Towards Efficient Explainability of Schedulability Properties in Real-Time Systems

Sanjoy Baruah
Washington University in Saint Louis, MO, USA

Pontus Ekberg
Uppsala University, Sweden

Abstract

The notion of **efficient explainability** was recently introduced in the context of hard-real-time scheduling: a claim that a real-time system is schedulable (i.e., that it will always meet all deadlines during run-time) is defined to be efficiently explainable if there is a proof of such schedulability that can be verified by a polynomial-time algorithm. We further explore this notion by (i) classifying a variety of common schedulability analysis problems according to whether they are efficiently explainable or not; and (ii) developing strategies for dealing with those determined to not be efficiently schedulable, primarily by identifying practically meaningful sub-problems that are efficiently explainable.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Scheduling

Keywords and phrases Recurrent Task Systems, Uniprocessor and Multiprocessor Schedulability, Verification, Explanation, Computational Complexity, Approximation Schemes

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2023.2

Funding *Sanjoy Baruah*: US National Science Foundation (Grants CNS-2141256 and CNS-2229290). *Pontus Ekberg*: Swedish Research Council grant 2018-04446.

1 Introduction

A workshop titled Explainability of Real-Time Systems and their Analysis (ERSA) was held as part of the 2022 edition of the IEEE Real-Time Systems Symposium (RTSS), with a goal "to understand the role, meaning, and value of explanation in critical systems – in particular real-time systems." In a paper [8] that we had presented at this workshop, we introduced the notion of **efficient explainability** in the context of hard-real-time scheduling. In this we drew inspiration from the remarkable success that cyclic-executive (CE) [4, 5] based approaches to demonstrating timing correctness in safety-critical systems have enjoyed, particularly with regard to achieving statutory certification. In such CE based approaches, the system developer provides the certification authority (CA) with a lookup table that explicitly enumerates which task will execute at each instant; the CA checks that repeated execution of this lookup table assigns adequate computing to each task to allow all its timing constraints to be met (provided, of course, that each task respects its worst-case execution time bound). From this perspective, the lookup table may be thought of as a *certificate* that "explains" the system-developer's claim that the system is schedulable. We accordingly defined a claim of schedulability to be efficiently explainable if there is a certificate of the schedulability that can be verified in time polynomial in the representation of the system for which schedulability is being claimed. We applied this notion to the schedulability analysis of independent sporadic task systems upon preemptive uniprocessors, and made some simple observations that may nevertheless be surprising (e.g., that for uniprocessor scheduling of

constrained-deadline sporadic task systems [27, 30], fixed-priority schedulability is efficiently explainable but, under the widely-held assumption that $\texttt{NP} \neq \texttt{coNP}$, EDF-schedulability is not), and posed some open questions and possible directions for further research.

This Work: Motivation and Scope. We believe that the concept of efficient explainability potentially has an important role to play in the verification of future safety-critical real-time systems as such systems become ever more complex, and as they are increasingly coming to be implemented upon resource-constrained platforms:

1. CAs may be unwilling to undertake inordinately long computational procedures in order to verify the correctness of systems that are submitted for certification. This becomes more challenging for modern systems given the increased complexity of these systems and the correctness properties that they are expected to maintain in the ever-increasingly complex environments within which they operate.
2. An additional motivation for efficient explainability comes from the increasing application of the paradigm of edge computing in many safety-critical applications like autonomous navigation. Due to size, weight and power (SWaP) constraints, devices on the edge typically have limited computational capabilities; hence it is beyond their ken to undertake complex computations that are computationally demanding. They may nevertheless need to perform complex run-time operations such as admission control. One approach to this would be to perform the actual admission control computation "in the cloud" where more extensive computational capabilities are available. If such cloud computations determine that the workload remains schedulable upon admitting the additional work, then the specifications of the schedulable system including the newly-admitted work, along with an efficiently-verifiable proof (the "efficient explanation") of the schedulability, are communicated to the computationally limited edge device. The edge device can then verify the schedulability and admit the new work, *without* needing to trust the cloud computation.

These motivating considerations have prompted us to study the concept of efficient explainability further, with a view of better understanding its applicability to safety-critical system design, implementation, and verification. In this paper we report on our investigations into two **aspects** of efficient explainability:

1. We characterize several schedulability-analysis problems beyond those considered in [8] as being efficiently explainable or not.
2. If some schedulability analysis problem is unlikely to allow efficiently-verifiable certificates for all instances, there may be sub-problems of it that do. We investigate this idea in depth, proposing several avenues to identifying efficiently explainable sub-problems, as well as alternative notions of efficient explainability.

The specific **contributions** of our work include the following.

1. We extend the results of [8] to *multiprocessors* by characterizing the efficient explainability (or absence thereof) of multiprocessor partitioned schedulability of recurrent task systems.
2. We obtain a series of results that identify *efficiently explainable sub-problems of uniprocessor EDF-schedulability* of sporadic task systems (a problem that was observed in [8] to not be efficiently explainable). We also extend these results to *partitioned EDF scheduling upon multiprocessor platforms.*

3. We propose a novel concept, *Fully Polynomial-Time* ***Verification*** *Approximation Scheme* (FPT**V**AS), that extends the notion of FPTAS's, widely studied in the context of approximation algorithms [3, 34], from polynomial-time computation to polynomial-time verification (i.e., from `P` to `NP`).
4. We introduce, and initiate a study of, a previously unconsidered complexity class – `pseudoNP`*: non-deterministic pseudo-polynomial time* that helps generalize the concept of efficient explainability. This generalization seems particularly relevant as the schedulability problems for which efficient explanations are sought become computationally more challenging.

Organization. The remainder of this manuscript is organized as follows. In Section 2 we place this work within the larger context of verification of real-time systems, and briefly review some concepts from real-time scheduling and complexity theory that will be used in later sections of this document. Sections 3–6 contain our main technical contributions: after briefly identifying some efficiently explainable multiprocessor schedulability analysis problems in Section 3, we turn our attention, in Sections 4–6, to identifying efficiently explainable sub-problems of uni- and multi-processor schedulability analysis problems that are unlikely to be efficiently explainable. We conclude in Section 7 by reiterating the significance of the research described in this paper, and proposing some directions for further research.

2 Context, and Background & Related Work

The web page[1] for the ERSA workshop observes that "many software-intensive systems of current and future application domains require (or will require) approval from a certification authority." It highlights limitations of current approaches to obtaining such approval, particularly when applied to advanced application domains, and states, as motivation for the workshop, that "it is worth exploring $[\cdots]$ whether explainability can help." We would like to emphasize that there are a myriad of aspects to explainability as it pertains to such a use-case. Let us consider, as an illustrative toy example, an effort at convincing a certification authority (CA) of the correctness of the timing behavior of a particular system by appealing to the well-known result in Liu and Layland's seminal paper [27] that *a periodic task system with utilization* $\leq \ln 2$ *will meet all its deadlines when scheduled using rate-monotonic scheduling* (or in the terminology of explainability, that *a periodic task system's utilization being* $\leq \ln 2$ *constitutes an 'explanation' of its schedulability.*)

1. The party seeking certification must demonstrate that the periodic task model proposed in [27] adequately models the salient characteristics of the workload, and that the preemptive uniprocessor model assumed in [27] adequately models the salient characteristics of the implementation platform.
2. They must provide justification for the values they have assigned to the task WCET parameters (by, for instance, showing that the values were obtained using tools [35] that have been certified for this purpose), as well as the values assigned to the task period parameters.[2]

[1] `https://sites.google.com/view/ersa22`
[2] Examples of WCET analysis tools are aiT (`https://www.absint.com/ait/`), Heptane (`https://team.inria.fr/pacap/software/heptane/`) and OTAWA (`http://www.tracesgroup.net/otawa/`).

3. The CA must accept the validity of the analysis presented in [27], that any rate-monotonic scheduled system with utilization $\leq \ln 2$ will meet all deadlines.[3]
4. Finally, the party seeking certification must demonstrate to the CA that the system utilization – i.e., the sum, over all the tasks in the system, of the ratio of their WCET parameters to their period parameters – does not exceed $\ln 2$. (Recall that this constitutes the 'explanation' of the system's schedulability, which the CA will presumably seek to verify on their own – i.e., they will not simply take the word of the party seeking certification that it is so.)

While each of these steps is crucial to achieving certification, *the focus of this paper is on the last step.* (Although this step is quite trivial for our toy example above, we will see that it is not always so.) Hence the work described in this paper should be looked upon as constituting only one part of a certification process that necessarily includes a social dimension (Step 1, accepting the mapping from an actual system to an abstract model, inherently possesses a social aspect in addition to technical ones, as do parts of Step 2); tool-development and certification (Step 2); and formal verification of schedulability results (Step 3). Our long-term vision for the non-social aspects of certification is that an automatically-verifiable proof of system correctness in some formalism such as Coq [11] or Prosa [15] will be provided to the CA. This will incorporate correctness proofs of the schedulability results (see, e.g., [13, 10] for some examples of such proofs) upon which such correctness depends, and additionally include a certificate that explains why these schedulability results imply the correctness of the specific system being considered for certification (see [29] for a seminal work on creating such certificates in Prosa). It is this last part – explaining why the formally-verified general schedulability results imply correctness for the particular system – that we address: we want to ensure that the certificate of correctness can be verified *efficiently* by the CA. In this work we outline and study theoretical underpinnings to understand which schedulability problems allow for efficient verifiability (or *explainability*, in the terminology adopted in this paper), and how to deal with schedulability problems that are unlikely to allow this for all instances.

Efficient Explainability. A schedulability problem is here said to be efficiently explainable if and only if for all schedulable task systems there is a certificate of this schedulability that can be verified by an algorithm that has running time polynomial in the size of the representation of the task system. This definition directly links efficient explainability to well-studied concepts in computational complexity theory [31, 2]; in particular, the complexity class NP: "NP is the class of languages that can be verified by a polynomial-time algorithm" [16, page 1064]. Hence, a schedulability problem is efficiently explainable if it belongs to NP, and showing that a schedulability-analysis problem is unlikely to be in NP offers strong evidence that it is not efficiently explainable. In Sections 5 and 6 we will also consider wider notions of efficient explainability than simply equating it with NP.

How does one show that a schedulability-analysis problem is unlikely to be in NP? Here one again makes use of well-established results from computational complexity theory: there are several complexity classes (see Figure 1 for some) that are widely believed to be distinct from NP in the sense that there are problems within these complexity classes that do not also belong in NP. Recall that a problem is defined to be *hard* for a complexity class if it

[3] We point out that this step is by no means trivial or automatic – examples abound of results that passed peer-review and were published in technical forums, only to subsequently be discovered to be erroneous. Hence it is understandable that a CA be sceptical of published results and seek further justification of their correctness than merely having passed peer review.

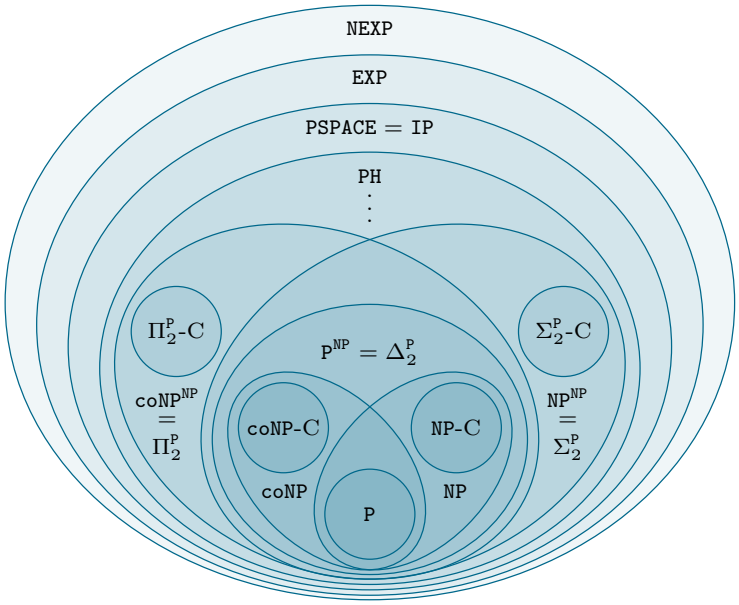

Figure 1 Complexity classes considered in this manuscript. Complexity theory researchers widely believe that no region in this diagram is empty – each is populated with problems.

is, intuitively speaking, at least as computationally difficult to solve as every other problem in that complexity class (or more precisely, every problem in the complexity class can be reduced, in polynomial time, to this hard problem). Hence showing a schedulability-analysis problem to be hard for any complexity class believed to be distinct from NP offers strong evidence that it cannot also be in NP and is therefore not efficiently explainable.

Efficient Explainability: Prior Results. The following observations and results were obtained in [8] by exploiting this equivalence between efficient explainability and membership in NP:

- It is efficiently explainable whether or not a constrained-deadline synchronous periodic or sporadic[4] task system is FP-schedulable upon a preemptive uniprocessor.
- In contrast, determining whether a constrained (or arbitrary) deadline synchronous periodic or sporadic task system is EDF-schedulable or not upon a preemptive uniprocessor platform is unlikely to be efficiently explainable.
- Suppose one were given an FP-scheduled constrained-deadline synchronous periodic task system in which each task is additionally characterized by a lower bound (a 'best-case execution time') on its execution duration, and a lower bound is specified on the response time – i.e., the duration between a job's arrival and its completion – that is acceptable for each task. Determining whether such a system will be scheduled to respect the specified response time lower bounds is not likely to be efficiently explainable.

[4] Recall that for a *periodic* task the period parameter denotes the exact duration between successive job arrivals, whereas for a *sporadic* task it denotes the minimum duration between successive job arrivals. In a *synchronous* periodic task system, the first task of each job arrives at time-instant zero.
In *constrained*-deadline task systems, the relative deadline parameter of each task is $\leq$ the task's period parameter; no such relationship need exist in *arbitrary*-deadline task systems. (*Implicit*-deadline task systems have each task's deadline parameter equal to its period parameter.)

FPTAS's. As stated in Section 1, one of our goals is to identify efficiently explainable sub-problems of schedulability-analysis problems that are determined to not be efficiently explainable. We will see that this goal essentially translates to one of obtaining sufficient (rather than exact) schedulability tests. *Speedup factors* [24, 32, 25] are a commonly used quantitative metric of the effectiveness of sufficient schedulability tests. The speedup factor of a sufficient schedulability test $\mathcal{A}$ is defined to be the smallest real number $\delta \geq 0$ such that if any task system Γ is schedulable upon a unit-speed processor, then $\mathcal{A}$ will determine that Γ is schedulable upon a speed-$(1+\delta)$ processor. Smaller speedup factors denote 'better' (i.e., closer to optimal in the worst case) sufficient tests. Thus, obtaining a good sufficient schedulability test may be thought of as obtaining a good approximation algorithm that minimizes the speedup factor. In the theory of approximation algorithms [3, 34], it is widely accepted that an FPTAS (see, e.g., [16, p. 1107] for a textbook description) is the 'best' kind of approximation algorithm: it allows for approximations that are arbitrarily close to the optimal by appropriately assigning a value to a parameter δ. In the context of sufficient schedulability tests, an FPTAS may be defined as follows:[5]

▶ **Definition 1** (FPTAS). *A fully polynomial-time approximation scheme (FPTAS) for a schedulability analysis problem is an algorithm that, given as input any problem instance* Γ *and a parameter* $\delta > 0$*, returns "unschedulable" if* Γ *is unschedulable on a speed-1 processor, and returns "schedulable" if* Γ *is schedulable on a speed-*$(1/(1+\delta))$ *processor. Its running time is bounded by a polynomial in the two parameters* $|\Gamma|$ *and* $\left(\frac{1}{\delta}\right)$*.*

2.1 Some relevant results from real-time scheduling theory

Fixed-Priority scheduling. *Response-time analysis (RTA)* [23, 26] is the standard technique for determining whether a constrained-deadline synchronous periodic task system is schedulable or not under fixed-priority (FP) scheduling. RTA is based on the observation [23] that if a constrained-deadline task system is schedulable under FP, then the maximum possible duration between the release of a job of τ_i and the instant this job completes execution (called the *worst-case response time* of task τ_i) is equal to the smallest positive value of R_i that satisfies the following recurrence (here $\text{hp}(\tau_i)$ denotes all jobs in the task system that have scheduling priority greater than τ_i's scheduling priority):

$$R_i = C_i + \sum_{\tau_j \in \text{hp}(\tau_i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \tag{1}$$

As observed in [8], it is well known that FP-schedulability is in NP and must therefore be efficiently explainable: the certificate for FP-schedulability for a given task system Γ is a value for R_i for each $\tau_i \in \Gamma$ that satisfies Expression 1 and is $\leq D_i$'s. Such a certificate comprises $|\Gamma|$ numbers, and so is polynomial (in fact linear) in the representation of the task system Γ. It is straightforward to observe that each claimed R_i can be verified to be a solution to Equation 1 in linear time.

[5] It should be noted here that we are abusing the notion of an FPTAS slightly, to be consistent with prior work in real-time scheduling. In Definition 1 we are assuming that processor speed is the quantity to be approximated, even though the usual definition of FPTAS's allow any other approximation metric. Other metrics relevant for scheduling are, for example, makespan or maximum tardiness. We are also not quite treating the FPTAS in Definition 1 as an approximation algorithm (since the algorithm should output only "schedulable" or "unschedulable"), but without much further work we can use binary search to turn such an algorithm into an algorithm for approximating the minimum processor speed needed to schedule the task system.

Earliest-Deadline-First scheduling. In earliest-deadline-first (EDF) scheduling, at each instance the currently active (i.e., needing execution) job with the earliest deadline is executed. It is unlikely that EDF-schedulability is efficiently explainable since it has been shown [18, 20, 19] to be coNP-hard. If it was possible to create polynomial-time verifiable certificates for *all* EDF-schedulable task systems, attesting their EDF-schedulability, then by definition the EDF-schedulability problem would be in NP. This would immediately imply that NP = coNP, which goes against the expectations of most researchers in complexity theory. It is interesting to note that even though uniprocessor FP-schedulability (which is NP-complete [22]) and uniprocessor EDF-schedulability (which is coNP-complete [18]) in some sense are qualitatively equally hard to solve (they are both complete at the first level of the polynomial hierarchy), their verification problems are indeed of very different hardness.

Processor-demand analysis (PDA) [7] is the standard technique for determining whether a synchronous periodic task system is schedulable or not under EDF scheduling. PDA is centered upon the concept of the *demand bound function* (DBF): for any sporadic task τ_i and any interval-duration $t \geq 0$, $\text{DBF}_i(t)$ denotes the maximum possible cumulative execution requirement by jobs of task τ_i that both arrive in, and have deadlines within, any interval of duration t. The following formula for computing $\text{DBF}_i(t)$ was derived in [7]:

$$\text{DBF}_i(t) = \max\left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1, 0\right) \times C_i \tag{2}$$

It was also shown in [7] that a necessary and sufficient condition for task system Γ to be EDF-schedulable is that the following condition holds for all $t \geq 0$:

$$\sum_{\tau_i \in \Gamma} \text{DBF}_i(t) \leq t \tag{3}$$

Processor Demand Analysis, PDA, is essentially a means of determining whether Expression 3 holds for all t. It was proved in [7] that Condition 3 need only be checked for values of t that are of the form $t = (k \times T_i + D_i)$ for some non-negative integer k and some $\tau_i \in \Gamma$; furthermore, only such values that are no larger than the hyper-period $\text{HP}(\Gamma)$ (the least common multiple of all the T_i parameters) need be tested. The set of all such values of t for which it needs to be checked that Condition 3 is satisfied in order to verify EDF-schedulability is called the *testing set* for the sporadic task system Γ and often denoted $\mathcal{T}(\Gamma)$. It is known [7] that the cardinality $|\mathcal{T}(\Gamma)|$ of the testing set $\mathcal{T}(\Gamma)$ may in general be exponential in the representation of Γ. However, it has been shown [6, Theorem 3.1] that a smaller testing set, of pseudo-polynomial cardinality (i.e., polynomial in the size of the representation of Γ and its largest numerical parameter), can be identified for *bounded-utilization* task systems. Those are the task systems Γ satisfying the additional condition that $\sum_{\tau_i \in \Gamma} C_i/T_i \leq c$ for some pre-defined constant $c < 1$.

3 Efficiently Explainable Schedulability

We start out identifying some important efficiently explainable schedulability-analysis problems. As mentioned in Section 2.1 above, fixed-priority (FP) schedulability of constrained-deadline sporadic task systems upon preemptive uniprocessor platforms was observed to be efficiently explainable in [8]; this result is easily generalized to show that FP schedulability of constrained-deadline sporadic task systems upon *multi*processor platforms under the partitioned paradigm[6] is also efficiently explainable:

[6] Throughout this paper, when discussing partitioned scheduling, we refer to strict temporal partitioning where tasks on one partition execute independently of tasks on another partition. Semi-partitioned approaches or cases where tasks share locks across partitions may require additional care for verifiability.

▶ **Theorem 2.** *Partitioned multiprocessor fixed-priority schedulability of constrained-deadline sporadic task systems is efficiently explainable.*

Proof. In Section 2.1 we have briefly described the procedure for constructing efficiently-verifiable certificates of uniprocessor FP-schedulability of constrained deadline task systems. This procedure is easily generalized to establish the efficient explainability of partitioned FP-schedulability for constrained-deadline systems upon multiprocessors as well: an efficiently-verifiable certificate of the partitioned fixed-priority schedulability for a constrained-deadline sporadic task system Γ upon an m-processor platform comprises

1. The actual partitioning of Γ into the m sets of tasks assigned to the m processors; and
2. For each of the m partitions, a certificate of its uniprocessor fixed-priority schedulability that is constructed using the procedure that we had described in Section 2.1.

It is evident that such a certificate can be verified by a polynomial-time algorithm, and hence establishes that FP-schedulability under the partitioned paradigm of multiprocessor scheduling is efficiently explainable.

We point out that nothing in this proof requires that the processors be identical to one another; hence the result of this theorem holds for the more general heterogeneous multiprocessor platforms. ◀

As stated in Section 2.1, preemptive uniprocessor EDF-schedulability analysis for sporadic task systems is not likely to be efficiently explainable, in contrast to FP scheduling. Since partitioned multiprocessor EDF scheduling is a generalization of uniprocessor EDF scheduling, it therefore follows that partitioned multiprocessor EDF-schedulability analysis problem is also unlikely to be efficiently explainable. However, for the special case of implicit-deadline task systems (for which the corresponding uniprocessor problem is efficiently explainable – simply determine whether system utilization does not exceed the processor capacity), efficient explainability is easily established:

▶ **Theorem 3.** *Partitioned multiprocessor EDF schedulability of implicit-deadline sporadic task systems is efficiently explainable.*

Proof. An efficiently-verifiable certificate of the partitioned EDF schedulability for an implicit-deadline sporadic task system Γ upon an m-processor platform comprises the actual partitioning of Γ into the m sets of tasks assigned to the m processors. Given such a certificate, it can be verified in polynomial time that the sum of the utilizations of the tasks assigned to each processor does not exceed the capacity of that processor. ◀

Perhaps somewhat surprisingly, most other common real-time schedulability analysis problems are unlikely to have polynomial-time verifiable certificates for explaining schedulability; this motivates our efforts, reported in Sections 4–6, to investigate other avenues to dealing with such problems.

4 Identifying Efficiently Explainable Sub-Problems

In this section we consider one approach to achieving efficient explainability of problems that are *not* in NP (and hence not efficiently explainable): to identify relevant sub-problems that are in NP. We exemplify this approach with the EDF-schedulability problem. As stated in Section 2.1, the uniprocessor EDF-schedulability problem of three-parameter[7] sporadic

[7] That is, a task specified by a triple of its task parameters: $\tau_i = (C_i, D_i, T_i)$.

(or periodic) tasks is coNP-complete, from which it follows that it cannot be in NP unless NP = coNP and the polynomial hierarchy collapses to its first level. Barring the unlikely event of such a collapse, we cannot produce polynomially-sized explanations of EDF schedulability for all schedulable task systems that can be verified by a third party in polynomial time.

We set out to identify sub-problems of the uniprocessor EDF-schedulability problem for sporadic (or synchronous periodic) task systems that are in NP, and therefore efficiently explainable. We do so by identifying subsets of the language of EDF-schedulable task sets that are in NP (or indeed, even in P). Since the union of a finite set of languages in NP is itself a language in NP, we can try to cover as much as possible of the EDF-schedulable task sets with different subsets in NP, and in the end take the union of any such subsets. Any EDF-schedulable task set that is in this union of smaller languages *does* have a polynomially-sized certificate that can be verified by a third party in polynomial time.

Before proceeding we should make some important points.

- It is not possible to completely cover the coNP-complete language of EDF-schedulability by a finite set of subset languages that are each in NP (assuming NP $\neq$ coNP) – there will remain an infinite set of EDF-schedulable task sets that are not covered.
- In this work we do not attempt to maximize the covering, but instead focus on identifying a few rather 'natural' subsets that are in NP. The covering presented here could most certainly be expanded – this is a direction of future work.
- We are here interested in the efficient explainability of the identified subsets – that it is possible to efficiently verify a certificate of a solution – but for now are not much concerned by the efficiency of finding those solutions (and certificates) in the first place. As a result, some subsets may in practice be more difficult computational problems to *solve* than the original EDF-schedulability problem, but they will be easier to verify.

4.1 Efficiently explainable subsets of uniprocessor EDF-schedulability

Here we will list efficiently explainable subsets (forming languages in NP) to the EDF-schedulability problem. The relationships between these subsets are shown in Figure 2. The most natural such subsets are *those for which the EDF-schedulability problem itself is easy to solve*. The following two subsets of EDF-schedulability are known to be in P.

[I] – Implicit deadlines. From Liu and Layland [27] we know that a task set Γ is EDF-schedulable if it has implicit deadlines and $\sum_{\tau_i \in \Gamma} C_i/T_i \leq 1$. Both conditions are trivial to check in polynomial time, and so EDF-schedulable task sets with implicit deadlines form a language in P (and therefore in NP).

[II] – Harmonic periods and constrained deadlines. Similarly, Bonifaci et al. [12] have given a polynomial-time algorithm for determining if a task set with constrained deadlines and harmonic periods is EDF-schedulable. Hence such task sets also form a language in P (and therefore in NP).

Another approach to identifying efficiently explainable subsets is by *exploiting the optimality of EDF upon preemptive uniprocessors*. From this optimality we trivially get that for any uniprocessor scheduling algorithm $\mathcal{A}$, the set of all of $\mathcal{A}$-schedulable task systems is a subset of the set of EDF-schedulable task systems. For which $\mathcal{A}$ is $\mathcal{A}$-schedulability in NP? An obvious candidate is Fixed-Priority (FP).

[III] – FP-schedulable with constrained deadlines. As explained in Section 2.1, the task systems that are FP-schedulable form a language in NP, and this language must be a subset of EDF-schedulability. We should use the optimal deadline-monotonic (DM) priority

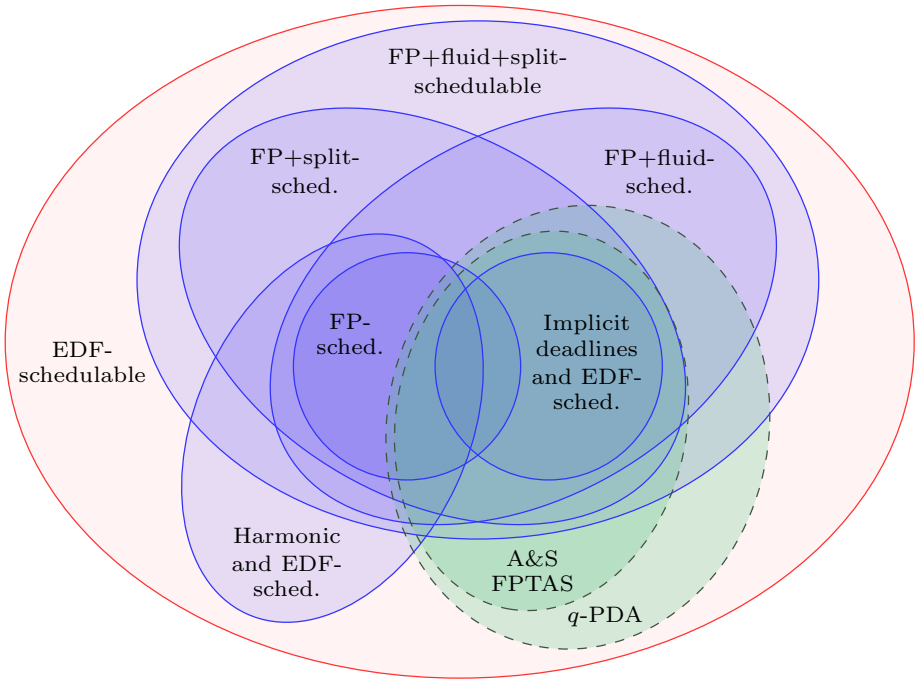

Figure 2 The relationships between the different languages of uniprocessor schedulability considered in this work. While the outer language of EDF-schedulability is coNP-complete, the marked subsets are all languages in NP. The union of any collection of these subsets is also in NP and therefore admits polynomial-time verifiable certificates. The two languages demarcated by dashed lines – the *Albers & Slomka FPTAS* and *EDF-schedulability by q-PDA* – are of a different sort than the others in that they are parameterized approximations; they will be described later in Section 5.

ordering to make the subset as large as possible. It is not known if FP-schedulability for task sets with *arbitrary* deadlines is in NP, but we can certainly truncate any $D_i > T_i$ to $D_i = T_i$ as preprocessing if we wish.

We note that Davis et al. [17] have shown that the considered subset of FP-schedulable task sets with constrained deadlines (and DM priorities) *itself contains as a strict subset* the set of all task sets with constrained deadlines that are EDF-schedulable on a speed-Ω processor, where $\Omega \approx 0.56714$ is the unique constant such that $\Omega e^{\Omega} = 1$. This gives us a bound on how much processor capacity we can lose in the worst case by proving EDF-schedulability by means of FP-schedulability.[8]

Are there more uniprocessor scheduling algorithms $\mathcal{A}$ for which $\mathcal{A}$-schedulability is in NP? There are very many, and the key to seeing this is to recognize that $\mathcal{A}$ *does not need to be any kind of "practically reasonable" scheduling algorithm*, since we do not intend to actually execute $\mathcal{A}$ during runtime – we are simply exploiting the fact that, since EDF is optimal,

[8] As a curiosity and an aside, this also means we have the marvelous situation that a coNP-complete language (EDF-schedulability on a speed-1 processor) contains an NP-complete language (FP-schedulability on a speed-1 processor) that in turn contains a coNP-complete language (EDF-schedulability on a speed-Ω processor). Indeed, we can make the infinite chain of strict subsets

$$\text{(EDF, speed-1)} \supset \text{(FP, speed-1)} \supset \text{(EDF, speed-}\Omega\text{)} \supset \text{(FP, speed-}\Omega\text{)} \supset \text{(EDF, speed-}\Omega^2\text{)} \supset \cdots$$

which alternate forever between being coNP- and NP-complete!

$\mathcal{A}$-schedulability necessarily implies EDF-schedulability. Thus we just want to efficiently verify EDF-schedulability by means of verifying $\mathcal{A}$-schedulability instead, but EDF, which has very efficient implementations, will be the run-time scheduling algorithm that is used. *Fluid schedulers* are among the scheduling algorithm that are easy to reason about, although they may be difficult to implement. Next, we consider a scheduler that schedules some tasks fluidly.

[IV] – FP+fluid-schedulability. A fluid scheduler assigns a constant fraction f_i of the processor to task τ_i, such that τ_i is served continuously at this rate. Clearly, τ_i will meet all of its deadlines if it scheduled fluidly with a rate $f_i = \delta_i$, where $\delta_i = C_i/\min(D_i, T_i)$ is the density of the task.

However, we do not need to schedule all tasks in the task set fluidly. We define the FP+fluid scheduler to be the scheduler that (optimally) partitions the task set Γ into two disjoint subsets Γ_{fluid} and Γ_{fp}, and then schedules each task $\tau_i \in \Gamma_{\text{fluid}}$ fluidly with rate δ_i, and schedules the tasks $\tau_i \in \Gamma_{\text{fp}}$ with an FP-scheduler (and DM priorities). The tasks in Γ_{fluid} run on a reserved processor fraction of speed Δ, where $\Delta = \sum_{\tau_i \in \Gamma_{\text{fluid}}} \delta_i$, and the tasks in Γ_{fp} run on the "remaining" processor fraction of speed $1 - \Delta$.

To see that FP+fluid-schedulability is in NP, consider that all tasks will meet their deadlines if both $\Delta \leq 1$ and the tasks in Γ_{fp} are FP-schedulable on a processor of speed $1 - \Delta$. A certificate of FP+fluid-schedulability can then simply consist of (1) the partitioning of Γ into Γ_{fluid} and Γ_{fp}, and (2) fixed-points to the response-time equation of the tasks in Γ_{fp}, where each C_i has been multiplied by $1/(1 - \Delta)$. Given such a certificate we could easily verify in polynomial time that indeed $\Gamma = \Gamma_{\text{fluid}} \cup \Gamma_{\text{fp}}$, that $\Delta \leq 1$, and that the given fixed-points are actual fixed-points ($\leq D_i$) to the response-time equation for each task $\tau_i \in \Gamma_{\text{fp}}$ on a speed-$(1 - \Delta)$ processor.

FP+fluid-schedulability is of course a superset of plain FP-schedulability (since we can set $\Gamma_{\text{fluid}} = \emptyset$ and $\Gamma_{\text{fp}} = \Gamma$) and a superset of plain fluid scheduling (since we can set $\Gamma_{\text{fluid}} = \Gamma$ and $\Gamma_{\text{fp}} = \emptyset$). To see that FP+fluid-schedulability is in fact a strict superset of the union of both, we can consider the following simple task set:

$$\Gamma = \{\tau_1 = (2, 4, 4), \tau_2 = (3, 6, 8), \tau_3 = (1, 9, 10)\}$$

It can be readily checked that this task set is not fluid-schedulable (the total density > 1) and is also not FP-schedulable (τ_2 will miss a deadline under DM-priority ordering). It is however FP+fluid-schedulable with $\Gamma_{\text{fluid}} = \{\tau_1\}$ and $\Gamma_{\text{fp}} = \{\tau_2, \tau_3\}$.

FP+fluid-schedulability is an example of a subset of EDF-schedulability that seems potentially harder to *solve* in practice than just solving EDF-schedulability. For example, it is well-known that EDF-schedulability can be solved in pseudo-polynomial time for bounded-utilization task sets [7], but it is not obvious that FP+fluid-schedulability can be so solved if we want to find the best possible partitioning of Γ into Γ_{fluid} and Γ_{fp}. However, this is not our main concern, and being in NP it is qualitatively easier to verify the solutions of FP+fluid-schedulability.

Task splitting is another common scheduling technique that we can exploit to come up with suitable scheduling algorithms $\mathcal{A}$. Task splitting is known to often improve schedulability (see for example [14], where this technique is referred to as *period transformation*), but it normally comes with the drawback of some extra runtime overheads, especially if tasks are

split into many much smaller pieces. This is not a concern in this context as we again do not intend to ever run the resulting scheduling algorithm $\mathcal{A}$, we merely want to find $\mathcal{A}$ such that $\mathcal{A}$-schedulability is in NP.

[V] – FP+split-schedulability. We consider here the simple splitting technique where a constrained-deadline task $\tau_i = (C_i, D_i, T_i)$ can be split into the smaller task

$$\tau_i' = \left(\frac{C_i}{k_i}, \frac{T_i}{k_i} - (T_i - D_i), \frac{T_i}{k_i}\right),$$

for $k_i \in \mathbb{N}_+$, with which every job of τ_i is served as k_i jobs of τ_i'. It can be readily confirmed that if the k_i jobs of τ_i' all meet their deadlines, then so does the original job that they serve. Note that splitting a task τ_i with $D_i < T_i$ may result in it receiving a negative relative deadline, in which case the split task is clearly unschedulable.

FP+split-schedulability is in NP since we can provide as a certificate the k_i's and fixed-points to the response-time equation for the split tasks τ_i'. The solution is then verified in polynomial time by reproducing the split tasks using the k_i's (some tasks may have $k_i = 1$, and remain unsplit) and verifying that the provided fixed-points are indeed valid fixed-points to the response-time equation that are each $\leq \frac{T_i}{k_i} - (T_i - D_i)$.

Now that we have FP+fluid and FP+split, nothing is stopping us from combining the power of both, if indeed verification time and not solution time is our main concern, because combining them will not take us out of NP.

[VI] – FP+fluid+split-schedulability. The FP+fluid+split scheduler simply partitions the task set Γ into Γ_{fluid} and Γ_{fp}, and then schedules the tasks in Γ_{fluid} fluidly and then allows the tasks in Γ_{fp} to be split into smaller tasks before they are scheduled by an FP scheduler on the remaining processor fraction. Polynomial-time verifiable certificates are easily constructed similarly to how they are constructed for FP+fluid and FP+split.

We note that FP+fluid-schedulability [IV] and FP+split-schedulability [V] both contain as subsets the plain FP-schedulability for constrained deadlines [III] as well as EDF-schedulability with implicit deadlines [I]. As we will see below, neither contain the other though, they each cover different parts of the original EDF-schedulability problem. The following is a simple task set that can be readily checked to be unschedulable by FP+fluid, but schedulable by FP+split (by splitting τ_1 into $\tau_1' = (1, 1, 2)$):

$$\Gamma = \{\tau_1 = (2, 3, 4), \tau_2 = (3, 6, 6)\}$$

In the other direction, the following example can be checked to be unschedulable by FP+split (no task can be split and keep a non-negative relative deadline), but to be schedulable by FP+fluid (by setting $\Gamma_{\text{fluid}} = \{\tau_3\}$ and $\Gamma_{\text{fp}} = \{\tau_1, \tau_2\}$):

$$\Gamma = \{\tau_1 = (1, 2, 9), \tau_2 = (7, 9, 100), \tau_3 = (1 + \epsilon, 10, 100)\}$$

Now that we have seen that FP+fluid-schedulability and FP+split-schedulability cover different parts of the EDF-schedulability language, we note that FP+fluid+split-schedulable is in fact more than just the union of these two. This is demonstrated by the following example, which is unschedulable by both FP+fluid and FP+split, but is schedulable by FP+fluid+split:

$$\Gamma = \{\tau_1 = (3, 6, 8), \tau_2 = (7, 12, 100), \tau_3 = (1/2 + \epsilon, 13, 100)\}$$

It is readily confirmed that none of the eight possible choices for which subset of tasks to schedule fluidly would cause Γ to be FP+fluid-schedulable, and none of the possible task splittings (only τ_1 can be split and keep a positive deadline) would cause Γ to be FP+split-schedulable. However, by splitting τ_1 into $\tau_1' = (3/2, 2, 4)$ and by setting $\Gamma_{\text{fluid}} = \{\tau_3\}$ and $\Gamma_{\text{fp}} = \{\tau_1', \tau_2\}$, the task system is indeed FP+fluid+split-schedulable (for small enough ϵ).

The above list of sub-problems in NP to uniprocessor EDF-schedulability is by no means comprehensive. It is always possible to come up with additional artificial sub-problems in NP (for example by hard-coding schedulable task systems), and quite likely there are several more "natural" sub-problems other than the ones presented here as well.

We note that while the above is presented for three-parameter sporadic or synchronous periodic tasks, we can trivially use exactly the same sub-problems for asynchronous periodic tasks as well, by simply ignoring the release offsets of all tasks. This is because the synchronous arrival sequence is the worst-case for uniprocessor EDF [7], so if we can show EDF-schedulability for the corresponding *synchronous* task system (for example using one of the explainable sub-problems above), then we immediately show it also for the asynchronous task system.

4.2 Efficiently explainable subsets of partitioned EDF-schedulability

Similarly to the FP-schedulability case in Section 3, as soon as we have efficient explainability of a uniprocessor schedulability problem, then we automatically get efficient explainability of the corresponding partitioned multiprocessor schedulability problem. This is the case even though the partitioned multiprocessor variant may be much harder to *solve* than the uniprocessor variant. For instance, while the EDF-schedulability problem is "only" coNP-complete on uniprocessors, it is both NP- and coNP-hard on partitioned multiprocessors, thus unlikely to be even in coNP. Indeed, the partitioned EDF-schedulability problem for asynchronous task systems is Σ_2^{P}-complete [21], and is a much harder problem than for uniprocessors. Even so, partitioned EDF-schedulability is no less explainable than the uniprocessor variant. A certificate of partitioned EDF-schedulability can simply consist of the partitioning $\Gamma_1, \ldots, \Gamma_m$ of Γ upon the m processors, together with a certificate of uniprocessor EDF-schedulability for each of the m partitions. This certificate is verified by checking that indeed $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_m$, and by verifying the per-partition certificates. Note that the per-partition certificates do not have to be of the same type, they could for example be certificates from different sub-problems listed earlier in this section.

5 A Scheme for Efficient Explainability

In Section 4 we explored two approaches to identifying efficiently explainable sub-problems of the EDF-schedulability analysis problem: (i) restricting the problem instances (in [I] and [II]); and (ii) instead testing schedulability for some other scheduler that is dominated by EDF (in [III]–[VI]). Here we propose a third approach: directly designing a sufficient schedulability test for EDF-schedulability analysis. For this the sufficient test should define a language in NP, in the sense that for all task systems that pass the test, we can create polynomial-time verifiable certificates of this fact.

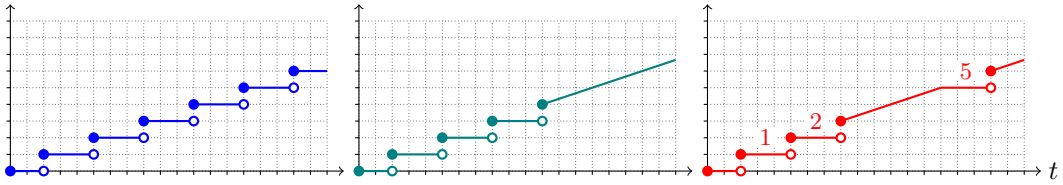

Figure 3 The left plot depicts $\text{DBF}_i(t)$ for a task τ_i with $C_i = 1, D_i = 2$, and $T_i = 3$. The center plot depicts $\overline{\text{DBF}}_i(t)$ with $k = 3$, and the right plot, $\overline{\text{DBF}}_i(t, \mathcal{S}_i)$ for $\mathcal{S}_i = \{1, 2, 5\}$. Note that $\overline{\text{DBF}}_i(t)$ ($\overline{\text{DBF}}_i(t, \mathcal{S}_i)$, respectively) is discontinuous only at the first k steps (only around the steps in $\mathcal{S}_i$, resp.), and is piecewise linear with slope $\leq C_i/T_i$ everywhere else.

As stated in Section 2.1, processor-demand analysis (PDA) is the standard test for EDF-schedulability. PDA checks that Eq. 3 (reproduced below):

$$\sum_{\tau_i \in \Gamma} \text{DBF}_i(t) \leq t$$

holds for all values of $t \in \mathcal{T}(\Gamma)$, where $\mathcal{T}(\Gamma)$ is the potentially exponentially-sized testing set. A sufficient EDF-schedulability test is easily obtained by replacing each task's demand bound function (the $\text{DBF}_i(t)$ terms in Eq. 3) with an approximation $\overline{\text{DBF}}_i(t)$, such that $\text{DBF}_i(t) \leq \overline{\text{DBF}}_i(t)$ for all t. It is evident (see Eq. 2) that $\text{DBF}_i(t)$ is a step function with a zeroth step over interval $[0, D_i)$, a first step over interval $[D_i, T_i + D_i)$ and so on – see the left plot in Fig. 3. Albers and Slomka [1] proposed the following approximation to $\text{DBF}_i(t)$ (depicted in the center plot in Fig. 3): letting k denote any positive integer constant, their approximation retains steps zero through k while the remainder is over-approximated by a straight line with slope equal to the task's utilization:

$$\overline{\text{DBF}}_i(t) = \begin{cases} \text{DBF}_i(t), & \text{for } t < D_i + kT_i \\ (T_i - D_i + t) \times \frac{C_i}{T_i}, & \text{for } t \geq D_i + kT_i \end{cases} \tag{4}$$

We note that $\overline{\text{DBF}}_i(t)$ is discontinuous only at points $t = \ell T_i + D_i$ for some $\tau_i \in \Gamma$ and $\ell \in \{0, 1, \ldots, k\}$. It therefore follows that the left-hand side of the approximated version of Eq. 3, i.e., $\sum_{\tau_i \in \Gamma} \overline{\text{DBF}}_i(t)$, is discontinuous at no more than $((k+1) \times |\Gamma|)$ points, and is piecewise linear with slope at most $\sum_{\tau_i \in \Gamma} C_i/T_i$ elsewhere. Assuming $\sum_{\tau_i \in \Gamma} C_i/T_i \leq 1$, we therefore only need to evaluate the approximated version of Eq. 3 at the $\leq ((k+1) \times |\Gamma|)$ points of discontinuity; for constant k, this yields a polynomial-time sufficient EDF-schedulability test. It was also shown in [1] that any Γ that is deemed to not be EDF-schedulable by this sufficient test is guaranteed to actually not be EDF-schedulable upon a speed-$(k/(k+1))$ processor. Since k may take on any value, the Albers and Slomka polynomial-time sufficient test [1] is therefore an FPTAS (see Definition 1) for EDF schedulability analysis; as mentioned in Section 2, FPTAS's are considered to be the 'best' kind of approximation algorithm (that runs in polynomial time).

A scheme for efficient explainability. We can directly use the Albers and Slomka FPTAS [1] to design a scheme for efficient explainability, in the following manner. Suppose that we wish to explain, with a speedup factor[9] equal to $(1 + \delta)$, that some task system Γ is EDF-schedulable – we can do so by simply using the Albers and Slomka [1] over-approximation

[9] I.e., if we fail to explain the EDF-schedulability of Γ then Γ is in fact guaranteed to not be EDF-schedulable upon a speed-$(1/(1+\delta))$ processor.

of the demand bound function with $k \leftarrow \lceil 1/\delta \rceil$. Since the running time of the Albers and Slomka [1] test is polynomial in $|\Gamma|$ and k, i.e., polynomial in $|\Gamma|$ and $(1/\delta)$, it follows that this does indeed constitute a scheme for efficient explainability of EDF schedulability.

An improved scheme for efficient explainability. The scheme described in the previous paragraph is obtained by direct application of the FPTAS from [1]. But we can in fact improve on this for the purposes of efficient explainability: There is no particular reason why it is the *first* k steps of $\overline{\text{DBF}}_i$ that must be exact (not over-approximated), nor why the $\overline{\text{DBF}}_i$ of each task τ_i must be exact for the *same* number of steps (the same value of k).

Let us examine these ideas a bit further. Observe that in the function DBF_i the interval of the ℓth step, $\ell \geq 1$, is given by

$$\text{step}_i(\ell) = \big[(\ell - 1)T_i + D_i,\ \ell T_i + D_i\big). \tag{5}$$

For any $\mathcal{S}_i \subset \mathbb{N}_+$, let us define $\overline{\text{DBF}}_i(t, \mathcal{S}_i)$ to be the approximation to $\text{DBF}_i(t)$ that agrees with $\text{DBF}_i(t)$ over the intervals of the steps in $\mathcal{S}_i$, and is a linear over-approximation elsewhere:

$$\overline{\text{DBF}}_i(t, \mathcal{S}_i) = \begin{cases} 0, & \text{if } t < D_i \\ \text{DBF}_i(t), & \text{if } t \in \text{step}_i(\ell) \text{ for some } \ell \in \mathcal{S}_i \\ (T_i - D_i + t)\frac{C_i}{T_i}, & \text{otherwise} \end{cases} \tag{6}$$

The approximation $\overline{\text{DBF}}_i(t, \mathcal{S}_i)$ is illustrated in the right-most plot of Fig. 3. For any $\mathcal{S}_i$, we have $\text{DBF}_i(t) \leq \overline{\text{DBF}}_i(t, \mathcal{S}_i)$ for all t. By picking some set of steps $\mathcal{S}_i$ for each task τ_i and then replacing $\text{DBF}_i(t)$ by $\overline{\text{DBF}}_i(t, \mathcal{S}_i)$ in Eq. 3 we therefore get a sufficient EDF-schedulability test.

As with the Albers and Slomka [1] approximation, we note that $\overline{\text{DBF}}_i(t, \mathcal{S}_i)$ is discontinuous only at points $t = \ell T_i + D_i$ where $\ell \in \mathcal{S}_i \cup \{0\}$, and is piecewise linear with slope at most C_i/T_i elsewhere. It follows that $\sum_{\tau_i \in \Gamma} \overline{\text{DBF}}_i(t, \mathcal{S}_i)$ is discontinuous at most at $\sum_{\tau_i \in \Gamma}(|\mathcal{S}_i|+1)$ points, and is piecewise linear with slope at most $\sum_{\tau_i \in \Gamma} C_i/T_i$ elsewhere. Assuming $\sum_{\tau_i \in \Gamma} C_i/T_i \leq 1$, we therefore only need to evaluate the approximated version of Eq. 3 at the points of discontinuity (that are less HP(Γ)). If $\sum_{\tau_i \in \Gamma}(|\mathcal{S}_i| + 1)$ is bounded by a polynomial in the size of the task system, then we can check all points in the testing set in polynomial time.

For a fixed polynomial q, we let q-PDA be the subset of all task systems Γ for which there exists sets $\mathcal{S}_i \subset \mathbb{N}_+$ for each $\tau_i \in \Gamma$, such that $\sum_{\tau_i \in \Gamma} \overline{\text{DBF}}_i(t, \mathcal{S}_i) \leq t$ for all $0 \leq t \leq \text{HP}(\Gamma)$ and $\sum_{\tau_i \in \Gamma}(|\mathcal{S}_i| + 1) \leq q(|\Gamma|)$. By the reasoning above, q-PDA is in NP since a task set Γ can be verified to be in q-PDA in polynomial time if given the sets $\mathcal{S}_i$ as a certificate.

While it may require significant effort to find the sets $\mathcal{S}_i$, the q-PDA has the potential to allow much more efficient verification of solutions than the Albers and Slomka FPTAS. In fact, using q-PDA we may exponentially decrease the discontinous points that need to be checked in Eq. 3. The following simple example demonstrates this.

▶ **Example 4.** Consider the task set $\Gamma = \{\tau_1 = (1, 1, 2), \tau_2 = (\alpha/2, \alpha, 2\alpha)\}$, where α is some large even number. It can be readily confirmed that Γ is EDF-schedulable. However, using the approximation in Eq. 4 with $k < \alpha/2$ we get $\sum_{\tau_i \in \Gamma} \overline{\text{DBF}}_i(\alpha) = \alpha + 1/2$, and therefore we need $k \geq \alpha/2$ to establish that Γ is schedulable with the Albers and Slomka FPTAS, for a total of at least α discontinuous points to check. However, using q-PDA and the approximation in Eq. 6 with $\mathcal{S}_1 = \{\alpha/2\}$ and $\mathcal{S}_2 = \{1\}$, we can establish that Γ is schedulable by only checking 4 discontinuous points. In other words, Γ is in q-PDA with $q(n) = 2n$. ⌟

How much of EDF-schedulability that is covered by q-PDA depends very much on 'q, which makes it a type of parameterized approximation. This motivates the following definition, which extends the concept of FPTAS's for schedulability tests that was introduced (Definition 1) in Section 2.

▶ **Definition 5** (FPTVAS). *A fully polynomial-time* ***verification*** *approximation scheme (FPTVAS) for a schedulability analysis problem is an algorithm that, given as input an instance* Γ*, a parameter* $\delta > 0$*,* ***and a certificate****, returns "unschedulable" if* Γ *is unschedulable on a speed-1 processor, and returns "schedulable" if* Γ *is schedulable on a speed-*$(1/(1+\delta))$ *processor. Its running time is bounded by a polynomial in the two parameters* $|\Gamma|$ *and* $\left(\frac{1}{\delta}\right)$*.*

While our definition of FPTAS's for schedulability analysis problems (Definition 1) is essentially an instantiation of the preëxisting concept of FPTAS's from approximation theory [3, 34], the notion of FPT**V**AS's in Definition 5 above is, to our knowledge, novel – we are not aware of prior work in complexity theory that lifts the concept of FPTAS's from polynomial-time computation (i.e., the complexity class `P`) to polynomial-time verification (the class `NP`).

5.1 Extension to Multiprocessors

In contrast to uniprocessor EDF schedulability where an FPTAS is known to exist [1], no FPTAS is known for multiprocessor partitioned EDF schedulability – indeed, a lower bound of 1.5026 was recently obtained [28] on the speedup factor of the state-of-the-art partitioned EDF scheduling heuristic [9]. We cannot therefore simply use a preëxisting FPTAS to obtain an approximation scheme for efficient explainability of partitioned multiprocessor EDF schedulability. We can, however, extend the FPTVAS for uniprocessor EDF schedulability obtained above in the following manner to obtain an FPTVAS for partitioned EDF-schedulability of sporadic / synchronous periodic task systems. Suppose task system Γ is EDF-schedulable upon being partitioned upon an m-processor platform. For any fixed polynomial function q, the certificate of its schedulability would consist of

1. The partitioning $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$ of Γ amongst the m processors; and
2. For each partition Γ_j, $1 \leq j \leq m$, a certificate of its uniprocessor EDF schedulability. Such a certificate would comprise the sets $\mathcal{S}_i$ for each $\tau_i \in \Gamma_j$, together satisfying the constraints that $\sum_{\tau_i \in \Gamma_j} \overline{\text{DBF}}_i(t, \mathcal{S}_i) \leq t$ for all $0 \leq t \leq \text{HP}(\Gamma_j)$ and $\sum_{\tau_i \in \Gamma_j} (|\mathcal{S}_i| + 1) \leq q(|\Gamma_j|)$.

As in the uniprocessor case, this FPTVAS immediately implies an approximation scheme for efficient explainability of partitioned multiprocessor EDF schedulability.

6 Explainability Beyond Polynomial-time

There are many examples in real-time scheduling theory where not only polynomial-time algorithms are considered to be efficient, but also *pseudo-polynomial* time algorithms. (E.g., both response-time analysis [23] for FP-schedulability and the processor-demand approach [7] for EDF-schedulability of bounded-utilization systems are widely used schedulability analysis algorithms that have pseudo-polynomial running times.) The fundamental reason for why pseudo-polynomial time algorithms are often considered as being efficient in real-time scheduling is simply that the numerical parameters that appear in scheduling problems tend to have some direct physical meaning. For example, the parameters C_i, D_i and T_i of a sporadic task are supposed to represent physical time in some given unit, and we would therefore not expect to be given input instances with numerical parameters that are too large to be meaningful on a human timescale. Whether a pseudo-polynomial time algorithm is to be considered efficient certainly depends on what we expect the inputs to look like.

If we do accept pseudo-polynomial time as acceptably efficient in the context of solving problems, should we also accept pseudo-polynomial time as acceptably efficient for *verifying* solutions? We see no particular reason to not do so, if the problem is such that numerical values tend to be reasonably small. This motivates the following definition.

▶ **Definition 6** (pseudoNP)**.** *We can consider* pseudoP *as the complexity class of problems that can be solved by a pseudo-polynomial time algorithm. Equivalently,* pseudoP *contains the problems that would be in* P *if numbers were written in unary. Analogously, we define* pseudoNP *as the class of problems that would be in* NP *if numbers were written in unary. The class* pseudoNP *then contains the problems for which there exist pseudo-polynomially sized certificates that can be verified in pseudo-polynomial time.*

The classes pseudoP and pseudoNP do not fit so neatly into the hierarchy of complexity classes shown in Figure 1. We can see that pseudoP of course contains P, but must also be contained in EXP since the value of a (naturally represented) numerical parameter is at most exponential in the total length of the input, and therefore any polynomial in the value of the largest number cannot be larger than exponential in the input length. Further, any problem in EXP can be (artificially) transformed to allow a pseudo-polynomial time algorithm by padding input instances with large numbers. Hence, pseudoP intersects with all the classes up to EXP in Figure 1, but only completely contains P. By similar arguments, pseudoNP contains NP, is contained in NEXP and intersects with all other classes in Figure 1, including NEXP. The class pseudoNP captures an interesting property of problems, which to the authors' knowledge is not well-studied.

What problems can be found in pseudoNP that are neither in pseudoP nor in NP? Partitioned EDF-schedulability is again a good example. We know that uniprocessor EDF-schedulability of three-parameter sporadic (or synchronous periodic) tasks can be solved in pseudo-polynomial time if the utilization of task systems is bounded by some constant $c < 1$ (say, $c = 0.99$). [7] On the other hand, we know that partitioned EDF-schedulability is both NP- and coNP-hard, even with utilization bounded by any $c > 0$ [21], and thus is not in NP (unless NP = coNP). It is not difficult to see that this is the case even if we enforce a per-partition utilization bound of c. Also, the partitioned problem is NP-hard in the strong sense (as it generalizes Bin-Packing) and is therefore not in pseudoP (unless P = NP). However, the partitioned EDF-schedulability problem with a per-partition utilization bound of $c < 1$ *is* in pseudoNP. A certificate for this problem can simply consist of the partitioning, and the verifier can check that each partition has a utilization of at most c, and then directly verify the schedulability of each partition in pseudo-polynomial time using the standard PDA test [7].

An example of a problem that is *not* in pseudoNP is the uniprocessor EDF-schedulability problem for unbounded-utilization task systems. Since this problem is coNP-complete in the strong sense [20], it is (by definition) coNP-complete also if numbers are written in unary. Since the unary version is coNP-complete, it cannot be in NP (unless NP = coNP), and therefore the uniprocessor EDF-schedulability problem for unbounded-utilization task systems is not in pseudoNP.

7 Discussion

1. We have classified several multiprocessor schedulability analysis problems as efficiently explainable or not.
2. Using uniprocessor EDF schedulability analysis as a concrete example problem, we have developed multiple distinct methods for identifying efficient explainable sub-problems of problems that are not efficiently explainable. (We have also applied these methods to partitioned multiprocessor EDF schedulability analysis.)

3. We have extended the notion of FPTAS's, which are widely studied for approximation algorithms, from approximately solving a problem to verifying an approximate solution – this yields the novel concept of FPTVAS's (Definition 5), and extends the idea of approximation schemes to schemes for efficient explainablity.
4. We have extended the concept of pseudo-polynomial time algorithms, which are increasingly coming to be accepted as being 'efficient enough' for pre-run-time analysis (such as schedulability analysis), from efficient determination of schedulability to efficient explanation of schedulability, by defining the novel complexity class `pseudoNP` (Definition 6).

Our contributions in all these aspects are by no means complete or comprehensive – a large amount of work remains to be done in both classifying the explainability or non-explainability of other important schedulability-analysis problems, and in identifying efficiently-explainable sub-problems for those determined to not be efficiently explainable. Additionally, our explorations of the concepts of FPTVAS's (Definition 5) and `pseudoNP` (Definition 6) are quite basic – we believe both these concepts are potentially very meaningful and so merit considerable additional investigation.

We reiterate that we believe the use of formal, machine-verifiable proofs (such as those in Maida et al. [29]) is a promising way forward if we wish to use advanced techniques and recent developments in real-time systems research to explain schedulability to a certification authority (CA). Rather than trying to convince the CA that, say, an analysis for FP+fluid+split is sound and can be used to indirectly prove EDF-schedulability, such details can all be contained in the formal proof. The CA needs only to trust the proof assistant itself (e.g., Coq) and agree with the model of the system and basic definitions. The stated goal of the certification step (e.g., that the system is EDF-schedulable) is then guaranteed by the proof produced by the proof assistant, and can be trusted without even knowing the particular proof strategy employed. With this work we want to put forward the idea that techniques for enabling efficient explainability, as outlined in brief in this paper, could guide the creation of such machine-verifiable proofs that may indeed be *verified efficiently*, even as systems grow in size and complexity.

Finally, a discussion on verifiability of solutions to computational problems would not be complete without mentioning interactive proof systems. `IP` is the complexity class of problems where a *verifier* with only polynomial computational resources can be convinced (to an arbitrary degree of certainty) by an all-powerful *prover* that a valid solution exists to a given problem instance. In contrast to the static certificates –the explanations– considered in this paper, interactive proof systems work by letting the verifier and prover interactively exchange messages with each other, where the verifier challenges the explanations of the prover by asking specially-crafted (and randomized) questions. `IP` was shown to equal `PSPACE` (see Figure 1) in a landmark result [33], meaning that very many practical problems have such interactive proof systems. While interactive proof systems come with their own set of significant challenges, requiring interactive communication and accepting a small probability of incorrectly verified solutions, we believe that they have a place in explainability of real-time systems as well, and represent an interesting direction for future work.

References

1 K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 187–195, Catania, Sicily, July 2004. IEEE Computer Society Press.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.

3 Giorgio Ausiello, Alberto Marchetti-Spaccamela, Pierluigi Crescenzi, Giorgio Gambosi, Marco Protasi, and Viggo Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer Verlag, 1999.

4 T. P. Baker and A. Shaw. The cyclic executive model and Ada. In *Proceedings of the 9th Real-Time Systems Symposium (RTSS)*, pages 120–129, 1988.

5 T. P. Baker and A. Shaw. The cyclic executive model and Ada. *Real-Time Systems*, 1(1):7–25, 1989.

6 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.

7 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 (RTSS)*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

8 Sanjoy Baruah and Pontus Ekberg. Certificates of real-time schedulability. In *International Workshop on Explainability of Real-time Systems and their Analysis (ERSA)*, 2022.

9 Sanjoy Baruah and Nathan Fisher. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers*, 55(7):918–923, July 2006.

10 Kimaya Bedarkar, Mariam Vardishvili, Sergey Bozhko, Marco Maida, and Björn B. Brandenburg. From intuition to Coq: A case study in verified response-time analysis of FIFO scheduling. In *IEEE Real-Time Systems Symposium, RTSS 2022, Houston, TX, USA, December 5-8, 2022*, pages 197–210. IEEE, 2022. doi:10.1109/RTSS55097.2022.00026.

11 Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013. Project website: https://coq.inria.fr.

12 V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese. Polynomial-time exact schedulability tests for harmonic real-time tasks. In *Proceedings of the 34th Real-Time Systems Symposium (RTSS)*, pages 236–245, December 2013.

13 Sergey Bozhko and Björn B. Brandenburg. Abstract Response-Time Analysis: A Formal Foundation for the Busy-Window Principle. In Marcus Völp, editor, *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:24, Dagstuhl, Germany, 2020. doi:10.4230/LIPIcs.ECRTS.2020.22.

14 Björn B. Brandenburg and Mahircan Gül. Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations. In *Proceedings of the 37th IEEE Real-Time Systems Symposium (RTSS)*, pages 99–110, 2016. doi:10.1109/RTSS.2016.019.

15 Felipe Cerqueira, Felix Stutz, and Björn B. Brandenburg. PROSA: A case for readable mechanized schedulability analysis. In *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 273–284, 2016. doi:10.1109/ECRTS.2016.28.

16 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, third edition, 2009.

17 Robert Davis, Thomas Rothvoss, Sanjoy Baruah, and Alan Burns. Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. *Real-Time Systems: The International Journal of Time-Critical Computing*, 43(3):211–258, 2009.

18 Friedrich Eisenbrand and Thomas Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 2010.

19 P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks remains coNP-complete under bounded utilization. In *Proceedings of the 36th Real-Time Systems Symposium (RTSS)*, pages 87–95, 2015.

20 P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-complete. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 281–286, 2015.

21 Pontus Ekberg and Sanjoy Baruah. Partitioned scheduling of recurrent real-time tasks. In *Proceedings of the 42nd Real-Time Systems Symposium (RTSS)*, pages 356–367, 2021. doi:10.1109/RTSS52674.2021.00040.

22 Pontus Ekberg and Wang Yi. Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard. In *Proceedings of the 38th Real-Time Systems Symposium (RTSS)*, pages 139–146. IEEE Computer Society, 2017. doi:10.1109/RTSS.2017.00020.

23 M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, October 1986.

24 B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 214–223, Los Alamitos, October 1995. IEEE Computer Society Press.

25 B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 37(4):617–643, 2000.

26 J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the 10th Real-Time Systems Symposium (RTSS)*, pages 166–171. IEEE Computer Society Press, December 1989.

27 C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

28 Xingwu Liu, Zizhao Chen, Xin Han, Zhenyu Sun, and Zhishan Guo. Tighter bounds of speedup factor of partitioned EDF for constrained-deadline sporadic tasks. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 431–440, 2021. doi:10.1109/RTSS52674.2021.00046.

29 Marco Maida, Sergey Bozhko, and Björn B. Brandenburg. Foundational Response-Time Analysis as Explainable Evidence of Timeliness. In *Proceedings of the 34th Euromicro Conference on Real-Time Systems (ECRTS)*, volume 231 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:25, Dagstuhl, Germany, 2022. doi:10.4230/LIPIcs.ECRTS.2022.19.

30 Aloysius Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

31 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

32 Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, 4–6 May 1997.

33 Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, October 1992. doi:10.1145/146585.146609.

34 Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin-Heidelberg-New York-Barcelona-Hong Kong-London-Milan-Paris-Singapur-Tokyo, 2001.

35 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, May 2008.