Rethinking efficiency in real-time schedulability analysis*

Sanjoy Baruah (Speaker) † Pontus Ekberg ‡

Whereas the algorithms community has traditionally associated computational intractability with NP-hardness, real-time systems has a more relaxed view. In particular, real-time schedulability-analysis algorithms that are designed for use prior to system runtime are considered acceptably efficient if they have pseudo-polynomial running time – examples of widely-used such pseudo-polynomial time algorithms include Response-Time Analysis [3] for fixed-priority scheduling and Processor-Demand Analysis [4, 1] for EDF scheduling of recurrent task systems upon preemptive uniprocessors. More recently, the presence of excellent ILP solvers has led to approaches based on transforming a schedulabilty analysis problem to a (reasonably small, i.e., polynomial sized) ILP to often also considered acceptably efficient.

Recall that showing a problem to be NP-hard implies it is unlikely to have a polynomial-time solution, and showing it to be strongly NP-hard implies it is unlikely to have a pseudo-polynomial solution. In a similar vein, showing a problem to be hard for the complexity class $\mathsf{NP}^{\mathsf{NP}}$ (also denoted as Σ_2^{P}) offers strong evidence that it cannot be efficiently solved even with a highly optimized ILP solver[1].

Summarizing the discussion above, notions of efficiency in real-time schedulability analysis have converged upon three beliefs. *(i).* Algorithms with pseudo-polynomial running times are efficient. *(ii).* "Polynomial-time + ILP-solver" algorithms – algorithms with polynomial running time that are additionally allowed to make calls to an ILP solver – are also coming to be considered efficient. *iii).* Showing a problem to be hard for Σ_2^{P} shows it to be truly intractable.

However, the following recent observation brings the third of these beliefs into question, by showing that there are problems that are hard for Σ_2^{P} (and indeed, even more general complexity classes) that have pseudo-polynomial time solutions.

Observation 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 complexity class of all decision problems that are solvable using exponential-time algorithms.)* □

A finer-grained view of pseudo-polynomial time. Pseudo-polynomial–time algorithms are considered efficient for real-time scheduling problems because their numerical

*Proofs of the results presented here appear in [2] (and the reference cited there).

†baruah@wustl.edu. Washington University in St. Louis, USA

‡pontus.ekberg@it.uu.se. Uppsala University, Sweden

[1] Woeginger [5] explains the implications in this manner: "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 concludes that "Σ_2^{P}-complete problems are much, much, much, much, much harder than any problem [...] that can be attacked via ILP solvers."

parameters typically have a direct physical interpretation, most commonly as measures of time, that are unlikely to be too large in practice. When the largest numerical parameter N greatly exceeds the total input size n, the running time of a pseudo-polynomial algorithm is dominated by its dependence on N. Despite this, little attention has been paid in the real-time scheduling literature to how pseudo-polynomial algorithms scale with respect to N, treating all such algorithms as essentially equivalent. This motivates a more fine-grained notion that makes the dependence on N explicit:

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.*

Thus 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. The result below states that any difference in polynomial degree between f and g differentiates pseudo-f from pseudo-g:

Theorem 2. *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.* □

Scale invariance. In addition to distinguishing pseudo-polynomial–time algorithms by their polynomial dependence on numerical input values, we also wish to distinguish them by whether their running time is invariant under uniform scaling of those values. Ideally, a pseudo-polynomial algorithm should become slower only when the relationships among numerical parameters become more complex, not when all values are scaled by a common factor. In scheduling problems where numerical parameters often represent time, this corresponds to insensitivity to the choice of time unit: an algorithm should have the same running time whether times are expressed in milliseconds, microseconds, or any other unit, provided the instance structure is unchanged and all values remain integral. The following definition formalizes this notion of scale invariance for pseudo-polynomial–time algorithms.

Definition 3 (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*

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 3. 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.* □

References

[1] S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.

[2] Sanjoy Baruah and Pontus Ekberg. A closer look at pseudo-polynomial time and its use in real-time scheduling theory. In Susanne Graf, Paul Pettersson, and Bernhard Steffen, editors, *Real Time and Such: Essays Dedicated to Wang Yi to Celebrate His Scientific Career*, pages 120–134. Springer Nature Switzerland, Cham, 2025.

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

[4] Joseph Y.-T Leung and M. Merrill. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11:115–118, 1980.

[5] Gerhard J Woeginger. The trouble with the second quantifier. *4OR: A Quarterly Journal of Operations Research*, 19(2):157–181, 2021.