Work in Progress: The ILP-Tractability of Schedulability Analysis Problems

Sanjoy Baruah
Washington University in St. Louis
baruah@wustl.edu

***Abstract*—Algorithms used for the pre-runtime analysis of safety-critical systems have traditionally been required to have running times no worse than pseudo-polynomial in the size of their inputs. Some recent work, however, has been motivated by a vast improvement in the performance of Integer Linear Programming (ILP) solvers and a concurrent widespread and inexpensive availability of increasing computing capabilities, to consider the use of ILP solvers as acceptably efficient for the purposes of such analysis. In this paper, the concept of ILP-tractability is proposed as a formal characterization of the class of scheduling problems that can be solved efficiently under this newer interpretation of efficiency. Techniques are presented for showing a problem to be ILP-tractable, as well as for showing a problem to be ILP-*in*tractable — i.e., it cannot be solved efficiently using ILP solvers.**

***Index Terms*—Schedulability-analysis; Integer Linear Programs (ILPs); ILP-tractability; Computational Complexity**

Safety-critical systems are required to have the correctness of their run-time behavior verified prior to deployment: such verification is usually done by analysis of models of the run-time behavior. It is desirable that we be able to design *efficient* algorithms that enable us to derive interesting properties of the model, if it to be of much use in system design and analysis. As computing capabilities have continued to grow at an exponential rate (as codified by Moore's Law), we have seen a continued increase in computing capabilities and as a consequence, our understanding of what forms of analysis may be considered "efficient" has continued to evolve. In this paper, we consider a particular aspect of run-time correctness – the ability to meet deadlines – that has long been a prime focus of the real-time scheduling community. The pre-run-time analysis of systems to check whether they will always meet their deadlines during run-time is called *schedulability analysis*. In the early years of the discipline of real-time computing, schedulability analysis algorithms were required to have worst-case running times that are low-degree polynomials of input size in order to be considered efficient. Examples of efficient algorithms of this first generation include the utilization-based schedulability tests [1], [2] for Earliest-Deadline First (EDF) and Rate-Monotonic scheduling of collections of independent implicit-deadline sporadic tasks ("Liu and Layland tasks") upon preemptive uniprocessors.

By the mid- to late-1980s, however, computing capabilities had increased enough that schedulability analysis algorithms with *pseudo-polynomial* running times were considered efficient. Early examples of efficient algorithms of this kind include the preemptive uniprocessor EDF schedulability test for bounded-utilization 3-parameter sporadic task systems [3], and Response-Time Analysis for fixed-priority task systems [4]. Ever since, there appears to have been a sort of consensus that pseudo-polynomial running time equates to efficiency, and there has been a continued quest to identify the most general models for which schedulability analysis can be done in pseudo-polynomial time (please see [5] for an excellent survey on this topic).

More recently, the continued increase in computing capabilities, combined with the increasingly complex nature of many schedulability-analysis problems that one encounters whilst seeking to verify the correctness of modern safety-critical cyber-physical systems, has motivated some real-time scheduling theory researchers to take the first tentative steps past the pseudo-polynomial time barrier. Many such investigations seek to transform a schedulability-analysis problem to some other form such as an integer linear program (ILP) or some satisfiability modulo theories (SMT) [6], which can then be solved by a solver of the appropriate kind (i.e., an ILP solver or an SMT-solver, respectively). In this work, we focus on schedulability-analysis research efforts that seek to go beyond the pseudo-polynomial time barrier via the use of ILP solvers.

ILP solvers. Determining whether an integer linear program (ILP) has a feasible solution was one of the earliest problems shown to be NP-complete [7] (recall that an NP-*complete* problem is both NP-hard and in the class NP). Indeed, it is known to be NP-complete *in the strong sense*; assuming P $\neq$ NP, this implies that ILP solvers with pseudo-polynomial running time cannot be developed. Despite this inherent intractability of ILP, however, the optimization community has recently been devoting immense effort to develop very efficient implementations of ILP solvers, and highly-optimized libraries with such efficient implementations are widely available today in both open-source and commercial offerings. Modern ILP solvers, particularly when running upon powerful computing clusters, are commonly capable of solving ILPs with tens of thousands of variables and constraints. The availability of such solvers is what motivates the research question is explored here: *which scheduling problems can be efficiently solved if one were to have access to a very good ILP-solver*?

ILP-Tractability. Informally speaking, a problem is ILP-tractable if it can be solved by a polynomial-time algorithm that has access to an efficient ILP-solver. Of course, since all ILP-solvers have exponential worst-case running times it

follows that making even a single call to the ILP solver could result in the overall running time of the algorithm no longer being polynomial in the worst case. However we propose to "factor out" the time taken by the ILP solver from worst-case consideration, by leveraging off the fact that modern efficient implementations of ILP-solvers should be able to solve the vast majority of ILPs in far better than exponential time. More precisely, *we define a problem as being ILP-tractable if it can be solved by an algorithm that has worst-case running time polynomial in the size of its input, and may additionally make polynomially many calls to an ILP solver.*

Showing ILP-Tractability. To show that a problem is ILP-tractable one need simply present (and prove correct, of course) an algorithm that bears witness to this fact. We illustrate via a pair of examples:

§1. Consider the partitioned preemptive EDF scheduling of a given collection of implicit-deadline sporadic tasks upon an identical or uniform multiprocessor platform [8]. This problem is very closely related to the well-known bin-packing problem [9], and hence NP-hard in the strong sense. This hardness result implies that we are unlikely to find a polynomial or pseudo-polynomial time exact algorithm for partitioned EDF scheduling of implicit-deadline sporadic tasks. However, this problem is easily seen to be ILP-tractable – it can be represented as an integer linear program in polynomial time, by simply defining 0-1 integer variables $x_{i,j}$ to denote whether the i'th task is to be assigned to the j'th processor, and writing constraints to require that (i) each task gets assigned to some processor; and (ii) the cumulative utilization assigned to each processor does not exceed the processor's capacity. Constructing such an ILP can clearly be accomplished in polynomial time, which can be solved by a single call to an ILP solver.

§**2.** As another illustration of the process of showing ILP-tractability, consider the uniprocessor preemptive EDF scheduling of systems of *asynchronous periodic tasks* [10]. It has been shown [10] that such systems are not EDF-schedulable if and only if there is a time interval over which the cumulative processor demand exceeds the processor capacity, and that determining whether such a time interval exists is co-NP complete in the strong sense [11], [12] (and hence unlikely to be solved by a polynomial or pseudo-polynomial time algorithm). However it is quite easy to write an ILP that determines whether such a time interval exists. Hence an algorithm that simply constructs such an ILP, calls the ILP solver, and returns "unschedulable" if and only if the ILP solver reports the existence of such a time interval, bears witness to the fact that uniprocessor preemptive EDF schedulability of asynchronous periodic task systems is ILP-tractable.

Showing ILP-*In*tractability. We saw above that problems can be shown to be IPL-tractable in a pretty direct manner: we simply produce the polynomial-time algorithm, that may additionally call an ILP solver polynomially many times, that solves the problem. But showing that a problem is ILP-*intractable* — i.e., that it is not ILP-tractable — is not quite as straightforward: how does one show that one *cannot* solve a problem in polynomial time with at most polynomially many calls to an ILP solver? We propose to do so by building off some results from computational complexity theory.

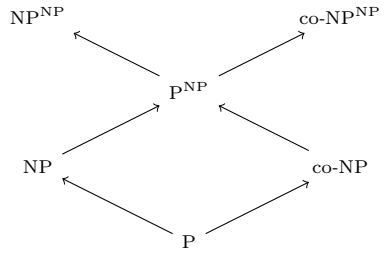

Fig. 1. Relationship between complexity classes

We came across the complexity classes NP and co-NP in the discussion on ILP-tractability above: partitioned preemptive EDF scheduling of implicit-deadline sporadic task systems is NP complete, while uniprocessor preemptive EDF scheduling of systems of asynchronous periodic tasks is co-NP complete. The *polynomial-time hierarchy* [13] extends computational complexity theory beyond P, NP, and co-NP, by defining an abstract model of computers (strictly speaking, Turing Machines) equipped with an *oracle*: a "black box" that is able to solve a specific problem in a single step, in constant time. (Thus the ILP solver in our discussion of ILP-tractability above can be thought of as an oracle for solving ILPs.) The complexity class P^{NP} denotes the class of all problems that can be solved in polynomial time by a computer that is equipped with an oracle that solves some NP-complete problem. In a similar vein, the complexity class $\mathrm{NP}^{\mathrm{NP}}$ denotes the class of all problems that can be *verified* in polynomial time by a computer that is equipped with an oracle that solves some NP-complete problem. And just as co-NP denotes the class of problems whose complement problems are in NP, co-$\mathrm{NP}^{\mathrm{NP}}$ denotes the class of problems whose complement problems are in $\mathrm{NP}^{\mathrm{NP}}$.

The relationship amongst the six complexity classes we have discussed above (P, NP, co-NP, P^{NP}, $\mathrm{NP}^{\mathrm{NP}}$, and co-$\mathrm{NP}^{\mathrm{NP}}$) is depicted in Figure 1; the arrows represent the SUBSET-OF relationship $\subseteq$. It is widely assumed (although unproven) that all of these SUBSET-OF relationships are strict.[1]

Based on these concepts from complexity theory, a technique for showing that a problem is ILP-intractable immediately suggests itself. As stated above, solving ILPs has been shown [7] to be an NP-complete problem. Thus any problem that can be solved by making polynomially many calls to an ILP solver is in the complexity class P^{NP}. Under the widely-

[1] Indeed, the famous "P$\stackrel{?}{=}$NP" question asks whether the arrow in Figure 1 from the set P to the set NP represents set equality (P=NP) or strict inclusion (P$\subsetneq$NP): the unproved near-consensus answer amongst experts is strict inclusion.

held assumption that each of the SUBSET-OF relationships in Figure 1 is strict, any problem that is shown to be hard for the complexity classes $\mathrm{NP}^{\mathrm{NP}}$ and co-$\mathrm{NP}^{\mathrm{NP}}$ cannot then belong to the class P^{NP}. Hence, *we can show that a problem is ILP-intractable by showing that it is hard for either of the complexity classes* $\mathrm{NP}^{\mathrm{NP}}$ *or co-*$\mathrm{NP}^{\mathrm{NP}}$.

And how does one show this? Recall that a new problem can be shown to be NP-hard (co-NP-hard, respectively) by demonstrating a polynomial-time reduction from some known NP-hard (co-NP-hard, resp.) problem to the new problem [14], [7]. In much the same manner [13], [15], one can show that a new problem is $\mathrm{NP}^{\mathrm{NP}}$-hard or co-$\mathrm{NP}^{\mathrm{NP}}$-hard by defining a polynomial-time reduction to the new problem from some known $\mathrm{NP}^{\mathrm{NP}}$-hard or co-$\mathrm{NP}^{\mathrm{NP}}$-hard problem. A canonical $\mathrm{NP}^{\mathrm{NP}}$-complete problem that may be used for this purpose is the $\exists\forall$**3**SAT Problem (Definition 1 below), while a canonical co-$\mathrm{NP}^{\mathrm{NP}}$-complete problem is the $\forall\exists$**3**SAT Problem (Definition 2 below):

Definition 1 (The $\exists\forall$**3**SAT Problem)**.**
INSTANCE. *A Boolean formula* $\phi(\vec{X},\vec{Y})$ *in 3CNF (Conjunctive Normal Form – i.e., as the "and" of clauses each comprising exactly 3 literals)*
QUESTION. *Is is true that* $(\exists\vec{X})(\forall\vec{Y})\phi(\vec{X},\vec{Y})$*?* □

Definition 2 (The $\forall\exists$**3**SAT Problem)**.**
INSTANCE. *A Boolean formula* $\phi(\vec{X},\vec{Y})$ *in 3CNF*
QUESTION. *Is is true that* $(\forall\vec{X})(\exists\vec{Y})\phi(\vec{X},\vec{Y})$*?* □

RESULTS OBTAINED THUS FAR

This concept of ILP-intractability arose out of efforts to design efficient algorithms for determining the minimum makespan (i.e., scheduling duration) for conditional DAGs [16], [17] upon identical multiprocessor platforms. This problem was already known to be NP-hard in the strong sense under most interesting restrictions including global scheduling (when individual jobs are permitted to migrate amongst the processors) [18], and for "typed" systems (where each job is pre-assigned to an individual processor or a specified subset of the processors) [19], both when preemptions are permitted and when they're not. These prior results imply that there cannot be polynomial or pseudo-polynomial time exact algorithms for makespan minimization, and hence we attempted to solve it by representing as an ILP and then using an ILP-solver. We were not successful in doing so, which led us to investigate why and motivated the definition of the ILP-intractability framework introduced here. Based on this conceptual framework, we defined a polynomial-time reduction from the $\forall\exists$**3**SAT Problem (Definition 2 above) to the makespan minimization problem upon conditional DAGs in which each job is pre-assigned to a specific processor, thereby showing that this makespan minimization problem is co-$\mathrm{NP}^{\mathrm{NP}}$-hard and thus ILP-intractable. It has subsequently been reported to us[2] that our reduction may be adapted to show that the makespan minimization of conditional DAGs scheduled under global preemptive or non-preemptive scheduling is also co-$\mathrm{NP}^{\mathrm{NP}}$-hard (and hence also ILP-intractable).

[2]Alberto Marchetti-Spaccamela, private communication.

ONGOING AND FUTURE WORK

Current and planned research on ILP-tractability can be considered to fall into one of four broad categories.

§**1: Classification.** We are working on categorizing other open scheduling problems as being ILP-tractable or not. For problems that are indeed ILP-tractable, we plan to identify ILP representations that perform well in practice, by, e.g., requiring fewer calls to ILP-solvers, and / or needing smaller ILPs (i.e., with fewer integer variables and fewer constraints) to be solved by these solvers.

§**2: Identifying the characteristics of ILP-intractability.** We would like to demarcate the precise boundary between ILP-tractability and ILP-intractability: what are the specific features that, when present in a problem, render it ILP-intractable?[3] In this direction, we have been able to show that conditional DAGs in which the number of conditional constructs is bounded from above by a constant is ILP-tractable (and can in fact be represented as a *single* ILP). Therefore the presence of an unbounded number of conditional constructs is what renders analysis of a conditional DAG instance ILP-intractable.

§**3: Generalization to Pseudo-polynomial time.** As stated earlier, our notion of efficiency evolved over time from polynomial time to pseudo-polynomial time. We are anticipating further such evolution and looking ahead to consider generalizations to our notion of ILP-tractability, to when it may be reasonable to consider as tractable those algorithms that take pseudo-polynomial time and make pseudo-polynomially many calls to ILP solvers. To our knowledge, this is unexplored territory in theoretical computer science: we have not been able to track down any prior work that considers such complexity classes of algorithms.

§**4: Approximation.** We are exploring relationships between ILP-intractability and polynomial-time approximability: does showing a problem to be (e.g.,) $\mathrm{NP}^{\mathrm{NP}}$-hard mean anything about whether it is polynomial time or pseudo-polynomial time approximable? (E.g., whether the existence of a PTAS or FPTAS for the problem is ruled out).

[3]Aa an analogy, consider again the uniprocessor preemptive EDF schedulability of asynchronous periodic task systems. We saw above that this is co-NP complete in the strong sense [11], [12] and hence not solvable in pseudo-polynomial time. However, it has been shown [3] that for *synchronous* systems – those for which the initial release-times of all the tasks are equal – for which system utilization is bounded from above by a constant < 1, preemptive EDF schedulability can be determined in pseudo-polynomial time. It is further known [20], [21] that preemptive EDF schedulability analysis of synchronous systems without the bounded-utilization property remains NP-complete in the strong sense, thereby allowing us to conclude that both synchronicity and bounded utilization are needed to ensure pseudo-polynomial time tractability.

REFERENCES

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

[2] Enrico Bini, Giorgio Buttazzo, and Giuseppe Buttazzo. Rate monotonic scheduling: The hyperbolic bound. *IEEE Transactions on Computers*, 52(7):933–942, 2003.

[3] 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.

[4] Mark H. Klein, Thomas Ralya, Bill Pollak, Ray Obenza, and Michael González Harbour. *A Practitioner's Handbook for Real-time Analysis*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[5] Martin Stigge. *Real-Time Workload Models: Expressiveness vs. Analysis Efficiency*. PhD thesis, Uppsala University, 2014.

[6] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[7] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[8] J. M. Lopez, J. L. Diaz, and D. F. Garcia. Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, 28(1):39–68, 2004.

[9] D. S. Johnson. *Near-optimal Bin Packing Algorithms*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1973.

[10] 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.

[11] S. Baruah, R. Howell, and L. Rosier. On preemptive scheduling of periodic, real-time tasks on one processor. In *Fifteenth International Symposium on the Mathematical Foundations of Computer Science*, volume 2, pages 173–179, Bansk'a Bystrica, Czechoslovakia, 1990. Springer-Verlar.

[12] 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.

[13] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.

[14] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[15] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3:23–33, 1976.

[16] Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *Proceedings of the 2014 26th Euromicro Conference on Real-Time Systems*, ECRTS '15, pages 222–231, Lund (Sweden), 2015. IEEE Computer Society Press.

[17] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo. Response-time analysis of conditional DAG tasks in multiprocessor systems. In *Proceedings of the 2014 26th Euromicro Conference on Real-Time Systems*, ECRTS '15, pages 222–231, Lund (Sweden), 2015. IEEE Computer Society Press.

[18] J. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384 – 393, 1975.

[19] Klaus Jansen. Analysis of scheduling problems with typed task systems. *Discrete Applied Mathematics*, 52(3):223 – 232, 1994.

[20] Pontus Ekberg. *Models and Complexity Results in Real-Time Scheduling Theory*. PhD thesis, Uppsala University, 2015.

[21] P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-complete. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 281–286, 2015.