A Closer Look at Pseudo-Polynomial Time and its Use in Real-Time Scheduling Theory

Sanjoy Baruah[1] and Pontus Ekberg[2]

[1] Washington University in St. Louis <baruah@wustl.edu>
[2] Uppsala University <pontus.ekberg@it.uu.se>

Abstract. Amongst Wang's contributions to real-time computing are those in which he and his collaborators have pushed the boundaries of pseudo-polynomial time schedulability analysis: developing expressive task models for which schedulability analysis can be done using algorithms that have pseudo-polynomial running time. In this note we revisit these contributions in the light of more recent work that provides additional context within which to view Wang's results, and investigate further directions in which his contributions can be extended.

1 Wang's contributions to real-time schedulability analysis

Wang and his research collaborators have played a major role in obtaining solutions to the **schedulability analysis** problem, one of the foundational problems studied in real-time scheduling theory. The schedulability analysis problem may be described as follows:

> GIVEN (i) the specifications of the computational demands of, and the timing constraints upon, a workload; (ii) the platform upon which this workload is to be executed; and (iii) the run-time scheduling algorithm that will be used, DETERMINE (prior to run-time) whether the workload is guaranteed to always meet all its timing constraints in all runtime scenarios permissible by the specification.

The workload is often modeled as a collection (called a *task system*) of independent recurrent tasks executing upon a shared computing platform. In the widely-studied *sporadic task model*, for instance, each task τ_i is characterized by a *worst-case execution time* (WCET) parameter C_i, a *relative deadline* D_i, and a *period* T_i (each $\in \mathbb{N}$). Each such τ_i releases a potentially unbounded sequence of *jobs*, with successive job releases separated by a duration no smaller than T_i units, during any particular execution of the system; each job released by τ_i may need to execute for up to C_i time units and must complete execution within a duration D_i following its release time. The shared platform upon which the system is to be implemented may be a *uniprocessor* one or it may comprise *multiple processors*

that may be *identical* or *heterogeneous*; furthermore, processors are said to be *preemptive* if an executing job can be interrupted and have its execution resumed later at no cost or penalty, or non-preemptive otherwise. A variety of scheduling algorithms have been studied, amongst the most common of which are the *Fixed-Priority* (FP) [26, 27, 31] and *Earliest-Deadline First* (EDF) [17, 31] algorithms. (FP and EDF scheduling are discussed further in Sections 4.1 and 4.2.)

In earlier times the limited availability of computation meant that schedulability analysis algorithms were restricted to have worst-case running times that are low-degree polynomials in the size of their inputs in order to be considered "efficient." A famous early example of such an efficient algorithm is the utilization-based FP schedulability test [31] for implicit-deadline sporadic task systems[3] upon preemptive uniprocessors. This utilization-based test is approximate rather than exact in the sense that it is sufficient but not necessary; indeed, polynomial-time exact schedulability-analysis algorithms are scarce (one exception is the utilization-based EDF schedulability test for implicit-deadline sporadic task systems upon preemptive uniprocessors). This is not surprising: most schedulability-analysis problems, including FP and EDF schedulability analysis of sporadic task systems upon preemptive or non-preemptive uniprocessor or multiprocessors, have been shown to be NP- or coNP-hard and hence unlikely to admit to exact polynomial-time schedulability tests (e.g., [5, 7, 18, 19, 21, 23, 29, 30]).

As computing capabilities increased over time, schedulability analysis algorithms with *pseudo-polynomial* running times came to be considered efficient. Early examples of efficient algorithms of this kind include Response-Time Analysis (RTA) [3, 16, 26, 27, 42, 43], an exact FP-schedulability test for constrained-deadline sporadic task systems upon preemptive uniprocessors, Processor-Demand Analysis (PDA) [6, 7, 29], an EDF-schedulability test for sporadic task systems, also upon preemptive uniprocessors, that is pseudo-polynomial time for bounded-utilization systems.

The emergence of this consensus within the safety-critical real-time computing community that pseudo-polynomial running time equates to efficiency spurred the real-time scheduling theory community to ask: *what is the most general workload model* for which preemptive uniprocessor schedulability analysis remains doable in pseudo-polynomial running time? Wang and his team made very significant contributions in this area, proposing a variety of increasingly expressive models – see Fig. 1 – that demarcate a precise boundary between efficiency and intractability by identifying the most expressive models for which schedulability-analysis can be done in pseudo-polynomial time and the least expressive models

[3] Some terminology: an *implicit-deadline* sporadic task system is a sporadic task system in which each task τ_i satisfies the additional constraint that $D_i = T_i$, while each task in a *constrained-deadline* sporadic task system satisfies $D_i \leq T_i$. The ratio (C_i/T_i) of task τ_i is called its *utilization*; a *bounded-utilization* system is one for which the cumulative utilization is a priori bounded by some constant that is strictly smaller than 1. A *utilization-based* schedulability test determines schedulability by comparing the cumulative utilization of all the tasks in the system to some limit.

for which it cannot (assuming $\mathsf{P} \neq \mathsf{NP}$). In particular, they established that EDF schedulability analysis of bounded-utilization task systems upon preemptive uniprocessors can be done in pseudo-polynomial time for task systems represented using all of the models depicted in Fig. 1 with the exception of the two most general ones (EDRT [39] and Timed Automata with Tasks [24]). For FP-schedulability upon preemptive uniprocessors, pseudo-polynomial time algorithms are known to exist only for the Liu & Layland [31] and constrained-deadline sporadic [32] task models. The existence of pseudo-polynomial time FP-schedulability tests is still open for the non-cyclic GMF model [34], while all other models in Fig. 1 were shown to be strongly coNP-hard by Stigge [37], and so do not have such tests unless $\mathsf{P} = \mathsf{NP}$.

Our perception of what constitutes an efficient algorithm for schedulability analysis continues to evolve [1], and some recent research in the real-time scheduling community is exploring ways of moving beyond the pseudo-polynomial time barrier. Many such investigations (see, e.g., [4, 11, 12, 15, 46]) seek to transform a schedulability-analysis problem to some other form such as an integer linear program (ILP), which can then be solved by an ILP solver. Although solving an integer linear program is itself computationally intractable (it is strongly NP-complete to decide if any feasible solution exists), excellent off-the-shelf solvers exist that, by incorporating a combination of expert techniques, special-purpose heuristics, and highly optimized implementation, are able to handle surprisingly large problem instances in reasonable amounts of time.

Does this emerging acceptance of ILP representations as adequately efficiently solvable imply that all schedulability-analysis problems can now be considered to be efficiently solvable? By no means: while it is known that solving an ILP (specifically, determining whether a feasible solution exists), is an NP-complete problem as mentioned above, computational complexity theory defines various additional complexity classes that are widely presumed to encompass problems beyond NP or coNP (akin to NP being believed to contain problems beyond P – i.e., unsolvable by polynomial-time algorithms). Demonstrating the hardness of a schedulability analysis problem for one of these complexity classes would strongly suggest that it cannot be efficiently solved, even with a highly optimized ILP solver. The complexity class $\mathsf{NP}^{\mathsf{NP}}$ (commonly denoted as Σ_2^{P}) is an example: it is widely conjectured that $\mathsf{NP} \subsetneq \Sigma_2^{\mathsf{P}}$ and $\mathsf{coNP} \subsetneq \Sigma_2^{\mathsf{P}}$. Woeginger [45] explains the implications: "*If you hit a Σ_2^{P}-complete problem, then there is no way of formulating it (in polynomial time) as an integer program (of polynomial size)*" and goes on to state that "*Σ_2^{P}-complete problems are much, much, much, much, much harder than any problem [...] that can be attacked via ILP solvers.*"

Summarizing the discussion above, until quite recently notions of efficiency in schedulability analysis appear to have been centered around these three beliefs:

1. Schedulability-analysis algorithms that have pseudo-polynomial running times are accepted as being efficient; Wang and his colleagues have made significant contributions in developing very general task models for which such efficient algorithms exist for preemptive uniprocessor EDF schedulability-analysis.

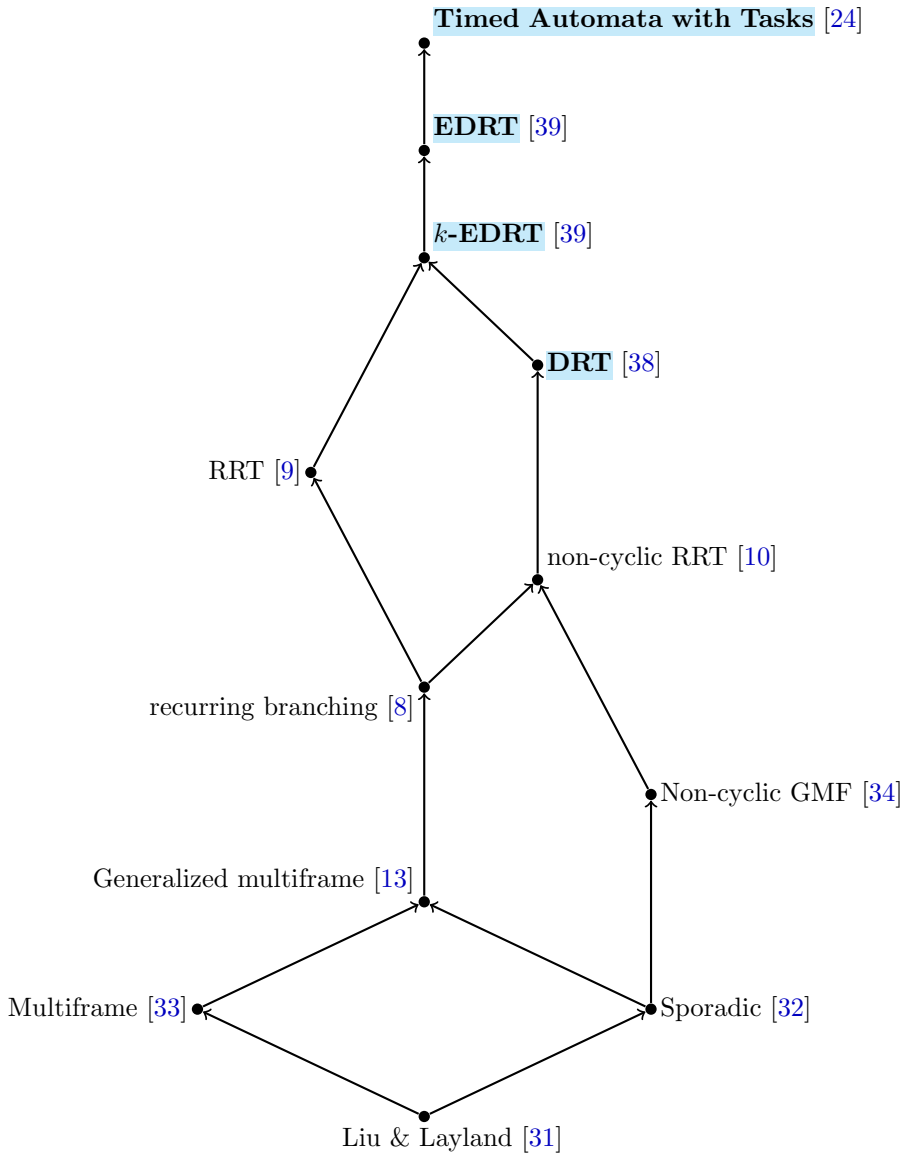

Fig. 1. Models for recurrent real-time tasks (the highlighted models have been proposed by Wang and his collaborators). Arrows point from less expressive models to more expressive ones; models not connected by a path have incomparable expressiveness. Wang and his collaborators proved that the k-EDRT model is the most general one for which preemptive uniprocessor EDF-schedulability analysis can be done in pseudo-polynomial time, and the sporadic model the most general one for which preemptive uniprocessor FP-schedulability analysis can be done in pseudo-polynomial time. This figure is adapted from [41].

2. More recently, "polynomial-time + ILP-solver" algorithms – algorithms that have polynomial running time and are additionally allowed to make calls to an ILP solver – are also coming to be considered efficient (although less so than algorithms with pseudo–polynomial running times without needing an ILP solver).
3. Showing a problem to be hard for a complexity class believed to contain problems not in NP or coNP shows it to be truly intractable (as stated above, it is considered to be "much, much, much, much, much harder" [44] than any problem in NP or coNP).

However, recent research [1, page 4] somewhat blurs the third of these beliefs:

Observation 1 (From [1]) *If* C *is a complexity class contained in* EXP *and* C *is closed under polynomial-time reductions, then there exist* C*-complete problems with pseudo-polynomial time solutions.* □

(Here EXP is the exponential-time analog of complexity class P: it is the complexity class of all decision problems that are solvable using exponential-time algorithms. A complexity class C is said to be closed under polynomial-time reductions if whenever problem X is in C and problem Y can be reduced to problem X in polynomial time, then problem Y is also in C.)

It immediately follows from Observation 1 that showing a problem to be Σ_2^{P}-hard does not necessarily rule out the possibility of there being a pseudo-polynomial-time algorithm for solving it – pseudo-polynomial time algorithms may exist for solving problems that belong to any complexity class $\subseteq$ EXP. This fact motivates us to also take a closer look at the first of these three beliefs: we make a case that not all pseudo-polynomial algorithms should be considered equally tractable, and, in Section 2 below, propose a finer-grained classification within the class of pseudo-polynomial time algorithms.

2 A finer-grained notion of pseudo-polynomial time

The fundamental reason for why pseudo-polynomial time algorithms are considered efficient for scheduling problems is that the numerical parameters in such problems tend to have a direct physical interpretation, often as a measure of time, and therefore do not grow without bound in meaningful instances. Even so, numerical values found in input instances can quickly grow much larger than the instance size as measured in bits. In real-time scheduling problems, time may be measured in milliseconds, microseconds, or even CPU clock cycles, and numerical values can easily range into the thousands or millions even for "small" instances. If we expect to see instances where the largest numerical value N is much larger than the total input size n, then clearly the efficiency of a pseudo-polynomial time algorithm is often dominated by its scaling with respect to N rather than to n. Still, in the real-time scheduling theory literature at least, we seldom see

much attention paid to which polynomial scaling applies with respect to N.[4] Motivated by the above observations we define the following more fine-grained notion of pseudo-polynomial time, where the particular dependence (e.g., linear or quadratic) on the largest numerical parameter is made explicit.[5]

Definition 1 (Pseudo-f). *An algorithm's running time is pseudo-f if it is $O(n^k \times f(N))$, where n is the size of the representation of the problem instance, N is its largest numerical parameter value, k is a constant, and f is some function.* □

It follows from this definition that an algorithm's running time is pseudo-polynomial if and only if it is pseudo-f for some polynomial f. We will say that an algorithm with pseudo-f running time is *pseudo-linear* if $f(N) \in O(N)$, *pseudo-quadratic* if $f(N) \in O(N^2)$, etc.

A simple observation to make is that the notion of pseudo-f is unaffected by any logarithmic factors in f:

Theorem 2. *If $f(N) = \log^a(N) \times N^b$ and $g(N) = N^b$ for constants a and b, then an algorithm's running time that is pseudo-f is also pseudo-g.*

Proof. With any standard representation of numbers we have $\log(N) \in O(n)$, where N is the largest numerical parameter and n is the size of the input. If the running time is pseudo-f, then for some k it is

$$O(n^k \times f(N)) = O(n^k \times \log^a(N) \times N^b) \subseteq O(n^{k+a} \times N^b) = O(n^{k+a} \times g(N))$$

and is therefore also pseudo-g. □

Despite being insensitive to logarithmic factors, Definition 1 provides meaningful distinctions not just between algorithm running times, but also between the complexity of problems. We see below that any difference in polynomial degree between f and g differentiates pseudo-f from pseudo-g, and the problems that can be solved correspondingly.

Theorem 3. *For all $a > b > 0$, there are problems that can be solved in time $O(\mathsf{poly}(n) \times N^a)$, but not in time $O(\mathsf{poly}(n) \times N^b)$, where n is the size of the input and N is the value of its largest numerical parameter.*

Proof. Let $c = (a+b)/2$ and let S be a problem that can be solved in time $O(2^{an})$, but not in time $O(2^{cn})$. (We know that such S exist by the Time Hierarchy Theorem [25].) Let

$$S' \stackrel{\text{def}}{=} \left\{(x, y) \mid x \in S \wedge y = 2^{|x|}\right\}$$

[4] The authors of this paper are most certainly guilty of having, in multiple previous works, declared success as soon as a pseudo-polynomial time algorithm has been found for the problem under study.

[5] This definition is a generalization of one that appeared in [1].

and we will see that S' is the problem we are looking for. It is evident that y is the largest number in any instance $(x, y) \in S'$.

First, we can solve S' in time $O(\mathsf{poly}(n) \times N^a)$ as follows. Given (x, y) we check that $y = 2^{|x|}$ and if so we use the $O(2^{an})$ algorithm for S to check that $x \in S$ in time $O(2^{a|x|}) = O(y^a)$.

Second, we cannot solve S' in time $O(\mathsf{poly}(n) \times N^b)$. Suppose for the purpose of contradiction that we could, then we could solve S as follows. Given an instance x of S we reduce to instance $(x, 2^{|x|})$ of S' in polynomial time. Then we use the algorithm for S' to determine if $(x, 2^{|x|}) \in S'$ (and hence $x \in S$) in time

$$O\big(\mathsf{poly}(|x| + |y|) \times y^b\big) = O\big(\mathsf{poly}(|x|) \times 2^{b|x|}\big) \subsetneq O\big(2^{c|x|}\big),$$

thereby contradicting the assumption that S cannot be solved in time $O(2^{cn})$. □

Corollary 1. *More problems can be solved in pseudo-quadratic time than in pseudo-linear time, and in pseudo-cubic time than in pseudo-quadratic time, etc.* □

3 A notion of scaling invariance of pseudo-polynomial time

In addition to being able to differentiate pseudo-polynomial time algorithms based on their polynomial dependence on the numerical values found in inputs (as in Definition 1), we would also like to differentiate pseudo-polynomial time algorithms based on whether their running time is invariant to simple scaling of such values.

Preferably, a pseudo-polynomial time algorithm should take longer time to run only when there are more complicated relationships between the values of numerical parameters in the input, and not when those numerical parameters grow without their internal relationships changing. This is a natural property of many problems with pseudo-polynomial time algorithms. For example, it is not inherently harder to evenly partition boxes of sizes 1000, 2000, and 3000, than to partition boxes of sizes 1, 2, and 3.

In the context of scheduling problems, if numerical parameters represent time, this translates into whether a pseudo-polynomial time algorithm is sensitive to the *unit* used for specifying timing parameters. For example, we would like such an algorithm to have the same running time for a particular input instance regardless of whether time is given in units of milliseconds, microseconds or $1/17$'ths of a second, assuming that everything else remains unchanged and that numerical values are still integer.

The following definition attempts to capture this notion of invariance of a pseudo-polynomial time algorithm.[6]

[6] This definition (though not the analysis following it), appeared in [1]. There it was called *robust* pseudo-polynomial time, but we prefer the term used here as "robust" is an overloaded word.

Definition 2 (Scale-invariant pseudo-polynomial time). *An algorithm is* scale-invariant *pseudo-polynomial if it runs in time that is polynomial in n and in N/G, where n is the size of the input, N is its largest numerical parameter, and G is the greatest common divisor of all its numerical parameters.* □

By combining this definition with Definition 1 we will also refer to algorithms as being *scale-invariant pseudo-linear* if the dependence on N/G is linear, etc.

We note that it seems useful to retain some flexibility regarding which numerical parameters to include in the computation of the greatest common divisor G in Definition 2. For instance, in a multiprocessor scheduling problem with a multitude of numerical parameters denoting time (deadlines, periods, execution times) and a single numerical parameter denoting the number of processors, it may make sense to include only the parameters that share a unit (i.e., the parameters representing time), but not the processor count, when calculating G.

Not all pseudo-polynomial time algorithms are scale invariant, and in the following we see that this differentiates computational problems as well.

Theorem 4. *There are problems that can be solved in pseudo-polynomial time, but not in scale-invariant pseudo-polynomial time.*

Proof. This follows from a simple padding argument. Let S be an EXP-complete problem that is naturally encoded without using numbers, say *Generalized Go*, which was proven to be EXP-complete by Robson [35].[7] Say that S can be solved in time

$$O\left(2^{n^k}\right). \tag{1}$$

Then let

$$S' \stackrel{\text{def}}{=} \left\{(x, y) \mid x \in S \wedge y = 2^{|x|^k}\right\}$$

be another problem created by padding instances x of S by exponentially large numbers y, where k is the constant from Eq. 1. A reduction from S to S' is clearly in polynomial time, so S' is EXP-hard. We can also solve S' in pseudo-polynomial time by first checking that $y = 2^{|x|^k}$ and then use the algorithm for S to check $x \in S$ in time $O(2^{|x|^k}) = O(y)$.

However, since we consider instances of S to have no numbers in them, the only number in an instance (x, y) of S' is y. Therefore y/G, where G is the GCD of numbers in (x, y) is simply $y/y = 1$. A *scale-invariant* pseudo-polynomial time algorithm for S' must therefore run in time $\mathsf{poly}(n, N/G) = \mathsf{poly}(|x| + |y|, y/y) = \mathsf{poly}(|x| + |y|)$, which is impossible by the Time Hierarchy Theorem since S' is EXP-hard. □

[7] Sorry, Wang, that's with traditional Japanese rules. Generalized Go with Chinese rules is not known to be EXP-complete. [36]

In real-time scheduling problems, a lack of scale invariance of a schedulability test could stem from a reliance on discrete schedules (e.g., that jobs can only be released at integer time points or executed for an integer number of time units at a time)[8], or from scheduling decision based on things such as absolute time points or number of time units executed. In other cases, however, there seems to be little reason for why a pseudo-polynomial time schedulability test could not in principle be made scale invariant. The following natural property seems self-evident.

Proposition 1. *Let $\mathcal{J}$ be a sequence of independent jobs, each specified by release time, deadline and execution time, in a setting where scheduling does not have to be discrete. Let $\mathcal{J}$ be scheduled by a scheduler that, if it bases any scheduling decisions on the jobs' parameters or remaining execution times, only does so on the relative ordering of those values (e.g., as with EDF, which considers the relative ordering of deadlines). If $\mathcal{J}'$ is another sequence of jobs that differs from $\mathcal{J}$ only in that its jobs have their parameters uniformly scaled by some factor, then the scheduler would meet all deadlines of $\mathcal{J}$ if and only if it would meet all deadlines of $\mathcal{J}'$.*

Most common scheduling algorithms fit the above description and would simply generate scaled, but otherwise identical, schedules from job sequences $\mathcal{J}$ and $\mathcal{J}'$. This is easily seen to include schedulers such as EDF, FP and FIFO.

The above proposition immediately generalizes to task systems where, if task system Γ' is a uniformly scaled version of task system Γ, then any job sequence released by Γ' is a uniformly scaled version of a job sequence released by Γ. This includes basic periodic and sporadic task systems.

Corollary 2. *Let Γ be a task system of periodic or sporadic tasks not limited to discrete behaviors, and let Γ' be an identical task system, but with uniformly scaled task parameters. Then schedulers such as EDF, FP and FIFO would correctly schedule Γ if and only if they correctly schedule Γ'.*

We note that if a scheduling algorithm has the above properties of producing identical, but scaled, schedules on scaled inputs for a given setting, then a schedulability test for it could be made scale invariant by simply dividing every task parameter by the GCD before applying the test.

4 A (very) brief survey of some schedulability algorithms

In this section we take a quick look at three schedulability problems and corresponding schedulability tests from the real-time scheduling literature. These

[8] In some settings this is a very important distinction. For example, in the popular *mixed-criticality* scheduling model some task systems are schedulable if job releases are restricted to integer time points, but become impossible to schedule correctly if they are not (see [22, Claim 2]).

tests are pseudo-polynomial time algorithms; we classify them further on the basis of Definitions 1 and 2.

4.1 Response-time analysis

Response-time analysis (RTA) [3, 16, 26, 27, 42, 43] is the classical schedulability test for Fixed-Priority (FP) scheduling, which computes the worst-case response time of each task and compares it with the deadline. In the basic setting of independent sporadic or synchronous periodic tasks with constrained deadlines scheduled on a preemptive uniprocessor, if the smallest positive fixed point R_i to

$$R_i = C_i + \sum_{\tau_j \in \mathrm{hp(i)}} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j$$

is such that $R_i \leq T_i$, where $\mathrm{hp}(i)$ is the set of tasks with higher priority than τ_i, then R_i is the worst-case response time of task τ_i. The fixed-point iteration

$$R_i^{n+1} = C_i + \sum_{\tau_j \in \mathrm{hp(i)}} \left\lceil \frac{R_i^n}{T_j} \right\rceil \times C_j \tag{2}$$

is known to find the smallest positive fixed point R_i if started with an initial value that is at most R_i. Since for schedulability we are interested only in knowing if $R_i \leq D_i$ for each task, we can immediately stop the fixed-point iteration if we encounter a $R_i^n > D_i$. Assuming integer task parameters, the number of iterations of Eq. 2 before converging or reaching $R_i^n > D_i$ is therefore bounded by D_i. From this follows the well-known result that determining whether $R_i \leq D_i$ can be done in time $O(|\mathrm{hp}(i)| \times D_i)$ using RTA, and the entire task system's schedulability can then certainly be determined in time $O(n^2 \times N)$, where n is the size of the task system and N the largest task parameter value. From Definition 1 we have then that RTA is not only pseudo-polynomial in this setting, but is in fact pseudo-linear.

Just a little closer inspection of Eq. 2 reveals that the number of iterations is also bounded by $\sum_{\tau_j \in \mathrm{hp(i)}} \lceil D_i/T_j \rceil$, which is $O(n \times N/G)$, where G is the greatest common divisor of all numerical task parameters. From Definition 2, we have then that RTA is also scale-invariant pseudo-linear.

We note that RTA in the form above does not work for task systems with arbitrary deadlines (where we may have $D_i > T_i$). It has been extended by Lehoczky [28] to work in such settings, but that variant of RTA takes exponential time and it is not known if the schedulability problem with arbitrary deadlines can be solved in pseudo-polynomial time at all in the general setting.

4.2 Processor-demand analysis

Processor-demand analysis (PDA) [6, 7] is a schedulability test for Earliest Deadline First (EDF) scheduling of independent synchronous periodic or sporadic

tasks on a preemptive uniprocessor. EDF-schedulability in this setting can be solved in linear time for task systems with implicit deadlines using a simple utilization test [31], but with constrained or arbitrary deadlines PDA is the canonical test. According to PDA, a task system Γ is EDF-schedulable if and only if

$$\sum_{\tau_i \in \Gamma} \left(\max \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1, 0 \right) \right) \times C_i \leq t \tag{3}$$

for all time-window sizes $t \in \mathcal{T}(\Gamma)$, where $\mathcal{T}(\Gamma)$ is the so-called *test set* for Γ.

The EDF-schedulability problem is strongly coNP-complete [20], and so the size of the test set is not pseudo-polynomial in general. However, if the task system's utilization, $\sum_{\tau_i \in \Gamma} C_i/T_i$, is bounded by some constant c, where $c < 1$, then a valid test set is

$$\mathcal{T}(\Gamma) = \left\{ 0, 1, \ldots, \frac{c}{1-c} \times \max_{\tau_i \in \Gamma} \{T_i - D_i\} \right\}.$$

In this case we have that $|\mathcal{T}(\Gamma)|$ is of pseudo-polynomial size, and indeed it is easy to see that PDA in this case meets the requirements of Definition 1 to be pseudo-linear. Bounding the utilization to something less than 100% (using, say, $c = 0.99$) is not particularly limiting, and good engineering practice is anyway to not utilize the processor to its limits at 100%.

We can note that the above pseudo-polynomial version of PDA is not scale invariant, as per Definition 2, as scaling all task parameters by some integer factor would cause the test set to grow by the same factor. A common optimization applied to PDA is to include only the points t where $t \equiv D_i \pmod{T_i}$ for some $\tau_i \in \Gamma$ in the test set (those are exactly the points at which the left-hand side of Eq. 3 is discontinuous). By applying this optimization, the size of the test set remains the same if a uniform scaling is applied to the task parameters, and PDA becomes scale-invariant pseudo-linear as per Definition 2.

The fact that both PDA and RTA (as seen in Section 4.1) are scale-invariant pseudo-linear in common settings may go some way to explain their practical efficiency.

4.3 Semi-clairvoyant scheduling of mixed-criticality tasks

We will now take a look at a more recent schedulability test for what is called semi-clairvoyant scheduling [2] of mixed-criticality tasks with graceful degradation [14]. In this scheduling problem we have a set of regular sporadic tasks, but each task is marked as having either high or low criticality. The tasks with high criticality come with alternative implementations that may have longer execution times (for handling extraordinary or critical situations), while the tasks with low criticality come with alternative implementations that may have lower execution times (as

lower-quality backup implementations that consume less resources in critical scenarios). The arrival of any job from a high-criticality task may trigger a critical scenario and result in the use of, from that time point onwards, the alternative implementations of all new jobs from all tasks.

In [14], an exact EDF-schedulability test was described for this scheduling problem, which builds on PDA as described in Section 4.2 and also assumes bounded utilization. Instead of quantifying over a single pseudo-polynomially sized test set, as in PDA, this test requires two universal quantifiers and has the following general form (we skip the details of the expressions here, they are available in [14, Theorem 7]):

$$\forall t \in A, \forall s \in B : \mathrm{dbf}(t, s) \leq t,$$

where A and B are pseudo-linearly sized sets, and $\mathrm{dbf}(t, s)$ is a simple function that can be evaluated in linear time (essentially a generalization of the left-hand side of Eq. 3).

The extra quantification, compared to PDA, serves the purpose of testing all the possibilities as to which job triggers critical behavior, and there seems to be no obvious way to get rid of it. The effect is that this schedulability test is not pseudo-linear, though it can be seen to be pseudo-quadratic.[9] Some sort of additional quantification over time points in a time interval, as seen here, is not an uncommon pattern of schedulability tests and seems a likely cause of a number of slower-than-pseudo-linear pseudo-polynomial time schedulability tests in the literature.

The test described above is also not scale invariant, as in Definition 2, because one of the sets that is quantified over grows with an increased uniform scaling of task parameters. It is not trivial to see in the proofs presented in [14] that it does not have to be so, and it may seem as if the test strategy employed inherently lacks scale invariance. However, a higher-level reasoning similar to that of Corollary 2 shows that EDF-schedulability must be unaffected if all task parameters were uniformly scaled in this setting, and so the test can in fact easily be made scale invariant by first dividing all parameters by the GCD.

5 Concluding remarks

The real-time scheduling community has long considered the availability of pseudo-polynomial time algorithms as one of the important delimiters between

[9] A pseudo-quadratic test is certainly much better than a full-blown exponential-time test, but we should also expect that this test can be much slower than the pseudo-linear PDA that it builds upon. The authors of [14] did however not dwell on this fact, and instead happily reported the success of finding a pseudo-polynomial time test for the problem without further discussion. The authors of [14] are also the authors of the current paper. *Nostra culpa.*

tractable and intractable problems. This largely holds true because numerical parameters in such problems tend to represent physical time, and therefore do not grow without bound. Wang has been part of several important works that have pushed the boundary of what can be analyzed in pseudo-polynomial time, and via complexity analysis found where that boundary lies exactly (for example in [20, 38–40]).

Recent research [1] has investigated some ways in which the pseudo-polynomial time boundary can be blurred. In this note we have looked closer at some of those ways, in particular by further classifying pseudo-polynomial time algorithms based on their particular polynomial dependency on the numerical variables, and based on whether their running time is invariant to scaling of those parameters. We anticipate that the findings reported here may spur additional research building upon Wang's works by extending his ideas and approaches to our finer-grained classification and by incorporating scale-invariance.

References

1. Agrawal, K., Baruah, S., Ekberg, P.: Rethinking tractability for schedulability analysis. In: 2023 IEEE Real-Time Systems Symposium (RTSS). pp. 1–12. IEEE Computer Society, Los Alamitos, CA, USA (Dec 2023). https://doi.org/10.1109/RTSS59052.2023.00011
2. Agrawal, K., Baruah, S., Burns, A.: Semi-clairvoyance in mixed-criticality scheduling. In: Proceedings of the Real-Time Systems Symposium (RTSS). pp. 458–468 (Dec 2019)
3. Audsley, N.C., Burns, A., Davis, R.I., Tindell, K.W., Wellings, A.J.: Fixed priority preemptive scheduling: An historical perspective. Real-Time Systems **8**, 173–198 (1995)
4. Baruah, S.: An ILP representation of a DAG scheduling problem. Real-Time Systems: The International Journal of Time-Critical Computing (2021)
5. Baruah, S., Howell, R., Rosier, L.: 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)
6. Baruah, S., Howell, R., Rosier, L.: Feasibility problems for recurring tasks on one processor. Theoretical Computer Science **118**(1), 3–20 (1993)
7. Baruah, S., Mok, A., Rosier, L.: Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of the 11th Real-Time Systems Symposium. pp. 182–190. IEEE Computer Society Press, Orlando, Florida (1990)
8. Baruah, S.: Feasibility analysis of recurring branching tasks. In: Proceedings of the Tenth EuroMicro Workshop on Real-time Systems. pp. 138–145. Berlin, Germany (June 1998)
9. Baruah, S.: Dynamic- and static-priority scheduling of recurring real-time tasks. Real-Time Systems: The International Journal of Time-Critical Computing **24**(1), 99–128 (2003)
10. Baruah, S.: The non-cyclic recurring real-time task model. In: Proceedings of the Real-Time Systems Symposium. IEEE Computer Society Press, San Diego, CA (2010)

11. Baruah, S., Bini, E.: Partitioned scheduling of sporadic task systems: an ILP-based approach. In: Proceedings of the 2008 Conference on Design and Architectures for Signal and Image Processing (2008)
12. Baruah, S., Bonifaci, V., Bruni, R., Marchetti-Spaccamela, A.: ILP-based approaches to partitioning recurrent workloads upon heterogeneous multiprocessors. In: Proceedings of the 2016 28th EuroMicro Conference on Real-Time Systems. ECRTS '16, IEEE Computer Society Press, Toulouse (France) (2016)
13. Baruah, S., Chen, D., Gorinsky, S., Mok, A.: Generalized multiframe tasks. Real-Time Systems: The International Journal of Time-Critical Computing **17**(1), 5–22 (July 1999)
14. Baruah, S., Ekberg, P.: Graceful Degradation in Semi-Clairvoyant Scheduling. In: Brandenburg, B.B. (ed.) 33rd Euromicro Conference on Real-Time Systems (ECRTS 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 196, pp. 9:1–9:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.ECRTS.2021.9
15. Baruah, S.K., Bonifaci, V., Bruni, R., Marchetti-Spaccamela, A.: ILP models for the allocation of recurrent workloads upon heterogeneous multiprocessors. Journal of Scheduling (Dec 2018). https://doi.org/10.1007/s10951-018-0593-x, `https://doi.org/10.1007/s10951-018-0593-x`
16. Burns, A., Tindell, K., Wellings, A.: Effective analysis for engineering real-time fixed priority schedulers. IEEE Transactions on Software Engineering **21**(5), 475–480 (May 1995)
17. Dertouzos, M.: Control robotics : the procedural control of physical processors. In: Proceedings of the IFIP Congress. pp. 807–813 (1974)
18. Eisenbrand, F., Rothvoss, T.: Static-priority real-time scheduling: Response time computation is NP-hard. In: Proceedings of the Real-Time Systems Symposium. IEEE Computer Society Press, Barcelona (December 2008)
19. Eisenbrand, F., Rothvoß, T.: EDF-schedulability of synchronous periodic task systems is coNP-hard. In: Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (January 2010)
20. Ekberg, P., Yi, W.: Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-complete. In: 2015 27th Euromicro Conference on Real-Time Systems. pp. 281–286 (2015)
21. Ekberg, P.: Models and Complexity Results in Real-Time Scheduling Theory. Ph.D. thesis, Uppsala University (2015)
22. Ekberg, P., Yi, W.: A note on some open problems in mixed-criticality scheduling. In: Proceedings of the 6th International Real-Time Scheduling Open Problems Seminar (RTSOPS) (2015)
23. Ekberg, P., Yi, W.: 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. pp. 139–146. IEEE Computer Society (2017). https://doi.org/10.1109/RTSS.2017.00020, `https://doi.org/10.1109/RTSS.2017.00020`
24. Fersman, E., Pettersson, P., Yi, W.: Timed automata with asynchronous processes: Schedulability and decidability. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 67–82. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
25. Hartmanis, J., Stearns, R.E.: On the computational complexity of algorithms. Transactions of the American Mathematical Society **117**, 285–306 (1965)
26. Joseph, M., Pandya, P.: Finding response times in a real-time system. The Computer Journal **29**(5), 390–395 (Oct 1986)

27. Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proceedings of the Real-Time Systems Symposium - 1989. pp. 166–171. IEEE Computer Society Press, Santa Monica, California, USA (Dec 1989)
28. Lehoczky, J.P.: Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings of the 11th Real-Time Systems Symposium (RTSS). pp. 201–209 (Dec 1990)
29. Leung, J.Y.T., Merrill, M.: A note on the preemptive scheduling of periodic, real-time tasks. Information Processing Letters **11**, 115–118 (1980)
30. Leung, J.Y.T., Whitehead, J.: On the complexity of fixed-priority scheduling of periodic, real-time tasks. Performance Evaluation **2**, 237–250 (1982)
31. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM **20**(1), 46–61 (1973)
32. Mok, A.: Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment. Ph.D. thesis, Laboratory for Computer Science, Massachusetts Institute of Technology (1983), available as Technical Report No. MIT/LCS/TR-297
33. Mok, A.K., Chen, D.: A multiframe model for real-time tasks. In: Proceedings of the 17th Real-Time Systems Symposium. IEEE Computer Society Press, Washington, DC (1996)
34. Moyo, N.T., Nicollet, E., Lafaye, F., Moy, C.: On schedulability analysis of non-cyclic generalized multiframe tasks. In: Proceedings of the EuroMicro Conference on Real-Time Systems. IEEE Computer Society Press, Brussels (July 2010)
35. Robson, J.M.: The complexity of Go. In: Mason, R.E.A. (ed.) Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983. pp. 413–417. North-Holland/IFIP (1983)
36. Saffidine, A., Teytaud, O., Yen, S.J.: Go complexities. In: Advances in Computer Games. pp. 76–88. Springer International Publishing, Cham (2015)
37. Stigge, M.: Real-Time Workload Models: Expressiveness vs. Analysis Efficiency. Ph.D. thesis, Uppsala University, Department of Information Technology (2014)
38. Stigge, M., Ekberg, P., Guan, N., Yi, W.: The digraph real-time task model. In: Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS). pp. 71–80. IEEE Computer Society Press, Chicago (2011)
39. Stigge, M., Ekberg, P., Guan, N., Yi, W.: On the tractability of digraph-based task models. In: Proceedings of the EuroMicro Conference on Real-Time Systems. IEEE Computer Society Press, Porto, PT. (July 2011)
40. Stigge, M., Yi, W.: Hardness results for static priority real-time scheduling. In: 2012 24th Euromicro Conference on Real-Time Systems. pp. 189–198 (2012). https://doi.org/10.1109/ECRTS.2012.13
41. Stigge, M., Yi, W.: Graph-based models for real-time workload: a survey. Real-Time Systems pp. 602—-636 (2015). https://doi.org/10.1007/s11241-015-9234-z
42. Tindell, K.W., Burns, A., Wellings, A.J.: An extendible approach for analysing fixed priority hard real-time tasks. Real-Time Systems: The International Journal of Time-Critical Computing **6**, 133–151 (1994)
43. Wellings, A., Richardson, M., Burns, A., Audsley, N., Tindell, K.: Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal **8**, 284–292 (1993)
44. Woeginger, G.J.: When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? INFORMS Journal on Computing **12**(1), 57–74 (2000). https://doi.org/10.1287/ijoc.12.1.57.11901

45. Woeginger, G.J.: The trouble with the second quantifier. 4OR: A Quarterly Journal of Operations Research **19**(2), 157–181 (2021). https://doi.org/https://doi.org/10.1007/s10288-021-00477-y
46. Zheng, W., Zhu, Q., Natale, M.D., Vincentelli, A.S.: Definition of task allocation and priority assignment in hard real-time distributed systems. In: 28th IEEE International Real-Time Systems Symposium (RTSS 2007). pp. 161–170 (2007). https://doi.org/10.1109/RTSS.2007.40