Fixed-Parameter Analysis of Preemptive Uniprocessor Scheduling Problems

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

Pontus Ekberg
Uppsala University
pontus.ekberg@it.uu.se

Abhishek Singh
Washington University in St. Louis
abhishek.s@wustl.edu

***Abstract*—The algorithmic technique of *fixed-parameter analysis* of computationally intractable problems seeks to obtain a deeper understanding of the underlying causes of the intractability, with a view to identifying conditions under which the problem becomes tractable. We apply fixed-parameter analysis to the fixed-priority and EDF scheduling of recurrent (periodic and sporadic) task systems upon preemptive uniprocessor platforms.**

I. Introduction

Many (arguably, most) important and interesting real-time schedulability analysis problems are computationally intractable—NP-hard or coNP-hard—even upon preemptive uniprocessor platforms (see, e.g., [1] for an overview). Somewhat paradoxically, it has widely been observed that many of these problems seem to be solved quite efficiently for many (though not all) of the kinds of systems that are commonly encountered in practice. The objective of the research reported in this paper is to investigate this paradox, primarily through the algorithmic analysis technique of *fixed-parameter analysis* [2]–[4] (see [5] for a short review) which has recently become the focus of a good deal of attention in the algorithms community. Fixed-parameter analysis of a computationally intractable problem seeks to identify the problem parameters that are responsible for the combinatorial explosion arising in algorithms for solving the problem. The motivation for doing so, as articulated in [6], is that "it is hoped then that these parameters (in the concrete application behind the problem under consideration) might take only relatively small values, so that the exponential growth becomes affordable".

In exploring the applicability of fixed-parameter algorithms for some particular application domain, one of the most important issues to be resolved is *identifying the parameters* that are subject to fixed-parameter analysis, i.e., that are 'fixed' in the analysis. Many factors play a role in making this decision. First and foremost for our purposes, the choice of parameters should be meaningful from the perspective of the problem application domain, in the sense that it is reasonable to consider problem instances in which these parameters take on relatively small values. These parameters should also facilitate the design of efficient algorithms, and hopefully be useful in providing insights into the inherent complexity of the underlying problem.[1] Additionally, one would like to restrict fewer parameters rather than more: the fewer the number of parameters that are required to take on relatively small values in order for a given fixed-parameter algorithm to be considered tractable, the more general the applicability of this algorithm. We will see that while the choice of parameter may be somewhat obvious in some cases (e.g., in multiprocessor scheduling it seems quite natural to consider the number of processors to be a parameter), in some cases it is not quite as straightforward (an example: we will see, in a later section, that considering the ratio of the largest period of any task in a task system to the smallest period as the parameter provides a satisfactory explanation for the excellent performance that has been observed in practice of the widely-used response-time analysis (RTA) [8], [9] and processor-demand analysis (PDA) [10] schedulability tests).

Significance. Fixed-parameter analysis has proved helpful in making advances in other domains in which some very basic problems are computationally intractable. Recently, researchers have begun exploring the applicability of fixed-parameter analysis for "traditional" scheduling problems of the kind studied in the Operations Research community — e.g., Mnich and Wiese [11], [12] have applied the lens of fixed-parameter analysis to the traditional scheduling problems of non-preemptive makespan minimization on an arbitrary number of identical machines or a bounded number of unrelated machines; scheduling with rejection; and a profit maximization variant of the Generalized Scheduling Problem defined by Bansal and Pruhs [13]. We believe that it is high time that the real-time scheduling theory community also considers this analysis technique, with the dual objectives of better understanding some of our intractable basic scheduling problems, and developing design guidelines that, upon identifying inherent sources of intractability, specify how these should be avoided in the systems we build.

This work. The larger algorithms community has begun looking upon fixed-parameter analysis as a promising means of dealing with inherently intractable problems; it seems only

Funded in part by the US National Science Foundation under Grants CPS-1932530 and CNS-2141256, and by the Swedish Research Council under Grant 2018-04446. Authors listed in alphabetical order; all contributed equally.

[1]An article [7] reviewing some early works on fixed-parameter analysis had observed that "the choice of parameters can make a problem interesting or boring, tractable or non-tractable, and even tractable but for mundane reasons"; we seek to identify parameters that render our scheduling problems tractable, for reasons that are not mundane.

proper that the real-time scheduling theory community does likewise. Preemptive uniprocessor schedulability analysis of periodic and sporadic task systems are the building blocks of much of real-time scheduling theory (see, e.g. [8], [9], [14]–[17]—this is only a short representative sample). But these seminal papers were all written before fixed-parameter analysis was even proposed [2], [3] as a rigorous analysis method in the mid-1990s. In this paper, we report on our findings and experiences in applying fixed-parameter analysis to such foundational schedulability analysis problems. We claim the following specific contributions.

1) The framework of fixed-parameter analysis has recently been considerably enriched (see [5], published in 2021, for a short review of recent developments). However, to our knowledge, there is no prior research explicitly applying this rich recently developed framework to real-time scheduling problems; our first contribution is to have initiated this. We hope that the experiences we report here will encourage other members of the real-time research community to also dive in, and that our brief introduction to the topic (Section III) will lower the barrier to entry.
2) Via the application of fixed-parameter analysis, we have obtained some explanations for the excellent perceived performance in practice of widely-used schedulability-analysis algorithms (such as RTA [8], [9] and PDA [10]) under a variety of circumstances. For other circumstances, we have identified other schedulability analysis algorithms that are not quite as widely known or used (e.g., the HET algorithm [18], [19], or ones based on integer linear programming), which may exhibit superior properties with increasing problem size.
3) Fixed-parameter analysis includes its own notions of *lower bounds*—characterizations of problems for which fixed-parameter analysis may not yield better algorithms. We provide a brief review of lower bound theory for fixed-parameter analysis (Section VI) and apply this theory to establish lower bounds for several well-known schedulability analysis problems.

Organization. The remainder of this paper is organized in the following manner. In Section II we briefly present the real-time task model that we will be using in the remainder of this paper. In Section III we provide a very brief introduction to some of the more important concepts of fixed-parameter analysis, from a real-time scheduling perspective. In Section IV we apply fixed-parameter analysis to uniprocessor fixed-priority schedulability analysis; in Section V, we do likewise for uniprocessor EDF schedulability analysis. In Section VI we provide a brief introduction to the theory of lower bounds for fixed-parameter analysis and establish lower bounds for a variety of real-time scheduling problems. We conclude in Section VII by placing this work within a larger context of research on real-time scheduling theory.

II. Workload Model, and Some Background

We consider the problem of scheduling a real-time system Γ that is modeled as a collection of independent recurrent tasks, upon a preemptive uniprocessor. Each task τ_i is characterized by a *worst-case execution time* (WCET) parameter C_i, denoting the maximum duration that a job of τ_i would take to complete execution on a processor; a *relative deadline* D_i; and a *period* T_i. Task τ_i may release a potentially unbounded sequence of jobs during any given execution of the task system; each job released by τ_i is required to complete execution within a duration D_i of its release time. We will consider systems of both *sporadic* and *periodic* tasks: for sporadic tasks, the period T_i represents the minimum duration between the release of successive jobs while for periodic tasks, T_i represents the exact duration between the release of successive jobs. A periodic task τ_i is additionally characterized by an *initial release time* or *offset* O_i, and hence releases jobs at instants $O_i + k \times T_i$ for all $k \in \mathbb{N}$.

Synchronous and asynchronous periodic task systems. Periodic task systems may be further characterized as being *synchronous* if they satisfy the additional constraint that their initial release times are all equal (in which case they are typically assumed to be equal to zero and omitted from the specifications). Periodic task systems that are not synchronous, i.e., in which all tasks do not have equal initial release times, are called *asynchronous* periodic task systems. (Synchronous periodic and sporadic task systems have been observed [20] to be remarkably similar from a uniprocessor schedulability analysis perspective; we will therefore frequently use the phrase 'synchronous task systems' to include both synchronous periodic and sporadic task systems.)

EDF and FP scheduling. We will consider the preemptive scheduling of recurrent task systems on a shared processor where:

1) priorities are dynamically assigned to jobs during runtime such that jobs with earlier deadlines receive higher priorities; or
2) priorities are statically assigned to tasks before runtime.

The former case is referred to as Earliest Deadline First (EDF) scheduling, and the latter case is referred to as Fixed Priority (FP) scheduling.

Implicit-, constrained- and arbitrary-deadline systems. Periodic or sporadic task systems are said to be *implicit-deadline* if $D_i = T_i$ for all tasks (the D_i parameter is typically dropped from the specification of tasks in such systems), and *constrained-deadline* if $D_i \leq T_i$ for all tasks. If no such restrictions are placed on the D_i and T_i parameters (i.e., we may have $D_i > T_i$), then the task system is said to be *arbitrary-deadline*.

Bounded-utilization systems. The ratio of the WCET parameter of a recurrent task to its period is termed its *utilization*. The *(system) utilization* $U(\Gamma)$ of task system Γ is defined to be the sum of the utilizations of all tasks in Γ. A *bounded-utilization* system Γ satisfies the constraint that its utilization is guaranteed to be no larger than a specified constant that is strictly smaller than 1.

III. Fixed-Parameter Analysis: A Brief Introduction

In this section we provide a brief introduction to some of the main ideas underlying fixed-parameter analysis. We do not intend to be comprehensive (or completely rigorous and formal), and as far as possible draw upon examples from real-time scheduling problems with which we expect the reader is already very familiar.

In the classical study of algorithms and complexity, the running time of an algorithm is stated as a function of the size of its input. In the parameterized study of algorithms and complexity, the running time of an algorithm is stated as a function of both the size of the input and a *parameter* which is a numerical value that depends on the input and is obtained via a parameterization (defined next).

Definition 1 (parameterization)**.** A *parameterization* for a problem is a computable function that accepts as input any problem instance and returns an integer:

$$\kappa : \{\text{problem instances}\} \to \mathbb{N}$$

Consider, as an example problem, the preemptive uniprocessor fixed-priority (FP) schedulability analysis of constrained-deadline synchronous task systems – this is the schedulability analysis problem that is solved by Response-Time Analysis (RTA) [8], [9]. For this problem, any constrained-deadline synchronous task system Γ constitutes a problem instance. We may define many parameterizations for this problem. For instance, a particular parameterization of Γ may return *the number of tasks* in Γ; another may return *the largest-valued parameter* in the specification of Γ; a third, *the ratio of the largest period to the smallest period* of tasks in Γ, and so on (these and a couple of other possible parameterizations are enumerated in Table I. [2])

While Definition 1 is very liberal allowing any computable function to be considered a parameterization, some parameterizations are more meaningful than others from a practical perspective: for example, while it is perfectly correct to define a parameterization that returns zero (one, respectively) if the number of tasks in the instance is odd (even, resp.), it is difficult to see such a parameterization playing a meaningful role in schedulability analysis.

Definition 2 (xp-algorithm)**.** For a given problem P and a parameterization κ for P, an xp-algorithm for (P, κ) solves any instance x in running time $O(f(\kappa(x)) \times |x|^{g(\kappa(x))})$, where $f(\cdot)$ and $g(\cdot)$ are some computable functions on integers.

Note that for any particular fixed value $k = \kappa(x)$, both $f(k)$ and $g(k)$ are also fixed, and so an xp-algorithm runs in time polynomial in the size of the input $|x|$ (the size is normally the number of bits needed to represent the instance), for each fixed value k of the parameter. The *xp* stands for "slice-wise polynomial" and refers to this fact that every *slice* of the problem that is defined by a particular value $k = \kappa(x)$ is solved in polynomial time.

[2]Some subtle issues arise in the use of parameterization κ_3—this will be discussed in the Appendix (but may be ignored for now).

Example 1. Consider the multiprocessor problem of partitioned EDF scheduling of a given collection Γ of implicit-deadline tasks upon m processors (a problem instance here is the ordered pair (Γ, m)). It is widely known that this problem is equivalent to the BIN-PACKING problem [21, Problem SR1] and thus NP-hard. There is no xp-algorithm for this problem under the parameterization that returns the number of processors (since BIN-PACKING is NP-hard even upon two processors). However since exhaustively considering all possible partitionings of the tasks amongst the m processors will require us to check $m^{|\Gamma|}$ possibilities, each in polynomial time, such an 'exhaustively-enumerate and test' approach is an xp-algorithm under a parameterization that returns $|\Gamma|$, the number of tasks in Γ. $\square$

Example 1 indicates that the choice of parameterizations is crucial from two perspectives. First, it reveals that a problem may have an xp-algorithm under some parameterizations but not others. Second, it offers strong evidence that the mere presence of an xp-algorithm is not in itself sufficient to ensure that a practically efficient algorithm exists: the 'exhaustively-enumerate and test' xp-algorithm of Example 1 would not be considered practical even for relatively small values of the parameter.

Although the existence of an xp-algorithm is, from the perspective of fixed-parameter analysis, better than there not being such an algorithm, the presence of the parameter value as an exponent to the problem instance size ($|x|$) means that xp-algorithms are often practically significant only for very small values of the parameter. The parameter value may not appear as an exponent to the problem instance size in fpt-algorithms, defined next (*fpt* stands for "fixed-parameter tractable"):

Definition 3 (fpt-algorithm)**.** For a given problem P and a parameterization κ for P, an fpt-algorithm for (P, κ) solves any instance x in running time $O(f(\kappa(x)) \times |x|^c)$, where $f(\cdot)$ is any computable function and c is a constant number.

Notice that the exponent to the problem instance size is required to be a constant that does *not* depend on the parameter; rather, the impact of the parameter is contained in the multiplicative term $f(\kappa(x))$. It is primarily for this reason that *fpt-algorithms are considered particularly efficient* (while xp-algorithms are not necessarily so) in fixed-parameter analysis. Since the running time of an fpt-algorithm is of the form $f(\kappa(x)) \times |x|^c$, there are usually two clear directions for optimizing its performance:

- Reduce f, thereby reducing the dependence of the running time upon the parameter.
- Reduce c, thereby reducing the dependence of the running time upon the size of the input.

Several of the lower bounds that we will present in Section VI establish that although xp-algorithms may exist for certain problems, fpt-algorithms for these problems are unlikely to exist.

TABLE I
SOME POSSIBLE PARAMETERIZATIONS FOR PERIODIC TASK SYSTEMS.

Label	Meaning	Formal Meaning
κ_1	number of tasks, a.k.a, *cardinality* of Γ	$\|\Gamma\|$
κ_2	number of (offset, deadline, period) triples, a.k.a., *variety* of Γ	$\|\{(O_i, D_i, T_i) \mid \tau_i \in \Gamma\}\|$
κ_3	largest task parameter	$\max_{\tau_i \in \Gamma}\{C_i, D_i, T_i\}$
κ_4	number of periods	$\|\{T_i \mid \tau_i \in \Gamma\}\|$
κ_5	ratio of the largest period to the smallest period	$\lceil(\max_{\tau_i \in \Gamma}\{T_i\})/(\min_{\tau_i \in \Gamma}\{T_i\})\rceil$

Although Definition 3 does not restrict the form of the function $f(\cdot)$ other than that it be computable, of course, $f(\cdot)$ should not grow too quickly with increasing parameter values for an fpt-algorithm to be considered meaningful in practice.[3] We will see several schedulability-analysis algorithms that have fpt-algorithms for meaningful parameterizations (such as the parameterizations listed in Table I) in later sections; the function $f(\cdot)$ does not grow particularly rapidly with the parameter in all these fpt-algorithms.

The study of parameterized algorithms has yielded a diverse range of techniques: kernelization, bounded-depth search trees, iterative compression, dynamic programming on structures of bounded treewidth, to name a few [22]. The techniques used in the current document are quite elementary in most cases; in some cases, we use a non-trivial result of Lenstra for solving ILPs in fixed dimension but the applications of the result are quite straightforward, and the descriptions of its use are complete and self-contained.

IV. FIXED-PRIORITY SCHEDULING OF SYNCHRONOUS CONSTRAINED-DEADLINE SYSTEMS

In this section, we examine the FP-scheduling of constrained-deadline synchronous (i.e., either synchronous periodic or sporadic) task systems through the lens of fixed-parameter analysis. Let Γ denote a constrained-deadline synchronous task system comprising the n tasks $\tau_1, \tau_2, \ldots, \tau_n$. We assume, without loss of generality, that the tasks in Γ are indexed in decreasing priority order: τ_i has greater priority than τ_{i+1} for all $i, 1 \leq i < n$. Let us define $T_{\min}$, $T_{\max}$, and $D_{\max}$ as follows

$$\begin{aligned} T_{\min} &= \min_{\tau_i \in \Gamma}\{T_i\} \\ T_{\max} &= \max_{\tau_i \in \Gamma}\{T_i\} \\ D_{\max} &= \max_{\tau_i \in \Gamma}\{D_i\} \end{aligned}$$

As we had stated earlier, Response-Time Analysis (RTA) [8], [9] is widely used for the schedulability analysis of constrained-deadline synchronous systems. It is very straightforward to show that the running time of RTA on a task instance Γ with n tasks is $O(n^2 \times D_{\max})$, which can be written as

$$f(D_{\max}) \times |\Gamma|^2$$

Theorem 1 immediately follows:

Theorem 1. *If the parameter is κ_3, the largest number in the specification of the task system, then RTA is an fpt-algorithm for the fixed-priority schedulability analysis of constrained-deadline synchronous systems.*

(Notice that in this case, the $f(\cdot)$ function in the definition of fpt-algorithms (Definition 3) is just the identity function; this is a very slow-growing $f(\cdot)$, but we should remember that linear in the *value* of a number is exponential in the number of bits used to represent that number in the input.)

Theorem 1 above is essentially a restatement, in fixed-parameter analysis terminology, of the well-known result that RTA has pseudo-polynomial running time. In Theorem 2 below we apply the fixed-parameter analysis lens to obtain additional insight on RTA's performance: the dependence of the running time of RTA upon κ_5, the ratio of the largest to the smallest periods in the problem instance. Theorem 2 tells us that if κ_5 is small for a given task system Γ, RTA will be very efficient on Γ even though the underlying schedulability analysis problem is NP-hard [23]. (We will also see that $f(\cdot)$ here is the identity function, i.e., the dependence of the running time on the parameterization is linear. Hence, the combinatorial explosion inherent in the underlying NP-hard problem is contained in the value of the parameter: $T_{\max}/T_{\min}$ must be very large for hard instances.)

Theorem 2. *If the parameter is $\kappa_5 = \lceil(T_{\max}/T_{\min})\rceil$, then RTA is an fpt-algorithm for the fixed-priority schedulability analysis of constrained-deadline synchronous systems.*

Proof. Under RTA, the *worst-case response time* R_i of task τ_i is given by the smallest non-negative solution to the recurrence

$$R_i = C_i + \sum_{1 \leqslant j < i} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \qquad (1)$$

Now it is evident that the value of at least one ceiling expression (a "$\lceil\cdot\rceil$") increases in each iteration of Recurrence 1 (this is also stated formally as Theorem 1 in [24]). If no solution to the recurrence of value $\leq D_i$ is found, then we cease evaluating it any further, and conclude that τ_i cannot be guaranteed to meet

[3]This situation is somewhat analogous to the definition of the tractable complexity class P as being all decision problems for which polynomial-time algorithms exist: an algorithm with running time a degree-100 polynomial is not likely to be considered practically meaningful.

its deadline; hence, the largest value of a ceiling expression is $\lceil D_i/T_{\min}\rceil$. Since there are $(i-1)$ ceiling expressions, it follows that Recurrence 1 will execute no more than $(i-1)\times\lceil(D_i/T_{\min})\rceil$ times. The total number of iterations of Recurrence 1 needed for RTA to determine the schedulability of all n tasks in task system Γ is therefore no more than

$$\begin{aligned}
&\sum_{i=1}^{n}\left((i-1)\times\left\lceil\frac{D_i}{T_{\min}}\right\rceil\right)\\
&\quad\leq \sum_{i=1}^{n}\left((i-1)\times\left\lceil\frac{D_{\max}}{T_{\min}}\right\rceil\right)\\
&\quad= \left\lceil\frac{D_{\max}}{T_{\min}}\right\rceil\times\sum_{i=1}^{n}(i-1)\\
&\quad= \left\lceil\frac{D_{\max}}{T_{\min}}\right\rceil\times O(n^2)
\end{aligned}$$

Since each iteration of Recurrence 1 clearly involves no more than $O(n)$ arithmetic operations the total number of arithmetic operations needed to complete these $\lceil(D_{\max}/T_{\min})\rceil\times O(n^2)$ operations is

$$\left\lceil\frac{D_{\max}}{T_{\min}}\right\rceil\times O(n^3)\leq\left\lceil\frac{T_{\max}}{T_{\min}}\right\rceil\times O(n^3)$$

where we simplified using $D_{\max}\leq T_{\max}$ for a constrained-deadline sporadic task system Γ. But the expression above can be written as

$$f\left(\left\lceil\frac{T_{\max}}{T_{\min}}\right\rceil\right)\times O(n^3) \tag{2}$$

where $f(\cdot)$ is the identity function and which fits Definition 3's specification of the running time of an fpt-algorithm. □

(In this case as well, the $f(\cdot)$ function in the definition of fpt-algorithms (Definition 3) is the identity function.)

We have seen that RTA is an fpt-algorithm for the parameters κ_3 and κ_5. When we choose any of κ_1,κ_2 or κ_4 as the parameters, however, straightforward implementations of RTA are no longer fpt-algorithms for the fixed-priority schedulability analysis of constrained-deadline synchronous systems, as illustrated in the following example.

Example 2. Consider the following system Γ of two tasks, where y is any positive integer:[4]

	C_i	D_i	T_i
τ_1	$(y-1)$	y	y
τ_2	y	$D_2\geq y^2$	$T_2\geq D_2$

In computing the response time R_2 of task τ_2, most implementations of RTA would initialize $R_2\leftarrow y$ and successively take on the values $2y-1, 3y-2, 4y-3, \ldots$ until finally converging at y^2. Hence, the straightforward implementation of RTA would have $\Theta(y)$ running time on the task system Γ in the example above. However, its parameters κ_1, κ_2, and κ_4 are each equal to two; therefore, RTA's running time, which is pseudo-polynomial in the representation of Γ, cannot be represented in the form needed by Definition 3 and therefore does not constitute an fpt-algorithm with respect to these parameters. □

[4]Notice that for this instance the ratio of the largest to the smallest period – the parameter κ_5– is at least y^2/y or y. Expression 2 in the proof of Theorem 2 still yields a pseudo-polynomial bound for RTA on this instance.

The HET algorithm [18], [19]. An algorithm called the Hyperplanes Exact Test (HET) has also been proposed for fixed-priority schedulability analysis of constrained-deadline synchronous systems [18], [19]. We will show that HET is an fpt-algorithm with respect to parameters κ_1, κ_2, κ_3, κ_4, and κ_5.

Recall that τ_i is guaranteed to always meet its deadline if and only if there exists a $t\in[0,D_i]$ that satisfies

$$C_i+\sum_{1\leqslant j<i}\left\lceil\frac{t}{T_j}\right\rceil C_j\leqslant t. \tag{3}$$

The correctness of HET depends on the observation that if tasks $\tau_1,\ldots,\tau_{i-1}$ are guaranteed to always meet their deadlines then there is some $t\in[0,D_i]$ satisfying Inequality 3 if and only if there is such a t in the set $\mathcal{P}_{i-1}(D_i)$, where $\mathcal{P}_j(t)$ is defined inductively as

$$\begin{aligned}
&\mathcal{P}_0(t)=\{t\},\\
&\mathcal{P}_j(t)=\mathcal{P}_{j-1}\left(\left\lfloor\frac{t}{T_j}\right\rfloor T_j\right)\cup\mathcal{P}_{j-1}(t),\quad \text{if } j>1.
\end{aligned}$$

It was also shown that the cardinality of $\mathcal{P}_i(t)$ is at most 2^i for any t [19, p. 1466]; therefore, $\mathcal{P}_{i-1}(D_i)$ has at most 2^{i-1} elements. Testing inequality 3 for each element takes $O(i)$ arithmetic operations, from which it follows that determining whether τ_i is guaranteed to always meet its deadline takes at most $2^{i-1}\times O(i)$ arithmetic operations.

The FP-schedulability analysis of Γ requires us to determine whether each of the $n=|\Gamma|$ tasks $\tau_1,\tau_2,\ldots,\tau_n$ in Γ is guaranteed to meet its deadline. Applying the above algorithm for each task in order would therefore take $\sum_{i=1}^{n}\left(2^{i-1}\times O(i)\right)$ arithmetic operations, which is clearly bounded from above by

$$2^n\times O(n^2)\ . \tag{4}$$

Theorem 3 follows.

Theorem 3. *If the parameter is κ_1, the cardinality of Γ, then HET is an fpt-algorithm for the fixed-priority schedulability analysis of constrained-deadline synchronous systems.*

Notice that having n_i tasks in Γ with exactly the same parameters (C_i,D_i,T_i) (and the same priority) is functionally equivalent, in terms of preemptive FP-schedulability, to having a single task with the same relative deadline D_i and the same period T_i, and with the C_i parameter inflated by a factor n_i; Theorem 4 below immediately follows from Theorem 3.

Theorem 4. *If the parameter is κ_2, the variety of Γ, then HET is an fpt-algorithm for the fixed-priority schedulability analysis of constrained-deadline synchronous systems.*

We can also extend the result of Theorem 3 to be applicable when the parameter is κ_4, the number of distinct periods in

Γ. We observe that in FP scheduling, the relative deadline parameters of tasks of higher priority than τ_i's do not play a role in determining whether τ_i is guaranteed to always meet its deadline. Hence, when seeking to determine whether τ_i is guaranteed to always meet its deadline (as in the proof of Theorem 3), we could effectively "merge" all higher priority tasks of the same period but different deadlines into a single task. Upon doing this, it may be verified that the arguments of the proof of Theorem 3 continue to hold, with the modification that the cardinality of $\mathcal{P}_{i-1}(D_i)$ is always no greater than 2^{κ_4}, and hence the total number of arithmetic operations needed for the FP-schedulability analysis of Γ (the analog of Expression 4) is at most

$$2^{\kappa_4} \times O(n^2). \tag{5}$$

Theorem 5. *If the parameter is κ_4, the number of periods, then HET is an fpt-algorithm for the fixed-priority schedulability analysis of constrained-deadline synchronous systems.*

Finally, from the definition of set $\mathcal{P}_j(t)$, it can be shown by induction that

$$\mathcal{P}_{i-1}(D_i) \subseteq \{D_i\} \cup \{kT_j \mid k \in \mathbb{N}^+, j < i, kT_j \leq D_i\}$$

Therefore, $\mathcal{P}_{i-1}(D_i)$ has cardinality

$$O(n \times \lceil D_{\max}/T_{\min} \rceil)$$

Using the definitions of κ_3 and κ_5, it immediately follows that the cardinality of $\mathcal{P}_{i-1}(D_i)$ also is $O(n\kappa_3)$ and $O(n\kappa_5)$. Applying HET for one task therefore takes $O(n^2\kappa_3)$ and $O(n^2\kappa_5)$ arithmetic operations. Thus, the total number of arithmetic operations performed by HET is $O(n^3\kappa_3)$ and $O(n^3\kappa_5)$.

Theorem 6. *If the parameter is κ_3 or κ_5, then HET is an fpt-algorithm for the fixed-priority schedulability analysis of constrained-deadline synchronous systems.*

We summarize the results for fixed-priority schedulability analysis in the following table:

param.	meaning	fpt-algorithm
κ_1	cardinality	HET (Thm 3)
κ_2	variety	HET (Thm 4)
κ_3	largest number	RTA (Thm 1) and HET (Thm 6)
κ_4	no. of distinct periods	HET (Thm 5)
κ_5	max ratio of periods	RTA (Thm 2) and HET (Thm 6)

V. EDF Scheduling

In this section we conduct fixed-parameter analyses of EDF schedulability for arbitrary-deadline periodic task systems and constrained-deadline synchronous task systems.

We begin with an examination of the EDF scheduling of arbitrary-deadline periodic task systems, parameterized by their κ_2, the variety of Γ. Recall from Table I that the variety of Γ is the number of distinct (offset, deadline, period) triples in the system:

$$\kappa_2 = |\{(O_i, D_i, T_i) \mid \tau_i \in \Gamma\}|$$

We refer to each distinct triple as a *kind* of task. There are κ_2 kinds of tasks in the system. For each $j \in \{1, 2, \ldots, \kappa_2\}$, we assume that there are n_j tasks of the j'th kind in Γ. Thus, there are $\sum_{j=1}^{\kappa_2} n_j$ tasks in Γ.

An fpt-algorithm is known for the ILP feasibility problem parameterized by the number of variables: building upon prior work by Lenstra [25] and Kannan [26], Frank and Tardos [27] have derived an algorithm for determining whether an ILP is feasible that uses $O(p^{2.5p+o(p)} \times L)$ arithmetic operations, where p is the number of integer variables and L is the size of the representation of the ILP (see also the work of Eisenbrand et al. [28]). The important observation here is that the occurrence of p in the exponent is completely contained in the first term and therefore $O(p^{2.5p+o(p)} \times L)$ can be written as $O(f(p) \times L)$. The problem of deciding whether a task system is *not* EDF-schedulable has an integer programming formulation with $O(\kappa_2)$ variables [29]. Moreover, the size of this formulation is linearly bounded by the size of the representation of Γ, denoted $\langle\Gamma\rangle$. Thus, the Frank and Tardos [27] algorithm can solve this particular ILP in $O(f'(\kappa_2) \times \langle\Gamma\rangle)$ arithmetic operations. Theorem 7 follows.

Theorem 7. *If the parameter is κ_2, the variety of Γ, then the EDF schedulability analysis of arbitrary-deadline periodic systems can be solved by an fpt-algorithm consisting of formulating the problem as an ILP [29] and solving the formulation using the algorithm of Frank and Tardos [27].*

An obvious corollary to Theorem 7 is that this algorithm is also an fpt-algorithm for the EDF schedulability analysis of arbitrary-deadline periodic systems with parameter κ_1, the cardinality of Γ.

A note. We point out that the statement of Theorem 7 above both generalizes and refines [29, Theorem 3.5], which states "For [...] task systems with a fixed number of distinct types of tasks, the feasibility problem for one processor can be solved in polynomial time". Theorem 7 is first a generalization of this result from polynomial-time algorithms to fpt-algorithms, and also a refinement in the sense that [29, Theorem 3.5] leaves open the possibility that the polynomial-time algorithm could have had the parameter in the exponent (i.e., it could have been an xp-algorithm rather than an fpt-algorithm).

Our remaining results for EDF-schedulability analysis are for synchronous task systems only. For such task systems, *Processor Demand Analysis (PDA)* [10] is an exact technique for EDF-schedulability analysis upon preemptive uniprocessors. This technique is based on the concept of the *demand bound function* (dbf): for any synchronous task $\tau_i = (C_i, D_i, T_i)$ and any interval-duration $t \geq 0$, $\text{dbf}(\tau_i, t)$ denotes the maximum possible cumulative execution requirement by jobs of task τ_i that both arrive in, and have their deadlines within, any contiguous interval of duration t. The following formula for computing $\mathrm{dbf}(\tau_i, t)$ was derived in [10]:

$$\mathrm{dbf}(\tau_i, t) = \max\left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1, 0\right) \times C_i \qquad (6)$$

and it was shown that a necessary and sufficient condition for $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be EDF-schedulable on a preemptive unit speed processor is that the following condition hold for all integers $t \geq 0$:

$$\sum_{i=1}^{n} \mathrm{dbf}(\tau_i, t) \leq t \qquad (7)$$

Letting $H(\Gamma)$, $U(\Gamma)$, and $T_{\max}$ respectively denote the hyperperiod (i.e., the least common multiple of the period parameters), the system utilization, and the largest period of any task in the task system Γ under consideration, it has been shown [29], [30] that if Expression 7 is violated for any t, then it is violated for some t that is no larger than

$$\min\left(H(\Gamma), \left(\frac{U(\Gamma)}{1 - U(\Gamma)}\right) \times T_{\max}\right) \qquad (8)$$

If the utilization bound is some constant $c < 1$, then it follows that the upper bound of Expression 8 is no larger than $(c/(1-c)) \times T_{\max}$. Thus, for each i, $O(T_{\max})$ values of t need to be checked, and a total of $O(T_{\max} \times n)$ values need to be checked. Since checking Condition 7 for a given t requires $O(n)$ arithmetic operations, PDA uses

$$O\left(T_{\max} \times n^2\right)$$

arithmetic operations. Comparing this form with the one specified (Definition 3) for fpt-algorithms, we conclude that

Theorem 8. *If the parameter is κ_3, the largest number in the specification of the task system, then PDA is an fpt-algorithm for the EDF schedulability analysis of bounded-utilization constrained-deadline synchronous systems.*

Similar to Theorem 1, the above theorem is a restatement, in the terminology of fixed-parameter analysis, of the well-known result [10] that PDA runs in pseudo-polynomial time for bounded-utilization synchronous systems.

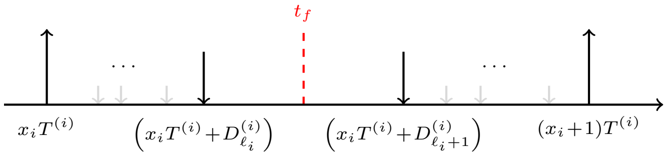
It was also shown in [10] that Condition 7 need only be checked for values of t that are of the form $t = (k \times T_i + D_i)$ for some non-negative integer k and some $i, 1 \leq i \leq n$. By this we can utilize the stronger observation that for each i, only $O(\lceil T_{\max}/T_{\min} \rceil)$ values of t need to be checked to conclude that PDA uses

$$O\left(\left\lceil \frac{T_{\max}}{T_{\min}} \right\rceil \times n^2\right)$$

arithmetic operations. Theorem 9 follows.

Theorem 9. *If the parameter is $\kappa_5 = \lceil T_{\max}/T_{\min} \rceil$, then PDA is an fpt-algorithm for the EDF schedulability analysis of bounded-utilization constrained-deadline synchronous systems.*

Our final parameterized algorithm for EDF schedulability analysis of synchronous task systems is for the parameterization κ_4, the number of distinct periods in the task system Γ. We were not able to obtain an fpt-algorithm for this problem; we present an xp-algorithm instead.


t_f
$\cdots$
$\cdots$
$x_i T^{(i)}$
$\left(x_i T^{(i)} + D^{(i)}_{\ell_i}\right)$
$\left(x_i T^{(i)} + D^{(i)}_{\ell_i+1}\right)$
$(x_i+1)T^{(i)}$
Fig. 1. Notation for the xp-algorithm for EDF-schedulability of synchronous systems with the parameterization *number of distinct periods*.

First, we find it convenient to introduce some additional notation for representing the task system. Let the periods in Γ be denoted by $T^{(1)}, T^{(2)}, \ldots, T^{(\kappa_4)}$. For each $j \in \{1, 2, \ldots, \kappa_4\}$,

- Let n_j denote the number of tasks with period $T^{(j)}$ (hence, Γ has a total of $n = (n_1 + n_2 + \cdots + n_{\kappa_4})$ tasks).
- Let $C^{(j)}_1, C^{(j)}_2, \ldots, C^{(j)}_{n_j}$ denote the execution requirements and $D^{(j)}_1, D^{(j)}_2, \ldots, D^{(j)}_{n_j}$ the corresponding relative deadline parameters for the n_j tasks with period $T^{(j)}$.
- We assume without loss of generality that tasks with equal period are indexed according to relative deadline: $D^{(j)}_1 \leq D^{(j)}_2 \leq \cdots \leq D^{(j)}_{n_j}$.

Suppose that the instance is unschedulable, and let t_f denote the earliest instant at which a deadline is missed. Let x_i denote the number of complete periods of $T^{(i)}$ that are fully within $[0, t_f]$

$$x_i = \left\lfloor \frac{t_f}{T_i} \right\rfloor$$

Let $\ell_1, \ell_2, \ldots \ell_{\kappa_4}$ denote non-negative integers such that out of the n_i jobs with period $T^{(i)}$ that are released at time $x_i T^{(i)}$, ℓ_i jobs have deadlines $\leq t_f$—see Figure 1. From the figure, it is evident that the deadline miss at t_f occurs if and only if a deadline miss at t_f occurs for the task system created by replacing the first ℓ_i tasks with period $T^{(i)}$ by the task

$$\left(\sum_{j=1}^{\ell_i} C^{(i)}_j, D^{(i)}_{\ell_i}, T^{(i)}\right),$$

and by replacing the next $n_i - \ell_i$ tasks by the task

$$\left(\sum_{j=\ell_i+1}^{n_i} C^{(i)}_j, T^{(i)}, T^{(i)}\right)$$

for each $i \in \{1, 2, \ldots, \kappa_4\}$. The variety of this new task system is at most $2\kappa_4$ since there are κ_4 periods, at most two deadlines per period, and all offsets are 0. Using Theorem 7, its schedulability can be determined by an fpt-algorithm for parameter κ_4. However, we do not know the values of $\ell_1, \ell_2, \ldots, \ell_{\kappa_4}$ that cause the deadline miss. We can simply try all

$$(n_1 + 1) \times (n_2 + 1) \times \cdots \times (n_{\kappa_4} + 1) = O(n^{\kappa_4})$$

possible values for the vector ℓ. The overall algorithm consists of the following parts:

- Nondeterministically guess the vector ℓ which causes the deadline miss in $O(n^{\kappa_4})$ steps.
- Create the reduced task system, using ℓ, as described above, using $O(n)$ arithmetic operations.
- Run the fpt-algorithm described in Theorem 7 on the reduced task system. The fpt-algorithm uses $O(f'(\kappa_4) \times \langle\Gamma\rangle)$ arithmetic operations.

The overall algorithm needs

$$O(n^{\kappa_4} \times f'(\kappa_4) \times \langle\Gamma\rangle \times n)$$

arithmetic operations. While the above is unlikely to be a practical algorithm, it matches the requirement on running time for an algorithm to be considered an xp-algorithm (see Definition 2); Theorem 10 follows.

Theorem 10. *If the parameter is κ_4, the number of periods, then there exists an xp-algorithm for the EDF-schedulability analysis of synchronous systems.*

The various results concerning EDF-schedulability that we have presented in this section are summarized in the table below:

param	meaning	algorithm
κ_1	cardinality	an fpt-algorithm (Thm 7)
κ_2	variety	an fpt-algorithm (Thm 7)
κ_3	largest number	PDA is an fpt-algorithm for synch (Thm 8)
κ_4	no. of periods	an xp-algorithm for synch (Thm 10)
κ_5	max ratio of periods	PDA is an fpt-algorithm for synch (Thm 9)

VI. Intractability of Parameterized Problems

We have seen that fixed-parameter analysis of problems can be used to show how some problems are tractable if some given parameter is small enough, even though the same problems could be, for example, NP-hard in the classical (non-parameterized) sense. We have seen examples of (co)NP-hard scheduling problems that are fixed-parameter tractable for some different natural parameterizations.

Parameterized complexity theory has also developed methods for showing (or at least for giving strong evidence) that certain problems are *not* tractable despite a given parameter being small. In this section we will review some of the complexity classes in parameterized complexity theory[5] and show that some well-known scheduling problems are likely not fixed-parameter tractable for certain natural parameterizations.

A. Parameterized complexity classes

FPT, the class of all parameterized problems for which there are fpt-algorithms, is the basic complexity class for problems that are considered tractable in parameterized complexity theory, similar to how P is often considered to be the class of tractable problems in classical complexity. The class XP contains all parameterized problems for which there are xp-algorithms (see, e.g., Theorem 10 above), but XP-hard problems are *not* considered as tractable for their parameterization. It is known, through a diagonalization argument using the time hierarchy theorem, that FPT is strictly contained in XP: FPT $\subsetneq$ XP.

There are also many complexity classes in between FPT and XP, for example the infinite W-hierarchy, which contains classes called W[1], W[2], etc. The classes in the W-hierarchy are defined as containing all those parameterized problems that are reducible via an fpt-reduction (see Definition 4 below) to canonical problems based on boolean decision circuits. It is known that

$$\text{FPT} \subseteq \text{W[1]} \subseteq \text{W[2]} \subseteq \cdots \subseteq \text{XP}$$

and these containments are generally believed to be strict (in fact, at least one must be strict since FPT $\subsetneq$ XP).

Instead of boolean decision circuits it may be more illuminating, in the context of real-time scheduling theory, to consider the many other natural and well-known problems that are now known to be complete at the lower levels of the W-hierarchy.[6] An archetypical W[1]-complete problem is CLIQUE parameterized by the clique size k, i.e., the decision problem of whether a given undirected graph G contains the complete graph K_k on k vertices as a subgraph. (We point out that CLIQUE was previously used by Leung and Whitehead [16] to establish the hardness of several basic real-time scheduling problems. We will revisit the reductions of Leung and Whitehead later in this section.)

It is known that if FPT = W[1], then the Exponential Time Hypothesis (ETH) [32], which asserts that 3-SAT and several other NP-complete problems cannot have sub-exponential time algorithms (see Definition 5), must be false. Already showing that a parameterized problem is W[1]-hard, for example by reducing from CLIQUE, therefore offers fairly strong evidence that it is not in FPT.

A parameterized problem that is NP-hard for some *constant* value of the parameter is called para-NP-hard. An example is GRAPH VERTEX COLORING parameterized by the number k of colors, as GRAPH VERTEX COLORING is NP-hard for any constant $k \geq 3$ of colors. It is known that FPT = para-NP if and only if P = NP. The classes XP and para-NP both contain the entire W-hierarchy, but neither contains the other (assuming P $\neq$ NP). All para-NP-hard problems are therefore outside XP and vice versa. Similar to para-NP-hard we can define para-coNP-hard, etc.

B. Parameterized reductions

To classify parameterized problems into complexity classes we need an appropriate type of reduction that can transform one parameterized problem into another. The standard polynomial-time many-one reduction used in the classical theory of NP-completeness is not fully satisfactory here as it lacks any notion of the parameterization, making it at once too weak and too powerful for some of our purposes. A reduction that can be used

[5]The discussion here about parameterized complexity classes is by necessity very brief and very incomplete. For much more in-depth coverage of these topics (and several more complexity classes), see for example the book by Downey and Fellows [4].

[6]See for example [31] for a compendium of such problems.

between parameterized problems is instead the *fpt-reduction*, defined below.

Definition 4 (fpt-reduction)**.** Let A and B be two problems parameterized by κ and κ', respectively. An fpt-reduction from (A, κ) to (B, κ') is a mapping f such that

1) $x \in A$ if and only if $f(x) \in B$ for all x.
2) $f(x)$ is computable by an fpt-algorithm with respect to parameter κ.
3) There exists a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(f(x)) \leqslant g(\kappa(x))$.

We note that an fpt-reduction is many-one (point 1 above), i.e., it maps yes-instances x of the first problem to yes-instances $f(x)$ of the second problem, and likewise for no-instances. However, in contrast to a polynomial-time many-one reduction it does not need to be computable in polynomial time, it is enough that it is computable by some fpt-algorithm with respect to parameter κ (point 2 above). On the other hand, it is also restricted in that it must produce a parameter κ' for the second problem that is bounded by some function only on the parameter κ of the first problem (point 3 above).

It is not difficult to see that the fpt-reduction has the basic property that if there is an fpt-reduction between parameterized problems (A, κ) and (B, κ'), and (B, κ') is in FPT, then (A, κ) is in FPT as well. Of course, it then also holds that if (A, κ) fpt-reduces to (B, κ') and (A, κ) is *not* in FPT, then neither is (B, κ'). We can therefore use fpt-reductions to transfer the hardness of one problem to another. For example, if (A, κ) is W[1]-hard and fpt-reduces to (B, κ'), then (B, κ') is also W[1]-hard.

C. Intractability results for some scheduling problems

In this section we show that some basic uniprocessor schedulability problems are unlikely to be in FPT for certain parameterizations. This will follow quite directly by reinterpreting some known classical reductions.

Leung and Whitehead [16] presented (polynomial-time many-one) reductions from CLIQUE to the SIMULTANEOUS CONGRUENCES PROBLEM (SCP), and from SCP to the *un*schedulability problem for asynchronous periodic tasks with constrained deadlines. They described the last reduction for fixed-priority unschedulability, but it is easy to see from their proof that it applies equally well to any work-conserving scheduler, and thus the schedulability problem for any work-conserving scheduler for asynchronous periodic tasks with constrained deadlines is coNP-hard.

If we consider the CLIQUE problem as parameterized by the clique size k, we are interested in seeing what happens to k while going through the two reductions of Leung and Whitehead. We will not go into the details of the reductions (they are given in Theorems 3.6 and 3.8 in [16], respectively), but we will simply note that given a graph $G = (V, E)$ as the CLIQUE instance, they construct a task set of $|V|$ tasks with different offsets and periods, but where every task τ_i has the same execution time and deadline: $C_i = 1$ and $D_i = k - 1$, where k was the clique size.[7] We note then that the schedulability problem is coNP-hard even when the number of distinct deadlines or execution times is a constant (in fact, even when that constant is just 1) and when the maximum execution time is a constant (again, even if it is simply 1). We immediately get the following as a corollary.

Theorem 11. *The schedulability problem of any work-conserving scheduler for asynchronous periodic tasks with constrained deadlines is para-coNP-hard under each of the following parameterizations:*

- *the number of distinct deadlines,*
- *the number of distinct execution times, and*
- *the maximum execution time.*

Contrast this result to the case where the parameter is the *variety* of the task system (the number of distinct combinations of offsets, deadlines, and periods), where we have seen that the same EDF-schedulability problem is in FPT (Theorem 7). We can also contrast Theorem 11 to the case where the parameter is the number of distinct *periods*. There we have seen that, at least with *synchronous* tasks, the FP-schedulability problem is in FPT (Theorem 5) and the EDF-schedulability problem is in XP (Theorem 10) and therefore neither can be para-coNP-hard unless P = NP. These observations offer some indication that *limiting the number of distinct periods generally is more important for the tractability of schedulability problems than limiting the other parameters.*

Looking again at the reductions of Leung and Whitehead we note that every task created has $D_i = k - 1$, where k was the original clique size. This is not a constant, so we cannot conclude that the schedulability problem is para-coNP-hard parameterized by the maximum deadline. However, we can trivially reinterpret Leung and Whitehead's reduction as an fpt-reduction where the parameter of the target problem is the maximum deadline. Since CLIQUE is W[1]-hard parameterized by the clique size, we then get the following.

Theorem 12. *The schedulability problem of any work-conserving scheduler for asynchronous periodic tasks with constrained deadlines is W[1]-hard when parameterized by the maximum deadline.*

This can be contrasted to the case with synchronous tasks, where FP-schedulability is in FPT when parameterized by the maximum deadline (follows from Theorem 1).

Finally, we make another observation from a known classical reduction. Ekberg and Yi [33] showed that the EDF-schedulability (or feasibility) problem for synchronous tasks with constrained deadlines is coNP-hard through a reduction that produces tasks that all have $C_i = 1$. Similar to Theorem 11, we immediately get the following.

[7]Here we have also scaled the task parameters to integers, whereas Leung and Whitehead produced rational task parameters.

Theorem 13. *The EDF-schedulability problem for synchronous tasks with constrained deadlines is para-coNP-hard under both of the following parameterizations:*

- *the number of distinct execution times, and*
- *the maximum execution time.*

D. Lower bounds on problems inside FPT

Here we will consider more fine-grained lower bounds on the runtime for problems that are in FPT. We will see that some parameterized scheduling problems cannot be solved in time that is sub-exponential in the parameter, given a plausible conjecture in complexity theory.

As we have seen, problems in FPT can be solved in time $O(f(k)\times|x|^c)$, where $|x|$ is the size of the input, c is a constant, $k=\kappa(x)$ is the parameter, and f is any computable function. While such problems are typically considered tractable in fixed-parameter analysis, the running time of such an fpt-algorithm of course strongly depends on the size of the constant c and the growth of function f. Given a parameterized problem in FPT we might wonder if the constant c can be reduced. Ideally, if $c=1$, then the problem is solved by some algorithm that is only *linear* in the input size (but still potentially exponential or worse in the parameter); such problems are said to be in complexity class FPL (for Fixed-Parameter *Linear*), and it is known that FPL $\subsetneq$ FPT. We can give upper bounds on c for a problem by simply providing an appropriate fpt-algorithm for it, but giving lower bounds on c would generally be very difficult.

We might also wonder if the function f can be replaced by a slower-growing one. Here we can use parameterized complexity theory to prove lower bounds on the growth of function f, given that the Exponential Time Hypothesis (ETH) is true. The ETH is a hypothesis[8] formulated by Impagliazzo and Paturi [32] that states the following.

Definition 5 (ETH)**.** There exists a $\delta>0$ such that no algorithm can solve 3-SAT in time $O(2^{\delta n})$, where n is the number of variables.

(A somewhat simpler form that is implied by the ETH is that 3-SAT cannot be solved in time $2^{o(n)}$.)

Note that the ETH says that 3-SAT cannot be solved in time that is sub-exponential in the number of variables, but it says nothing about the number of clauses. The *sparsification lemma* by Impagliazzo et al. [34] shows how a 3-SAT instance can be replaced by disjunction of 3-SAT formulas that are sparse, i.e., that have $m=O(n)$, where n is the number of variables and m is the number of clauses. A corollary of the sparsification lemma is the following.

Theorem 14 (see, for example, Theorem 14.4 in [22])**.** *If the ETH is true, then* 3-SAT *cannot be solved in time* $2^{o(n+m)}$.

An immediate consequence of Theorem 14 is that if we consider 3-SAT as a parameterized problem with parameter $n+m$, and show that it can be reduced via a polynomial-time reduction to another parameterized problem A such that the parameter for A is $\kappa=O(n+m)$, then A cannot be solved by an fpt-algorithm in time $O(2^{o(\kappa(x))}\times|x|^c)$ if the ETH holds (see, for example, Observation 14.10 in [22]). In other words, it would mean that problem A cannot be solved in time $O(f(\kappa(x))\times|x|^c)$ for any sub-exponential f.

[8]The majority of researchers today seem to think that ETH is likely true, but one should not overstate the evidence for it. The ETH is a much stronger assumption than P $\neq$ NP.

We can now, again, reinterpret some known classical reductions in scheduling theory to achieve new results on their parameterized complexity. First we consider the schedulability problem for asynchronous periodic tasks.

Baruah et al. [29] presented a reduction from 3-SAT to SCP as an alternative to the reduction from CLIQUE to SCP by Leung and Whitehead [16]. The original motivation for the alternative reduction was to show that SCP is *strongly* NP-hard, but here we will instead exploit its consequences based on Theorem 14. The reduction from 3-SAT to SCP by Baruah et al. (Theorem 3.2 in [29]) composed with the reduction from SCP to unschedulability for asynchronous periodic tasks by Leung and Whitehead (Theorem 3.8 in [16]) creates task sets with $2n+3m=O(n+m)$ tasks, where n is the number of variables and m the number of clauses in the 3-SAT formula. By the reasoning above we therefore get the following result.

Theorem 15. *The schedulability problem of any work-conserving scheduler for asynchronous periodic tasks with constrained deadlines cannot be solved in time $O(2^{o(k)}\times|x|^c)$, where k is the number of tasks, if the ETH holds.*

We already know that the EDF-schedulability problem for asynchronous periodic tasks (even with arbitrary deadlines) is in FPT when parameterized by the number of distinct kinds of tasks (Theorem 7). The result in Theorem 15 states, however, that the dependence on the runtime of that parameter (or even the total number of tasks) cannot be sub-exponential if the ETH holds.

Second we consider the EDF-schedulability problem for synchronous tasks. Here we can similarly compose the reduction from 3-SAT to SCP by Baruah et al. [29] with the reduction from SCP to EDF-unschedulability by Ekberg and Yi [33] to create task sets with $O(n+m)$ number of distinct periods. From this we can again provide a lower bound on the runtime.

Theorem 16. *The EDF-schedulability problem for synchronous tasks with constrained deadlines cannot be solved in time $O(2^{o(k)}\times|x|^c)$, where k is the number of distinct periods, if the ETH holds.*

We do not know yet if this schedulability problem is in FPT when the parameter is the number of distinct periods (Theorem 10 gives an xp-algorithm), but if it is, then Theorem 16 shows that it still cannot be solved in time that is sub-exponential in the parameter, given that the ETH holds.

VII. Context and Conclusions

Fixed-parameter analysis of algorithms is generating considerable interest in the Algorithms community as a potential means of dealing with the inherent computational complexity of many important problems.[9] Many of the basic and foundational problems in real-time scheduling are known to be computationally intractable; it therefore seems appropriate that the real-time scheduling theory community investigates the applicability of ideas from fixed-parameter analysis to our scheduling problems. This paper reports on our efforts at doing so. We have examined several basic real-time scheduling problems that are known to be intractable (NP-hard and coNP-hard) via the lens of fixed-parameter analysis. We have shown that appropriate parameterizations of these problems may allow one to identify root causes of intractability; some parameterizations admit to fpt-algorithms while others seem to not do so (recall our observation following Theorem 11 that in order to achieve tractability it appears that limiting the number of distinct periods is more important than limiting the other parameters.) We have obtained parameterized characterizations of widely-used real-time schedulability-analysis algorithms (including RTA [8], [9] and PDA [10]), and we use these characterizations to help explain why these algorithms perform excellently upon some kinds of systems but not upon others. We have classified individual schedulability-analysis problems into parameterized complexity classes (FPT, the W-hierarchy, and XP), and have shown hardness results for some particular problems.

We close with the caveat that although useful, such lower bounds do not tell the whole story—it is important to note that one algorithm being FPT for a particular parameter does not necessarily make it better than another algorithm that is not (known to be) FPT, even when that parameter happens to be small. It is well-known that the best polynomial-time algorithms are not always faster than even exponential time algorithms in practice.[10] We note that Bini and Buttazzo [19] reported favorable running times for HET on random task sets compared to RTA, while Davis et al. [36] later reported the opposite with different settings for the random task set generation, which they argued resulted in more representative task sets. Clearly, and unsurprisingly, different algorithms perform differently depending on the particulars of what the input looks like, sometimes for obvious reasons, but sometimes due to much more subtle structural differences in the input. The real-time systems community is well-versed in investigating and analyzing such differences for average running times over carefully crafted sets of representative inputs (which the above is an example of). We believe that fixed-parameter analysis is a useful complement to this, in that it can analytically explain differences in worst-case running times for different inputs. The choice of interesting parameters and the resulting analysis is not always straightforward, and a much more structured study than presented in this initial work would be needed to truly understand the differences in worst-case parameterized running times between, say, HET and RTA, with its variants.

[9]Indeed, a chapter on this topic [5] is the first technical one in a recent book [35] titled "Beyond the Worst-Case Analysis of Algorithms".

[10]A famous example is of course the simplex algorithm for solving linear programs, which has exponential worst-case complexity but in practice outperforms the best polynomial-time algorithms.

APPENDIX
A CAUTIONARY TALE ON THE CHOICE OF PARAMETER

We will round off by demonstrating concretely how care has to be taken with the choice of parameterization, lest it turn algorithms into fpt-algorithms for mundane reasons that offer no new insights. In particular, we will see that parameterization κ_3, the largest numerical task parameter in the task set, can act like this.

Let Γ be some asynchronous periodic task set of n tasks. First we note that for any task $\tau_i \in \Gamma$ we have

$$\frac{C_i}{T_i} \geq \frac{1}{T_{\max}},$$

and therefore we have

$$\mathrm{U}(\Gamma) \geq \frac{n}{T_{\max}}.$$

If $\mathrm{U}(\Gamma) \leq 1$ we must then have

$$\kappa_3 \geq T_{\max} \geq n. \tag{9}$$

To encode Γ into a bit-string we need to encode four numbers (the task parameters) per task, each of which is no larger than κ_3. So the bit-size b of an encoding of Γ is $O(n \log(\kappa_3))$, but by Eq. 9 we then have

$$b = O(\kappa_3 \log(\kappa_3)). \tag{10}$$

In other words, the *size of the input* can be bounded by some function $f(\kappa_3)$ that depends only on κ_3.

Say that we have some schedulability analysis algorithm $\mathcal{A}$ on task sets Γ that is by no reasonable measure to be considered efficient, perhaps its running time is double-exponential, 2^{2^b}, where b is the size of the input. By Eq. 10 we have

$$2^{2^b} = 2^{2^{O(\kappa_3 \log(\kappa_3))}},$$

and therefore there is a computable function $f(\cdot)$, such that $f(\kappa_3) \geq 2^{2^b}$. Now we can rewrite the running time of algorithm $\mathcal{A}$ as $f(\kappa_3) \times b^0$, and therefore $\mathcal{A}$ is in fact an fpt-algorithm when parameterized by κ_3.

Clearly, $\mathcal{A}$ was here established as an fpt-algorithm for entirely mundane reasons: $\mathcal{A}$ is still the same very inefficient algorithm. The underlying reason that it could be shown to be an fpt-algorithm under κ_3 is simply that κ_3 is *too large for a good parameterization* — it allowed us to bound the size of the input by a function on the parameter.

We should note that when κ_3 was used in Theorems 1 and 8, we did not exploit it like this: the function $f(\cdot)$ used there is the identity function. While the definition of an fpt-algorithm allows any function $f(\cdot)$, the growth of it is important from a practical perspective. The main purpose of introducing κ_3 in this paper was for Theorems 1 and 8 to state, in the terminology of fixed-parameter analysis, the known results that some popular schedulability algorithms run in pseudo-polynomial time. It then led to Theorems 2 and 9, which use the much more useful parameterization κ_5 — the ratio between largest and smallest period — that can stay small even for large inputs. Finally, κ_3 allows us to make the point of this appendix!

References

[1] Pontus Ekberg and Wang Yi. *Complexity of Uniprocessor Scheduling Analysis*, pages 1–18. Springer Singapore, Singapore, 2019.

[2] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[3] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[4] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[5] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. Parameterized algorithms. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 27–51. Cambridge University Press, 2021.

[6] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[7] William Gasarch and Keung Kin. Parameterized complexity: A joint review of two new books and a preview of a third. *The Computer Journal*, 51, 01 2008.

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

[9] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the Real-Time Systems Symposium - 1989*, pages 166–171, Santa Monica, California, USA, December 1989. IEEE Computer Society Press.

[10] 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.

[11] Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. In Jon Lee and Jens Vygen, editors, *Integer Programming and Combinatorial Optimization - 17th International Conference, IPCO 2014, Bonn, Germany, June 23-25, 2014. Proceedings*, volume 8494 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 2014.

[12] Matthias Mnich and Andreas Wiese. Scheduling and fixed-parameter tractability. *Math. Program.*, 154(1-2):533–562, 2015.

[13] Nikhil Bansal and Kirk Pruhs. The geometry of scheduling. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 407–414. IEEE Computer Society, 2010.

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

[15] 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.

[16] Joseph Y.-T Leung and Jennifer Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[17] 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.

[18] Y. Manabe and S. Aoyagi. A feasibility decision algorithm for rate monotonic scheduling of periodic real-time tasks. In *Proceedings Real-Time Technology and Applications Symposium*, pages 212–218, 1995.

[19] Enrico Bini and Giorgio Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, 53(11):1462–1473, 2004.

[20] Jane W. S. Liu. *Real-Time Systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458, 2000.

[21] M. Garey and D. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman and company, NY, 1979.

[22] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[23] Pontus Ekberg and Wang Yi. Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard. In *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, pages 139–146. IEEE Computer Society, 2017.

[24] M. Sjodin and H. Hansson. Improved response-time analysis calculations. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*, pages 399–408, December 1998.

[25] H. Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, November 1983.

[26] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[27] A. Frank and E. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[28] Friedrich Eisenbrand. Fast integer programming in fixed dimension. In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, volume 2832 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2003.

[29] 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.

[30] Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and non-preemptive real-time uniprocessor scheduling. Technical Report RR-2966, INRIA: Institut National de Recherche en Informatique et en Automatique, 1996.

[31] Marco Cesati. Compendium of parameterized problems. http://cesati.sprg.uniroma2.it/research/compendium/compendium.pdf, 2006.

[32] R. Impagliazzo and R. Paturi. Complexity of k-SAT. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity*, pages 237–240, 1999.

[33] 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.

[34] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

[35] Tim Roughgarden, editor. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2021.

[36] Robert I. Davis, Attila Zabos, and Alan Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Transactions on Computers*, 57(9):1261–1276, 2008.