,3} \;&\geq\; X_{i,1} - X_{i,2} + 1 \\ X_{i,3} \;&\leq\; X_{i,1} - X_{i,2} + 1 + M_i \cdot (1 - B_i) \\ X_{i,3} \;&\leq\; M_i \cdot B_i \qquad (B_i \in \{0, 1\}), \end{aligned}$$

Reformulating $\mathrm{dbf_S}$ **.** The standard demand-bound function for sporadic tasks in Eq. 3 can be similarly reformulated. We start by expressing the floor function using linear constraints with integer variables:

$$\frac{t - D_i}{T_i} \;\geq\; X_{i,1} \;\geq\; \frac{t - D_i}{T_i} - 1 + \epsilon \qquad (X_{i,1} \in \mathbb{N}),$$

By substituting the new variable, $X_{i,1}$, we have

$$\mathrm{dbf_S}(\tau_i, t) \;=\; C_i \cdot \max(0, X_{i,1} + 1)$$

The max function can be ignored assuming $t = t_2 - t_1 > 0$ and $D_i \leq T_i\ \forall \tau_i \in \mathit{\Gamma}_\mathrm{S}$. Now that all constraints for the demand bound functions are encoded, the following captures the condition of Theorem 2.

$$\begin{aligned} &\sum_{\tau_i \in \mathit{\Gamma}_\mathrm{P}} C_i \cdot X_{i,3} + \sum_{\tau_i \in \mathit{\Gamma}_\mathrm{S}} C_i \cdot (X_{i,1} + 1) \;\geq\; t_2 - t_1 + 1 \\ &\max_{\tau_i \in \mathit{\Gamma}_\mathrm{P}}(\phi_i) \leq t_1 \leq \max_{\tau_i \in \mathit{\Gamma}_\mathrm{P}}(\phi_i) + \mathrm{HP}(\mathit{\Gamma}_\mathrm{P}) - 1, \\ &\qquad\qquad t_2 \leq t_1 + \mathrm{B}(\mathit{\Gamma}) - 1, \qquad (t_1, t_2 \in \mathbb{N}), \end{aligned}$$

Note that a *feasibility* formulation is unsuitable here, since the solver only seeks one satisfying assignment rather than proving that the constraint hold for all time points. In contrast, the *in-feasibility* formulation allow the ILP solver to terminate as soon as it finds a single solution, which serves as a concrete proof of unschedulability. If the ILP solver returns Yes, it means there exist time points t_1 and t_2 for which Eq. 1 does not hold, and the task set is not EDF-schedulable. Conversely, if the solver returns No, then Eq. 1 holds for all such time points, confirming that the task set is EDF-schedulable.

6 Experimental Evaluation

We structure our evaluation into two distinct experiments with the main objective of identifying the most efficient feasibility tests for runtime deployment. The first experiment focuses solely on comparing exact feasibility tests based on their runtime performance, as they produce equivalent results in terms of feasibility. The tests included in this comparison are:

***PDA*:** the processor demand analysis described in Theorem 2,
***PDA**:** the PDA variant with the pre-processing step in Theorem 3 and stored array of pairs $(t, \mathbf{dbf}_{\Gamma_\mathrm{P}}(t))$ for t where $\mathbf{dbf}_{\Gamma_\mathrm{P}}$ in Eq. 10 is discontinuous,[10]
***QPDA**:** the *PDA** variant with the QPA technique from [19].
***ILP*:** the ILP model from Section 5 implemented in the *Gurobi* [29] solver.[11]

The second experiment evaluates our exact feasibility tests, *PDA*, *PDA** and *QPDA**, in comparison to the currently available EDF-based feasibility tests, focusing on both schedulability and runtime performance. A comparison with the offline schedule modification methods in Section 2 is left for future work. In particular, we consider Slot-Shifting's offline admission test for sporadic tasks in time-triggered systems [10], which we refer to here as *OTSS*. We assume that the parameters of the generated periodic tasks align with the target execution windows determined during the offline phase of Slot-Shifting. The available spare capacity is then computed based on ALAP execution of periodic tasks following EDF, and using a fixed time slot size, as outlined in [10].
All feasibility tests were implemented in Python. While the Gurobi solver leverage Python as an interface, its core functionalities are highly optimized and written in C. Implementing all tests in C or some other optimized compiled language would arguably provide a more balanced comparison, potentially yielding performance improvements of up to 25 to 30 times for the other tests, as indicated by previous studies on compiled language efficiency [31, 32]. However, to prioritize rapid development, we chose to forgo a native C implementation,

[10] The reported running times for *PDA** and *QPDA** do not include the time for preprocessing phase, whose runtime performance is comparable to that of the original PDA.
[11] We have also experimented with the *SCIP* solver [30], but found that *Gurobi* consistently outperforms it for the experiments in this paper.

and instead mitigate the runtime overhead by running the tests implemented in Python using PyPy's Just-in-Time (JiT) compiler [33][12].

We conducted all experiments on an Intel Core™ i7-9700 CPU @ 3.00 GHz machine with 24 GB RAM, running Ubuntu 24.04.2 LTS. We confined the execution of the tests to a single CPU core and ran them using PyPy version 7.3.19, with the exception of the Gurobi ILP solver (version 12.0.2), which we ran with the CPython interpreter due to its current incompatibility with PyPy.

For all experiments, we restrict task parameters to natural numbers. Moreover, for any task τ_k, its deadline D_k is uniformly drawn from the upper half of the interval $[C_k, T_k]$, whereas its offset, if periodic, is uniformly drawn from the interval $[0, T_k)$.

6.1 Experiment 1: Synthetic Task Sets

For this experiment, we generated synthetic task sets using *Roger Stafford's randfixedsum* algorithm [34]. For that, we extended the task generator from [35] to create mixed asynchronous periodic and synchronous sporadic task sets with specific hyper-periods. The *randfixedsum* algorithm produces rational wcet values for tasks in order to meet a predefined target utilization. Hence, to constrain wcet values to natural numbers and to prevent the occurrence of zero-valued wcet due to rounding, we begin by assigning each task an initial wcet of 1. We then generate rational wcet values using *randfixedsum*, round them to the nearest integer, and add them to the initial values. This process is repeated until the resulting total utilization matches the target within a 2% tolerance threshold.

Since input size directly affects the runtime cost of feasibility tests, we created multiple one-parameter-at-a-time test suites. In each test suite, we vary one input parameter and fix the rest.

Test Suite 1: Runtime vs. Task Set Size. In this test suite, we vary the number of tasks per task set, from 5 to 100 in increments of 5. For each configuration, we generate 100 task sets, resulting in a total of 2000 task sets. Each task set consists of 60% periodic tasks and 40% sporadic tasks. For all task sets, we set the target utilization to 85% and the hyper-period to 1000. To preserve the generality of the results, the task set generation deliberately avoids configuring all periods as pseudo-harmonic. Therefore, we uniformly select periods from the set: {4, 8, 10, 20, 25, 40, 50, 100, 125, 200, 250}.

Figure 1 shows the runtime (logarithmic y-axis) for all feasibility tests as function of task set size (x-axis). The runtime performance of both *PDA* and *ILP* is comparable and grows considerably with task set size. However, thanks to the pre-processing step, *PDA** achieves a two to three orders of magnitude improvement over *ILP* despite its pure Python implementation. Furthermore, *QPDA** is around 2.3 times faster than *PDA** on average. However, in a few outlier cases within the test suite, *QPDA** exhibits lower performance compared to *PDA**.

[12] We observed an average runtime improvement of approximately 6x relative to executing PDA using the CPython interpreter.

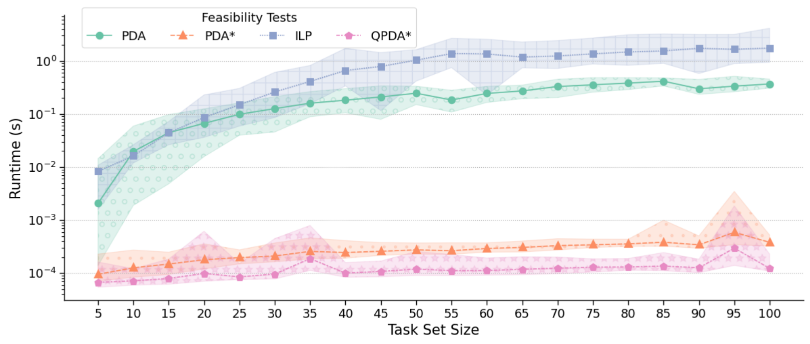

Fig. 1: Mean runtime of exact feasibility tests as a function of task set size; hatched areas show min-max observed values.

Test Suite 2: Runtime vs. Hyper-period Length. In this test suite, we vary the length of the combined task set hyper-period across multiple configurations. For each value in the sets $\{0.5 \times 10^3, 10^3, 5 \times 10^3, 10^4, 2.5 \times 10^4\}$ and $\{5 \times 10^4, 10^5, 5 \times 10^5\}$, we generate 100 and 10 task sets, respectively (530 sets in total). Each task set contains 30 tasks, composed of 60% periodic and 40% sporadic tasks, with a combined utilization of 85%.

Figure 2 shows the runtime (logarithmic y-axis) for all feasibility tests as a function of hyper-period length (logarithmic x-axis). The runtime of *PDA* grows fastest with the hyper-period, whereas *ILP* maintains relatively stable runtime across different hyper-period lengths. This is expected, as *PDA* scales with the number of release times and deadlines within $\mathrm{HP}(\mathit{\Gamma}_{\mathrm{P}}) \times \mathrm{B}(\mathit{\Gamma})$ (see Theorem 2), whereas the *ILP* solver identifies solutions that satisfy the imposed model constraints without direct dependency on hyper-period length. Thanks to the pre-processing step, *PDA** and *QPDA** no longer rely on the length of $\mathrm{HP}(\mathit{\Gamma}_{\mathrm{P}})$. Hence, they are expected to be at least $\mathrm{HP}(\mathit{\Gamma}_{\mathrm{P}})$ times faster than *PDA*. Both *PDA** and *QPDA** outperform ILP by approximately one to two orders of magnitude. Notably, *QPDA** performs comparably to *PDA** at shorter hyper-periods and exceeds its performance at longer hyper-periods, thanks to its deadline skipping optimization.

Memory footprint for (Q)PDA*. As demonstrated in the previous test suites, the pre-processing step employed by *PDA** and *QPDA** significantly improves runtime performance. However, as noted in Section 4, storing the demand from periodic tasks increases the memory footprint of the *PDA* test from polynomial to pseudo-polynomial complexity. Therefore, to further assess its suitability for runtime deployment, we calculate the memory requirement for storing the demand from periodic tasks considering the previous test suites and assuming that this demand is encoded in an array of integer pairs. Each pair $(t, \mathbf{dbf}_{\mathit{\Gamma}_{\mathrm{P}}}(t))$ holds the value of $\mathbf{dbf}_{\mathit{\Gamma}_{\mathrm{P}}}$ (see Eq. 10) at a discontinuity time point t, i.e., when $\mathbf{dbf}_{\mathit{\Gamma}_{\mathrm{P}}}$ changes value.

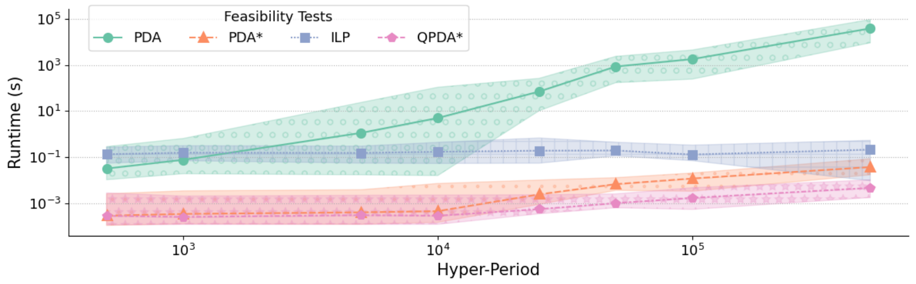

Fig. 2: Mean runtime of exact feasibility tests as a function of hyper-period length; hatched areas show min-max observed values.

Note that the results shown here are based on the a priori knowledge of the bound $\mathrm{B}(\Gamma)$. However, as discussed in Sec. 4, when the sporadic task set is not known offline, then this bound must be overestimated, which may lead to a larger memory footprint.

In Test Suite 1, the memory requirement ranged from just a few bytes to approximately 1.5 KB. By contrast, in Test Suite 2 the memory requirement exhibited more substantial increases as the hyper-period grew. Starting at less than 1 KB for $\mathrm{HP}(\Gamma) = 500$, the maximum observed memory requirement increased by roughly 2.5 KB per 1000 units of hyper-period. This trend continued up to $\mathrm{HP}(\Gamma) = 500{,}000$, where the maximum observed memory requirement dropped, with a maximum observed value of 600 KB. Note that when feasible, opting for smaller integer types can reduce memory usage, e.g., using unsigned 16-bit integers may cut the memory footprint in half.

We did not observe a clear correlation between the maximum memory requirement and the task set size. Although hyper-period length appears to have a stronger influence, the relationship is not straightforward. The memory required for storing the pre-processed demand is primarily driven by the length of the interval $[0, B(\Gamma)]$ and the number of discontinuities in $\mathbf{dbf}_{\Gamma_\mathrm{P}}(t)$ within that interval. These factors tend to grow with longer hyper-periods, influenced by task utilization and the difference between task periods and deadlines (see Eq. 7).

It is worth noting that many real-time systems operate with hyper-periods ranging from a few hundred to a few thousand time units. These systems typically have sufficient memory resources to accommodate the overhead, positioning *PDA** and *QPDA** as a viable feasibility test for runtime deployment.

6.2 Experiment 2: Real-World Automotive Benchmarks

For this experiment, we compare EDF-based feasibility tests based on task sets generated according to the *Real-World Automotive Benchmarks* in [36]. For that, we developed a task-set generator that selects periods from the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}ms$ using the distribution in Table III of [36], excluding angle-synchronous activation and re-normalizing the remaining probabilities. For each chosen period, the generator samples runnables that satisfy the constraints in Tables IV and V of [36], then sums their execution times to form a single task.

Furthermore, since $OTSS$ is based on a time-triggered scheduler, we evaluate it considering a slot-size of 100 μs and scale the task set parameters accordingly (referred to as $OTSS100$). To illustrate the trade-off between runtime efficiency and schedulability, we also present results for $OTSS$ under a 1 μs slot-size configuration ($OTSS$), although such fine granularity is generally impractical in time-triggered systems due to the substantial scheduling overhead [37].

For all generated task sets, we fixed the utilization of periodic tasks at 30% and varied the utilization of sporadic tasks from 20% to 70% in 10% increments. We generate for each configuration 100 feasible task sets (600 in total), each consisting of 15 periodic and 15 sporadic tasks. Note that since all tests in this experiment are purely implemented in Python, the runtime performance comparison is more accurate compared to the first experiment.

Figures 3 and 4 illustrate the runtime performance (logarithmic y-axis) and schedulability success rates, respectively, for the feasibility tests across varying task set utilization values. As utilization increases, the schedulability of *OTSS* declines sharply. On average, *OTSS100* achieves a schedulability rate of only 12.76%, approximately 4% lower than that of *OTSS1*. Increasing the slot size from 1 to 100μs results in a runtime reduction of one to two orders of magnitude, with higher utilization values leading to earlier violations of $OTSS$'s feasibility checks. Compared to the exact tests, *OTSS100* performs better than PDA in terms of runtime, but is consistently outperformed by our tests with preprocessing *PDA** and *QPDA**, which achieve two to three order of magnitude better performance.

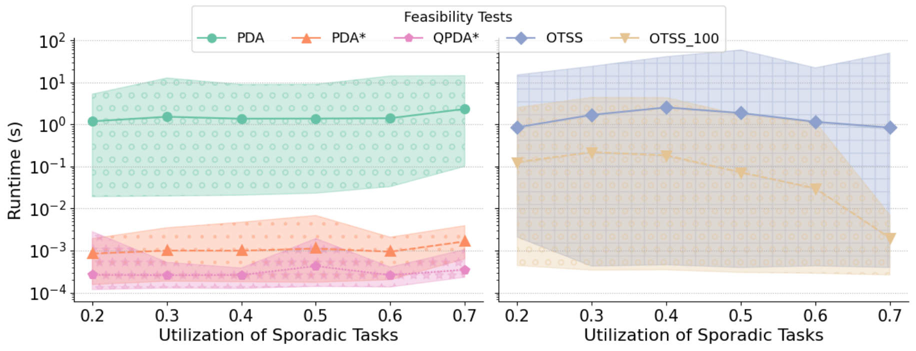

Fig. 3: Mean runtime of feasibility tests as a function of task set utilization; hatched areas show min-max observed values. The utilization of periodic tasks is fixed at 30%.

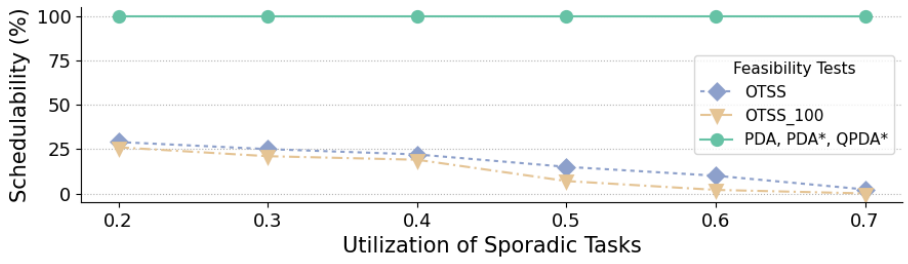

Fig. 4: Schedulability success percentage of feasibility tests as a function of task set utilization. The utilization of periodic tasks is fixed at 30%

Note that even if *OTSS* were deployed for runtime use, it would have comparable memory requirements to *QPDA**, as it needs to store Slot-Shifting's capacity intervals throughout the entire TT schedule. Consequently, *QPDA** emerges as the most suitable option for runtime deployment, offering superior overall performance in terms of both runtime efficiency and schedulability.
It is worth noting that PDA can be readily applied as an offline test for Slot-Shifting and other flexible time-triggered schedulers with dynamic slot assignment techniques [12], provided that task parameters are adjusted to be multiple of the time slot size.[13]

7 Context and Conclusions

In this paper, we have reported on our investigations of a scheduling problem that emerged within a long-term ongoing effort to enhance the applicability of the time-triggered scheduling framework by incorporating runtime flexibility. Given a system of asynchronous periodic tasks that is known to be feasible upon a single preemptive processor, this scheduling problem asks whether feasibility is preserved if a collection of sporadic tasks is added. We showed that this feasibility problem is computationally intractable (strongly coNP-hard) for any work-conserving scheduler, even when the sporadic task set contains only a single implicit-deadline task and the cumulative utilization of the entire task set (periodic and sporadic) is arbitrarily low. We provided, via a polynomial-time reduction, an ILP formulation of the (complement) scheduling problem under EDF, but showed that it is unlikely to be possible to make any such a formulation in polynomial time under fixed-priority scheduling.

[13] Note that we also explored a modified variant of *OTSS*, based on the notion of simulation interval [38], that eliminates its pessimism (effectively making the test exact) by assuming as soon as possible execution of periodic tasks and choosing the release times of periodic tasks as global offsets for the synchronous arrivals of sporadic tasks. Despite this refinement, the test consistently underperformed in terms of runtime compared to the original *OTSS*. As a result, and due to space limitations, we chose not to include it.

To support runtime deployment, we introduced *PDA**, an exact feasibility test that precomputes and summarizes the computational demand of the periodic tasks in a pre-runtime phase, thereby improving runtime efficiency (at the cost of increased memory usage). We further enhanced *PDA**'s runtime performance by employing the quick convergence processor demand analysis technique, obtaining a new test *QPDA**. Our experiments with synthetic task sets and automotive benchmarks demonstrate that *QPDA** has the best performance in terms of runtime and schedulability with manageable memory footprint.
Future work includes extending *QPDA** to support more task models and generalizing the analysis to resource-sharing platforms, and to multicore and many-core architectures.

References

1. C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
2. Joseph Y.-T Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.
3. H. Kopetz and G. Grunsteidl. TTP - a time-triggered protocol for fault-tolerant real-time systems. In *FTCS-23 The Twenty-Third International Symposium on Fault-Tolerant Computing*, pages 524–533, June 1993.
4. Hermann Kopetz and Günter Grünsteidl. TTP – a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, January 1994.
5. H. Kopetz and G. Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112–126, Jan 2003.
6. Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer, New York, 2011.
7. Gerhard Fohler. *Predictably Flexible Real-Time Scheduling*, pages 207–221. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
8. Damir Isovic and Gerhard Fohler. Online handling of hard aperiodic tasks in time-triggered systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, York, UK, June 1999. IEEE Computer Society Press.
9. Damir Isovic and Gerhard Fohler. Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints. In *Proceedings of the Real-Time Systems Symposium*, Orlando, FL, November 2000. IEEE Computer Society Press.
10. Damir Isovic and Gerhard Fohler. Handling mixed sets of tasks in combined offline and online scheduled real-time systems. *Real-Time Syst.*, 43(3):296–325, nov 2009.
11. G. Fohler. Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 152–161, 1995.
12. Mohammad Ibrahim Alkoudsi and Gerhard Fohler. Scheduling dynamic task-sets in time-triggered real-time systems. In *Proceedings of the 32nd International Conference on Real-Time Networks and Systems*, RTNS '24, page 48–58, New York, NY, USA, 2025. Association for Computing Machinery.
13. S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:301–324, 1990.

14. S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.
15. Michael Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.
16. P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-complete. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 281–286, 2015.
17. P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks remains coNP-complete under bounded utilization. In *2015 IEEE Real-Time Systems Symposium*, pages 87–95, 2015.
18. Shimon Even, Alan L. Selman, and Yacov Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61(2):159–173, 1984.
19. Fengxiang Zhang and Alan Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transactions on Computers*, 2009.
20. Carlo Meroni, Silviu S. Craciunas, Anaïs Finzi, and Paul Pop. Mapping and integration of event- and time-triggered real-time tasks on partitioned multi-core systems. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2023.
21. Anaïs Finzi, Silviu S. Craciunas, and Marc Boyer. Integrating sporadic events in time-triggered systems via affine envelope approximations. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 15–28, 2024.
22. D. Knuth. *The Art of Computer Programming: Vol. 2, Seminumerical Algorithms.* Addison-Wesley, 1973.
23. M. R. Garey and D. S. Johnson. Strong NP-completeness results: Motivation, examples, and implications. *Journal of the ACM*, 25(3):499–508, 1978.
24. Pontus Ekberg and Wang Yi. Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard. In *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, pages 139–146. IEEE Computer Society, 2017.
25. Sanjoy Baruah and Joel Goossens. Scheduling real-time tasks: Algorithms and complexity. In Joseph Y.-T Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRC Press LLC, 2003.
26. Rodolfo Pellizzoni and Giuseppe Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Real-Time Systems: The International Journal of Time-Critical Computing*, 30(1–2):105–128, May 2005.
27. Pontus Ekberg and Wang Yi. *Complexity of Uniprocessor Scheduling Analysis*, pages 1–18. Springer Singapore, Singapore, 2022.
28. I. Ripoll, A. Crespo, and A. K. Mok. Improvement in feasibility testing for real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 11:19–39, 1996.
29. Gurobi Optimization, Inc. Gurobi Optimizer 12.0 Documentation. `https://docs.gurobi.com/current/`, 2025. Last accessed July 3, 2025.
30. Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Christoph Graczyk, Katrin Halbig, Ivo Hedtke, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Julian Manns, Gioni Mexi, Erik Mühmer, Marc E.

Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Dieter Weninger, and Liding Xu. The SCIP Optimization Suite 9.0. arXiv preprint arXiv:2402.17702, 2024.

31. Naser Tamimi. How fast is c++ compared to python? https://docs.gurobi.com/current/, 2020. Last accessed July 3, 2025.
32. IBM Corporation. Compiled versus interpreted languages, 2020. Last accessed July 3, 2025.
33. Pypy documentation. https://doc.pypy.org/en/latest/. Last accessed July 3, 2025.
34. Roger Stafford. Random Vectors with Fixed Sum. https://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum, 2006. Last accessed July 3, 2025.
35. Paul Emberson, Roger Stafford, and Robert Davis. A Taskset Generator for Experiments with Real-Time Task Sets. http://retis.sssup.it/waters2010/tools.php, 2010. Includes Python implementation of randfixedsum. Released under a permissive license. Developed for WATERS 2010. Last accessed June 18, 2025.
36. Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmarks for free. In *Proceedings of the 6th International Workshop on Analysis Tools and Methodologies for Embedded Real-time Systems (WATERS)*, July 2015.
37. Silviu S Craciunas, Ramon Serna Oliver, and Valentin Ecker. Optimal static scheduling of real-time tasks on distributed time-triggered networked systems. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–8. IEEE, 2014.
38. Joël Goossens, Emmanuel Grolleau, and Liliana Cucu-Grosjean. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. *Real-time systems*, 52(6):808–832, 2016.