On the Complexity of Co-Static Scheduling

K. Subramani
Department of Computer Science and Electrical Engineering,
West Virginia University,
Morgantown, WV
ksmani@csee.wvu.edu

Abstract

Scheduling problems in real-time systems are often required to confront the issue of parameter variability, in particular execution time variability. Secondly, there exist complex timing constraints between jobs in such systems. A scheduling strategy must guarantee the meeting of constraints in the presence of variability. The algorithmic complexity of a real-time scheduling problem is determined by the *schedulability query*, i.e. the formal specification of what it means for a constraint system to be schedulable. In this paper, we focus on analyzing the complexity of the *Co-Static schedulability query*; we show that the problem is `coNP-complete` even for simple domains, when the constraints are arbitrary.

1 Introduction

Scheduling problems in Real-Time Systems confront two major issues, which are not addressed in traditional scheduling models viz. (a) Execution time variability, and (b) The presence of complex timing constraints between jobs that constitute the system [Pin95, SA00b]. In Static Scheduling, the dispatch vector is computed once and used in every scheduling window; online computing during the dispatching phase is not permitted. Consequently, we cannot exploit any knowledge that we may possess about the actual execution times of the jobs in the job-set. There exist periodic real-time systems, especially in Computer Controlled manufacturing, which confront the issue of parameter variability, in a manner that is fundamentally different from the Statically Schedulable systems studied in [SSRB98, Cho97]. In these systems called *Co-Static Systems*, the execution time of every job in the job-set for a particular period is known at the start of the period; however these times may be different in different periods. Co-Static systems deliver additional flexibility in the scheduling and dispatching of job-sets; but providing schedulability guarantees is provably hard.

In this paper, we shall be concerned with the following problems:

(a) Determining the co-static schedulability of a job set in a periodic real-time system (defined in Section §2),

(b) Determining the dispatch vector of the job set in a scheduling window, subject to the availability of execution time vectors, at the start of the window.

The rest of this paper is organized as follows: Section §2 describes the co-static scheduling problem and contrasts it with the static scheduling problem studied in real-time literature. The motivation for studying co-static scheduling is discussed in Section §3 while Section §4 details related approaches and perspectives. Section §5 analyzes the complexity of the co-static schedulability query and we show that the query is strongly `coNP-complete` for arbitrary constraint matrices. Polynomial time algorithms to determine co-static schedulability in certain restricted constraint classes are discussed in Section §6. Efficient online dispatching schemes are discussed in Section §7. We conclude in Section §8 by summarizing our contributions and mentioning problems for future research.

2 Statement of Problem

2.1 Job Model

Assume an infinite time-axis divided into windows of length L, starting at time $t = 0$. These windows are called *periods* or *scheduling windows*. There is a set of non-preemptive, ordered jobs, $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ that execute in each scheduling window.

2.2 Constraint Model

The constraints on the jobs are described by System (1):

$$\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}, \quad \vec{\mathbf{e}} \in \mathbf{E}, \tag{1}$$

where,

- $\mathbf{A}$ is an $m \times 2.n$ rational matrix,
- $\mathbf{E}$ is an arbitrary convex set; in this paper we confine ourselves to axis-parallel hyper-rectangles (`aph`) which can be represented as the product of n closed intervals $[l_i, u_i]$ (Also see System (15));
- $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$ is the start time vector of the jobs, and
- $\vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E}$ is the execution time vector of the jobs

2.3 Query Model

We assume that an oracle provides the execution time of every job in the job-set, at the start of each scheduling window.

Definition 2.1 *Co-static schedule - A schedule for a set of jobs, in which the start time of a job in a scheduling window can depend upon the execution times of all the jobs in that scheduling window.*

There are two orthogonal goals in Co-Static Scheduling:

1. Determining whether the input system of constraints i.e. $\mathbf{A}.[\vec{\mathbf{x}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}$ is valid for all execution time vectors $\vec{\mathbf{e}} \in \mathbf{E}$. `This corresponds to deciding the co-static schedulability query.` Deciding the query is the function of the offline schedulability analyzer.
2. Computing the start-time dispatch vector in each scheduling window, assuming that
   (a) The co-static schedulability query has been decided affirmatively, and
   (b) The execution time vector for that window is provided.

   This corresponds to the online dispatching phase.

In the co-static scheduling problem, we are interested in checking whether an input constraint system has a *co-static schedule*, i.e. a schedule in which the start times of jobs in a scheduling window can depend upon the execution time vector in that window.

Based on the discussion above, we are in a position to formally state the co-static schedulability query:

$$\forall \vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E} \quad \exists \vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n] \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}} \quad ? \tag{2}$$

The combination of the Job Model, Constraint Model and the Query Model together constitute the specification of a scheduling problem in the `E-T-C` scheduling framework [Sub00].

3 Motivation

In [SA00b, GPS95, Cho97], it has been noted that one of the principal disadvantages of static scheduling is its lack of flexibility [1]. We discuss below the phenomenon known as *Loss of Schedulability*, which is a direct consequence of this inflexibility.

Example (1): Consider a two task system $J = \{J_1, J_2\}$ *with start times* $\vec{\mathbf{s}} = [s_1, s_2]$*, execution times* $\vec{\mathbf{e}} = [e_1, e_2]$ *and the following set of constraints imposed on job execution:*

- *Task* J_1 *must finish before task* J_2 *commences; i.e.* $s_1 + e_1 \leq s_2$*;*
- *Task* J_2 *must commence within* 1 *unit of* J_1 *finishing; i.e.* $s_2 \leq s_1 + e_1 + 1$*;*

The execution time vectors $\vec{\mathbf{e}}$ *belong to the* aph $\tau = (e_1 \in)[2, 4] \times (e_2 \in)[4, 5]$.

Expressing the constraints in matrix form, we get:

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{3}$$

A static approach forces the following two constraints:

- $s_1 + 4 \leq s_2$,
- $s_2 \leq s_1 + 2 + 1 \Rightarrow s_2 \leq s_1 + 3$

which can be represented as the following linear program

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \leq \begin{bmatrix} -4 \\ 3 \end{bmatrix} \tag{4}$$

Clearly the linear program specified by System (4) is infeasible and a static schedule does not exist.

We now proceed to find a vector $\vec{\mathbf{e}'} \in \tau$, such that the polytope in $\vec{\mathbf{s}}$ space, resulting from substituting $\vec{\mathbf{e}'}$ in System (3) is empty. Such a vector is termed as a *co-static break vector*, since it causes the input constraint system to break, i.e. get violated. Consider the following four assignments to the vector $\vec{\mathbf{e}} = [e_1, e_2]$:

$$\vec{\mathbf{e}} = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \begin{bmatrix} 2 \\ 5 \end{bmatrix}, \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \begin{bmatrix} 4 \\ 5 \end{bmatrix} \tag{5}$$

Each of these assignments, when substituted in System (3), results in a feasible system. In fact, using the Linear Programming system in [Ber95], we get

$$\vec{\mathbf{s}} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ 4 \end{bmatrix}, \begin{bmatrix} 0 \\ 4 \end{bmatrix} \tag{6}$$

as the start times corresponding to the execution times in Equation (5).

Observe that any execution vector $\vec{\mathbf{e}'} \in \tau$ *can be represented as a convex combination of the four vectors in Equation (5).* Consequently, it suffices to consider only these extreme vectors in determining the existence of a break-vector. (This idea will be discussed further in Section §5.) Thus there is no break vector in τ. Likewise, for each vector $\vec{\mathbf{e}'} \in \tau$, there exists a (*possibly unique*) vector $\vec{\mathbf{s}}$ which satisfies the input constraint system. This phenomenon, in which a static scheduling approach declares a system infeasible in the absence of a break-vector in the execution time domain, is termed as *loss of schedulability.*

For practical applications, refer Appendix

[1] Static Scheduling is concerned with associating rational numbers to the start times in each scheduling window.

4 Related Work

In this section, we contrast the co-static scheduling problem with the *imprecise computation model* proposed in [TK91] and the *error-task model* discussed in [cFL97].

[Leu91] discusses some of the earliest formalizations in imprecise computation models. In these models, there is a trade-off between meeting deadline requirements and the quality of results that are output. Time-Critical tasks are permitted to degrade output, as long as temporal constraints are met [LLS+91]. These models are useful in applications such as image processing, which require the production of at least a fuzzy frame in time. A sharp image produced after the imposed deadline may have little or no value [LLN87, CLK90]. Our model does not permit the flexibility of the deadline-quality trade-off, but we consider more general constraints on jobs.

The error-task model in [cFL97] describes scheduling algorithms for preemptive, imprecise, composite real-time tasks. In this model, input error is explicitly accounted for in the design of solution strategies. Composite tasks consist of a chain of component tasks and each component task is composed of a mandatory part and an optional part. The key idea is to model imprecise inputs through an increase in the processing time for the mandatory and optional parts. Their strategy is similar to ours, in that they model error-rates through processing times, whereas we account for resource variability through execution times. Related modeling approaches have been suggested in [RCGF97], in which both "soft" and "hard" preemptive tasks are considered.

In both [Cho97] and [GPS95], the Loss of Schedulability phenomenon, discussed in Section §3 is mentioned. However, they use the phenomenon to motivate the necessity for *parametric scheduling* in periodic job-sets, with and without inter-period constraints, respectively. Parametric scheduling ([GPS95, SA00a]) is applicable when we have access to the execution times of jobs that have already executed, as opposed to the complete execution time vector that is required by co-static scheduling. Co-static scheduling algorithms exploit the knowledge of the execution time vector to compute dispatch vectors online.

Our work represents the first attempt to study the co-static scheduling problem from a computational complexity perspective; our investigations reveal that deciding the co-static schedulability query is strongly `coNP-complete` (Section §5), although online dispatching can be effected in polynomial time (Section §7).

5 Complexity of Co-Static Schedulability

The complexity analysis of the co-static scheduling problem is divided into two parts. This section analyzes the complexity of deciding the co-static schedulability query, while Section §7 is concerned with the efficiency of dispatching schemes.

Algorithm (5.1) presents a simple technique to decide the co-static schedulability query. It proceeds by eliminating one start time variable at a time, till all the start time variables are eliminated. At this point, the constraint system (1) becomes a polyhedral system in the space of the execution time variables say $\mathbf{\Psi} = \mathbf{K}.\vec{\mathbf{e}} \leq \vec{\mathbf{k}}$. Consequently, query (2) is true if and only if $\mathbf{\Psi} \supseteq \mathbf{E}$.

The Fourier-Motzkin elimination technique discussed in [DE73] represents one implementation of ELIM-EXIST-VARIABLE. In general, any polyhedral projection method suffices.

Algorithm (5.2) checks if the convex domain $\mathbf{E}$ is completely contained within the polyhedral system $\mathbf{K}.\vec{\mathbf{e}} \leq \vec{\mathbf{k}}$. A linear constraint of the form $\mathbf{K_i}.\vec{\mathbf{e}} \leq \mathbf{k_i}$ is a half-hyperplane in the space of the execution time variables. A convex body $\mathbf{E}$ is said to lie on the *valid* side of $\mathbf{K_i}.\vec{\mathbf{e}} \leq \mathbf{k_i}$ if and only if for all points $\vec{\mathbf{e}} \in \mathbf{E}$, we have $\mathbf{K_i}.\vec{\mathbf{y}} \leq \mathbf{k_i}$. When there is no confusion, we say that $\mathbf{K_i}.\vec{\mathbf{e}} \leq \mathbf{k_i}$ is valid for $\mathbf{E}$.

Lemma: 5.1 *A convex body* $\mathbf{E}$ *lies on the valid side of* $\mathbf{K_i}.\vec{\mathbf{e}} \leq \mathbf{k_i}$ *, if and only if*

$$\max_{\vec{\mathbf{e}} \in \mathbf{E}} \mathbf{K_i}.\vec{\mathbf{e}} \leq \mathbf{k_i}$$

Proof: *This is obvious as indicated by Figure (1). We also note that the validity of* $\mathbf{K_i}.\vec{\mathbf{e}} \leq \mathbf{k_i}$ *for* $\mathbf{E}$ *can be checked in* $O(\mathcal{C})$ *time, where* $\mathcal{C}$ *is the running time of the fastest convex minimization algorithm.* □

We can conclude that

Lemma: 5.2 *A convex body* $\mathbf{E}$ *is contained within a polyhedral system* $\mathbf{K}.\vec{\mathbf{e}} \leq \vec{\mathbf{k}}$*, if and only if it lies on the valid side of each constraint hyper-plane of the polyhedral system.*

Function CO-STATIC-SCHEDULER $(\mathbf{E}, \mathbf{A}, \vec{\mathbf{b}})$

1: { Recall that the co-static scheduling query is:

$$\forall \vec{\mathbf{e}} \in \mathbf{E} \quad \exists s_1, s_2, \ldots s_n \quad \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}} \quad ?$$

}
2: **for** ($i = n$ **down to** 1) **do**
3: ELIM-EXIST-VARIABLE(s_i)
4: **end for**
5: { The system $\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}$ is transformed into a polyhedral system $\mathbf{\Psi} = \mathbf{K}.\vec{\mathbf{e}} \leq \vec{\mathbf{k}}$ in the execution time variable space. Let m_1 denote the number of constraints in this polyhedral system.}
6: **if** (POLYTOPE-INCLUDE$(\mathbf{K}, \vec{\mathbf{k}}, \mathbf{E})$) **then**
7: `System is Co-Statically schedulable`
8: **return**
9: **else**
10: `No Co-Static Schedule Exists`
11: **return**
12: **end if**

Algorithm 5.1: A Quantifier Elimination Algorithm for determining Co-Static Schedulability

Function POLYTOPE-INCLUDE $(\mathbf{K}, \vec{\mathbf{k}}, \mathbf{E})$

1: { Assume that $\mathbf{K}.\vec{\mathbf{e}} \leq \vec{\mathbf{k}}$ has m_1 constraints. Let the i^{th} constraint be represented as: $\mathbf{K_i}.\vec{\mathbf{e}} \leq \mathbf{k_i}$ }
2: **for** ($i = 1$ **to** m_1) **do**
3: **if** ($\max_{\mathbf{E}} \mathbf{K_i}.\vec{\mathbf{e}} > \mathbf{k_i}$) **then**
4: **return**(**false**)
5: **end if**
6: **end for**
7: **return**(**true**)

Algorithm 5.2: Polytope Inclusion

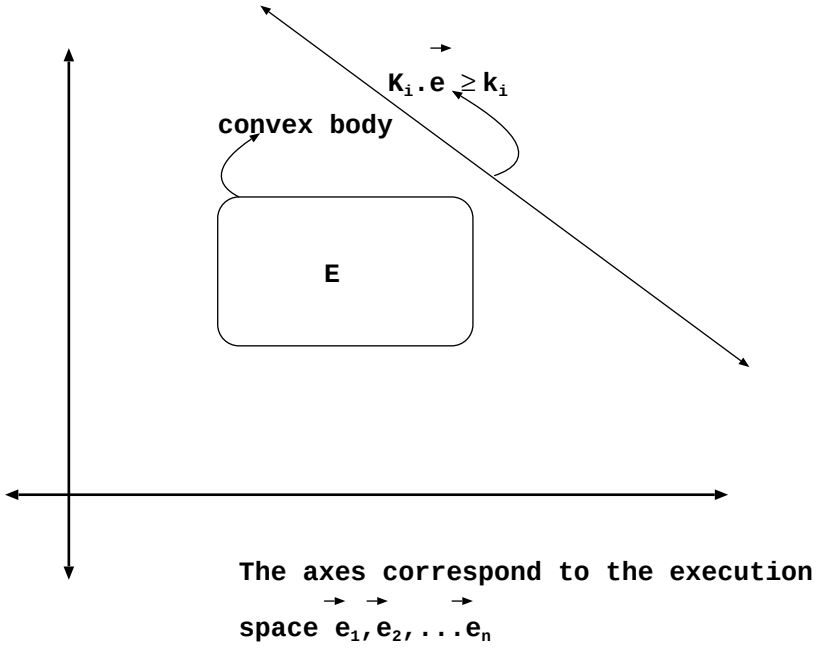

Figure 1: Polytope inclusion

Lemma: 5.3 *The query:* Does a convex body $\mathbf{E}$ lie within a polyhedron $\mathbf{K}.\vec{\mathbf{e}} \leq \vec{\mathbf{k}}$ *can be decided in time polynomial in the size of* $\mathbf{K}, \vec{\mathbf{k}}$ *and* $\mathbf{E}$.

Proof: *As indicated by Lemma (5.2), the decision procedure has to execute m_1 convex minimization calls, where m_1 is number of constraints in the System* $\mathbf{K}.\vec{\mathbf{e}} \leq \vec{\mathbf{k}}$. □

Although POLYTOPE-INCLUDE() runs in time polynomial in the size of its input, the output of elimination procedures such as the Fourier-Motzkin elimination Algorithm are exponentially large when the constraints are arbitrary, thereby rendering Algorithm (5.1) impractical [CR99].

In the discussion that follows, we establish that the computational complexity of deciding co-static schedulability is a characteristic of the problem itself and not of the particular projection procedure employed.

We commence our analysis of the co-static scheduling query (2) by studying its complement. *Note that a query is true if and only if its complement is false.*

The complement of query (2) is:

$$\neg(\ \ \forall \vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E} \ \ \exists \vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n] \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}} \ \ ) \quad ? \tag{7}$$

Moving the negation inside the parentheses of query (7), we obtain

$$\exists \vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E} \ \ \forall \vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n] \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \not\leq \vec{\mathbf{b}} \quad ? \tag{8}$$

We use $\mathbf{A}.\vec{\mathbf{x}} \not\leq \vec{\mathbf{b}}$ *to indicate that the polyhedral set* $\{\vec{\mathbf{x}} : \mathbf{A}.\vec{\mathbf{x}} \leq \vec{\mathbf{b}}\}$ *is empty.*

Query (8) basically asks whether there exists an execution time vector $\vec{\mathbf{e}} \in \mathbf{E}$ such that the linear system resulting from substituting these execution times in the constraint system $\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}$ is infeasible, i.e. whether the polyhedral set $\{\vec{\mathbf{s}}: A.[\vec{\mathbf{s}}.\vec{\mathbf{e}}] \leq \vec{\mathbf{b}} | \vec{\mathbf{e}} = [e'_1, e'_2, \ldots, e'_n]\}$ is empty. As mentioned in Section §3, such a vector is called a co-static break vector. Thus, the existence of a break vector would imply the non-existence of a co-static schedule. On the other hand, proof of non-existence of a break vector implies that query (2) can be answered in the affirmative, thereby implying the existence of a co-static schedule.

We rewrite the constraint system $\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}$ in the form

$$\mathbf{G}.\vec{\mathbf{s}} + \mathbf{H}.\vec{\mathbf{e}} \leq \vec{\mathbf{b}} \tag{9}$$

Accordingly, Equation (8) gives

$$\exists \vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E} \quad \forall \vec{\mathbf{s}} = [s_1, s_2, \ldots s_n] \quad \mathbf{G}.\vec{\mathbf{s}} \not\leq \vec{\mathbf{b}} - \mathbf{H}.\vec{\mathbf{e}} \quad ? \tag{10}$$

($\vec{\mathbf{b}} - \mathbf{H}.\vec{\mathbf{e}}$) is an $m-$vector, with each element being an affine function in the e_i variables. We set $\vec{\mathbf{g}} = \vec{\mathbf{b}} - \mathbf{H}.\vec{\mathbf{e}}$, so that we can rewrite query (7) as

$$\exists \vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E} \quad \forall \vec{\mathbf{s}} = [s_1, s_2 \ldots \ s_n] \quad \mathbf{G}.\vec{\mathbf{s}} \not\leq \vec{\mathbf{g}} \quad ? \tag{11}$$

The analysis of query (11) is complicated by the existence of the universally quantified variable $\vec{\mathbf{s}}$. Our efforts would be significantly simplified if we could replace it with an *equivalent existentially quantified variable.* The following lemma called Farkas' lemma helps us achieve this goal.

Lemma: 5.4 *[Farkas] Either* $\{\vec{\mathbf{x}} \in \Re_+^n : \mathbf{A}.\vec{\mathbf{x}} \leq \vec{\mathbf{b}}\} \neq \phi$ *or (exclusively)* $\exists \vec{\mathbf{y}} \in \Re_+^m$*, such that,* $\mathbf{A^T}.\vec{\mathbf{y}} \geq \vec{\mathbf{0}}^T$ *and* $\vec{\mathbf{y}}^{\mathbf{T}}.\vec{\mathbf{b}} = -\infty$.

Proof: *See [Sch87, PS82, NW88]. Figure (2) provides a graphical illustration of the lemma.* □

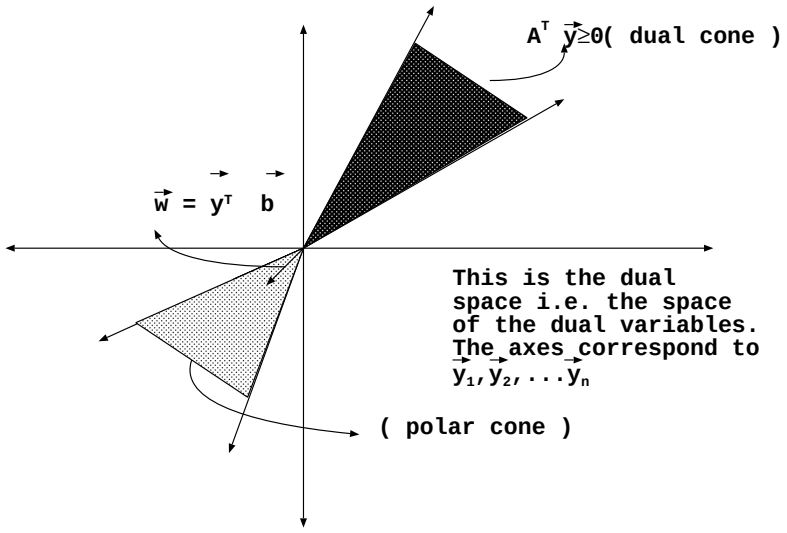

Figure 2: Farkas' lemma

Lemma (5.4) has the following interpretation: If the primal system is infeasible, then there exists a "short" proof of this infeasibility (See [NW88]). This proof takes the form of a witness vector which is unbounded below in the dual space. In Figure (2), $\vec{\mathbf{w}} = \vec{\mathbf{y}^{\mathbf{T}}}.\vec{\mathbf{b}}$ represents the witness vector and the shaded region is the dual cone corresponding to the linear system $\mathbf{A}.\vec{\mathbf{x}} \leq \vec{\mathbf{b}}$. According to Farkas' lemma, if the linear system is infeasible, then $\vec{\mathbf{w}}$ will lie in the polar cone corresponding to the dual cone.

Applying the lemma to our problem, we note that query (11) asks whether the system $\mathbf{G}.\vec{\mathbf{s}} \leq \vec{\mathbf{g}}$ is infeasible for a particular $\vec{\mathbf{e}} \in \mathbf{E}$. Farkas' lemma assures us that this is possible if and only if

$$\exists \vec{\mathbf{e}} \in \mathbf{E} \quad \exists \vec{\mathbf{y}} \in \Re_+^m \quad \vec{\mathbf{y}}^T.\mathbf{G} \geq \vec{\mathbf{0}}, \quad \vec{\mathbf{y}}^{\mathbf{T}}.\vec{\mathbf{g}} = -\infty \quad ? \tag{12}$$

which implies that

$$\exists \vec{\mathbf{e}} \in \mathbf{E} \quad \exists \vec{\mathbf{y}} \in \Re_+^m \quad \mathbf{G}^{\mathbf{T}}.\vec{\mathbf{y}} \geq \vec{\mathbf{0}}, \quad \vec{\mathbf{y}}^{\mathbf{T}}.(\vec{\mathbf{b}} - \mathbf{H}.\vec{\mathbf{e}}) = -\infty \quad ? \tag{13}$$

Query (13) has the following algorithmic interpretation:

Let z be the minimum of the bilinear form $\vec{\mathbf{y}}^{\mathbf{T}}.(\vec{\mathbf{b}} - \mathbf{H}.\vec{\mathbf{e}})$ over the two convex bodies : $\{\vec{\mathbf{y}} : \vec{\mathbf{y}} \geq \vec{\mathbf{0}}, \mathbf{G}^{\mathbf{T}}.\vec{\mathbf{y}} \geq \vec{\mathbf{0}}\}$ and $\mathbf{E}$. If $z = -\infty$, the input system of constraints does not have a co-static schedule.

We make the following observations:

1. From an algorithmic perspective, we need only check if $z < 0$; in the dual cone, i.e. $\mathbf{G^T}.\vec{\mathbf{y}} \geq \vec{\mathbf{0}}$, $z < 0 \Rightarrow z = -\infty$ [Sch87].

2. The bilinear form $z = \vec{\mathbf{y}}^{\mathbf{T}}.(\vec{\mathbf{b}} - \mathbf{H}.\vec{\mathbf{e}})$ can be expressed as the following quadratic form:

$$z = [y_1, y_2, \ldots, y_m, e_1, e_2, \ldots, e_n]\mathcal{F}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ e_n \end{bmatrix}, \tag{14}$$

   for suitably chosen F.

3. The bilinear form has extremal properties, i.e. its minimum is achieved at extreme points of the two sets $\{\vec{\mathbf{y}} : \vec{\mathbf{y}} \geq \vec{\mathbf{0}}, \mathbf{G^T}.\vec{\mathbf{y}} \geq \vec{\mathbf{0}}\}$ and $\mathbf{E}$ [BSS93].

The dual form of the complement of the co-static scheduling query, i.e. query (13), will henceforth be called the *co-scheduling query.*

For the rest of this paper, we assume that the set $\mathbf{E}$ in query (2) is an axis-parallel hyper-rectangle (`aph`) $\mathbf{\Upsilon} = [l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_n, u_n]$ represented by:

$$\mathbf{\Upsilon} = \{\vec{\mathbf{e}} : \mathbf{N}.\vec{\mathbf{e}} \leq \vec{\mathbf{f}}\}. \tag{15}$$

Accordingly, query (13) becomes

$$\exists \vec{\mathbf{e}} \in \mathbf{\Upsilon} \quad \vec{\mathbf{y}} \in \Re_+^m \quad \mathbf{G^T}.\vec{\mathbf{y}} \geq \vec{\mathbf{0}}, \quad \vec{\mathbf{y}}^{\mathbf{T}}.(\vec{\mathbf{b}} - \mathbf{H}.\vec{\mathbf{e}}) = -\infty \quad ? \tag{16}$$

In this case, if a break vector exists, then there must exist a break vector $\vec{\mathbf{e}}$ with elements $e_i \in \{l_i, u_i\}, \forall i = 1, \ldots, n$, i.e. only the end-points of the execution time ranges matter. Example (4) in Section §3 utilized precisely this property. The extremal point property of the bilinear form also suggests the following strategy to decide query (16): Substitute the 2^n vertices of the `aph` represented by System (15) in System (16) and check for the unboundedness of the cost function in the dual space. If the cost function is unbounded for any $\vec{\mathbf{e}} \in \mathbf{\Upsilon}$, we have obtained the desired break vector. On the other hand, if z has minimum value 0 for every $\vec{\mathbf{e}} \in \mathbf{\Upsilon}$, then the constraint system has a co-static schedule. We proceed to show that we cannot hope to substantially improve this naive strategy, at least from a deterministic perspective.

Lemma: 5.5 *Query (16) lies in the class* `NP`.

Proof: *From the discussion above, we know that optimality occurs at an extreme point of the polyhedron* $\mathbf{\Upsilon} = \{\mathbf{N}.\vec{\mathbf{e}} \leq \vec{\mathbf{f}}\}$. *Hence an oracle need only guess one of these points and check that the corresponding linear function in* $\vec{\mathbf{y}}$ *is unbounded below in* $\mathbf{G^T}.\vec{\mathbf{y}} \geq \vec{\mathbf{0}}$. *Since all the extreme point components belong to* $\{l_i, u_i\}, \forall i = 1, \ldots, n$, *the lemma follows. We observe that the* $\{l_i, u_i\}$ *are provided as part of the input and hence the guess of the oracle is of length polynomial in the size of the input.* □

Lemma: 5.6 *Query (16) is* `NP-hard`.

We state below some results from Non-linear Programming that we shall use in our proof of Lemma (5.6),

Result: 5.1 *The bilinear program:*

$$\begin{aligned} \min z &= \vec{\mathbf{u}}.\mathbf{Q}.\vec{\mathbf{v}} \\ \texttt{subject to} \quad \mathbf{S}.\vec{\mathbf{u}} &\leq \vec{\mathbf{b}}, \mathbf{T}.\vec{\mathbf{v}} \leq \vec{\mathbf{d}}, \end{aligned} \tag{17}$$

where all vectors and matrices have the appropriate dimensions, is `NP-hard`.

Proof: *From [HT90], we know that Result (5.1) is true, even when both* $\mathbf{S}\vec{\mathbf{u}} \leq \vec{\mathbf{b}}$ *and* $\mathbf{T}.\vec{\mathbf{v}} \leq \vec{\mathbf{d}}$ *are axis-parallel hyper-rectangles.* □

Result: 5.2 *The decision version of the bilinear program, i.e*

$$\exists \vec{\mathbf{u}}, \vec{\mathbf{v}} \geq \vec{\mathbf{0}}, \quad \vec{\mathbf{u}}.\mathbf{Q}, \vec{\mathbf{v}} \leq k? \tag{18}$$
$$\texttt{subject to} \quad \mathbf{S}.\vec{\mathbf{u}} \leq \vec{\mathbf{b}}, \mathbf{T}.\vec{\mathbf{v}} \leq \vec{\mathbf{d}},$$

is polynomially equivalent to the optimization version represented by System (17).

Proof: *It is clear that*

$$\texttt{System (18)} \leq_p \texttt{System (17)},$$

where $\leq_p$ *indicates polynomial-time reducibility. Let* M *denote the minimum of System (17). Assume that there exists an oracle to decide System (18). At most,* $O(\log M)$ *calls to this oracle are required to solve System (17). Thus, we have*

$$\texttt{System (17)} \leq_p \texttt{System (18)}$$

where the reduction is a "Cook" or "Turing" reduction [GJ79]. □

We are now ready to prove Lemma (5.6).

Proof: *Any arbitrary bilinear program, in the form of System (17), in which the polyhedron representing* $\mathbf{T}.\vec{\mathbf{v}} \leq \vec{\mathbf{d}}$ *is an axis-parallel hyper-rectangle can be decided by query (16).* □

Corollary 5.1 *Query (16) is* `NP-complete`.

Proof: *Follows from Lemma (5.5) and Lemma (5.6).* □

Corollary 5.2 *Co-static Scheduling with arbitrary constraints over an axis-parellel hyper-rectangle is strongly* `coNP-complete`.

Proof: *Follows from the strong* `NP-completeness` *of Bilinear Programming.* □

6 Special Case Analysis

In this section, we enumerate the cases in which it is possible to determine co-static schedulability in polynomial time.

1. The matrix $\mathcal{F}$ in query (14) is positive semidefinite - In this case, we can use the ellipsoid algorithm [HuL93, Kha79], which is guaranteed to run in polynomial time. If the global minimum of query (16) is less than 0, we know that the constraint system does not have a co-static schedule.
2. The execution time variables, i.e. the e_i, occur with the same sign in every constraint - We use Algorithm (6.1) to eliminate the execution time variables from query (16).

Function EXECUTION-TIME-ELIMINATION $(\mathbf{\Upsilon}, \mathbf{A}, \vec{\mathbf{b}})$
1: **for** (i=1 **to** n) **do**
2: **if** (e_i can be put in the form $e_i \leq f'()$, in every constraint that it occurs) **then**
3: Substitute $e_i = u_i$.
4: **else**
5: Substitute $e_i = l_i$.
6: **end if**
7: **end for**

Algorithm 6.1: Special case for co-static schedulability

The correctness of the elimination procedure follows from the correctness of the universal variable elimination procedure discussed in [CH99]. We note that in this case, *co-static schedulability reduces to static schedulability* and the need for online computation is obviated.

7 Complexity of Online Dispatching

The discussion in Section §5 established that deciding co-static schedulability for an arbitrary constraint set is `coNP-complete`. But determining schedulability is only part of the co-static scheduling problem. Suppose that for a given instance of the problem, we are able to use hill-climbing techniques, such as simulated annealing, to successfully conclude that there is a co-static schedule. We are still faced with the problem of computing online dispatch schedules in each scheduling window. This section is devoted to addressing the dispatching problem, assuming that the co-static scheduling query (2) has been answered affirmatively.

Result: 7.1 *The online dispatch schedule for an instance of co-static scheduling when the domain is an* `aph` *can be computed in $O(\mathcal{L})$ time, where $\mathcal{L}$ is the running time of the fastest Linear Programming algorithm.*

Proof: *Given the execution time vector $\vec{\mathbf{e'}} = [e_1, e_2, \ldots, e_n]$, for the current scheduling window, we can substitute it in System (9) to get a linear system of the form*

$$\mathbf{G}.\vec{\mathbf{s}} \leq \vec{\mathbf{d}}', \vec{\mathbf{s}} \geq \vec{\mathbf{0}},$$

which can be solved through the application of a Linear Programming algorithm, such as [Vai87], to provide the dispatch schedule for the current scheduling window. □

Result: 7.2 *The online dispatch schedule for an instance of co-static scheduling when the constraints are standard can be computed in $O(m.n)$ time.*

Proof: *When the constraints are standard, the constraint system can be represented as a network graph, as discussed in [Sub00]. Determining the dispatch schedule is equivalent to finding the single-source shortest paths in this graph [CLR92]. An algorithm, such as the Bellman-Ford algorithm, computes the single-source shortest paths in a network graph, in time $O(m.n)$, thereby giving the schedule.* □

8 Summary

In this paper, we studied the computational complexity of the co-static schedulability query. Previous research had considered similar problems and even the same problem from a different perspective; however our work represents the first attempt at characterizing the complexity of this problem.

We showed that the co-static schedulability query is `coNP-complete` for arbitrary constraint sets, even when the execution time domain is an axis-parallel hyper-rectangle. However, once the query is decided, the problem of dynamic dispatching is relatively easy. A problem that is currently open is determining the complexity of co-static scheduling over standard constraints.

A A Job-Shop application

We discuss a manufacturing automation application that demonstrates the applicability of the co-static scheduling query.

Figure (3) represents the working of a typical manufacturing shop [Kam86]. An untooled metal object called *workpiece* is operated upon sequentially by a number of machines (usually lathes) to deliver a *finished* (tooled) object which meets the required specifications. The sequential flow of the workpiece through the machines represents an *assembly line*. Each machine is part of multiple assembly lines and works on one or more workpieces in parallel. Associated with each machine are a number of tools; these tools are apportioned among the various workpieces that it (the machine) is currently operating on. The time taken by the machine to operate on a workpiece (called the *latency* of the machine for the workpiece) varies inversely as the number of tools it allocates to the workpiece. At any point in time, the resource constraints on the machine, i.e. the number of tools available for allocation to the workpiece, are known. However, this number could change over time, affecting its latency for the workpiece.

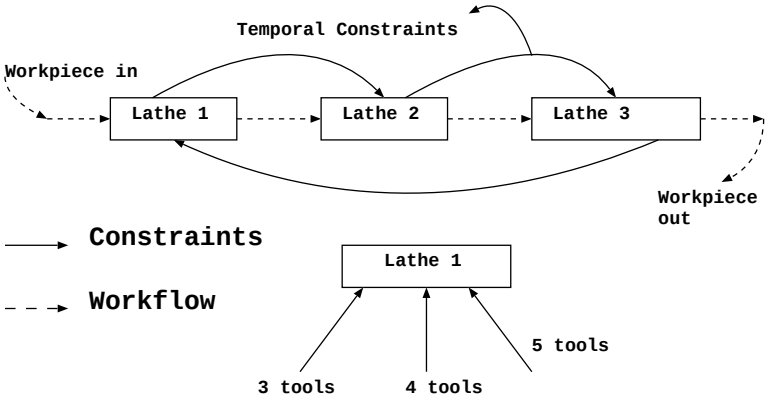

Figure 3: A Computer Controlled machine operating on a work-piece

The temporal constraints between machines represent the timing relationships on their operation; e.g. consider the specification that Lathe 2 must wait at least 10 minutes, after the workpiece has exited Lathe 1, due to cooling requirements. Since there are definite upper and lower bounds on the number of tools that a machine can allocate to a workpiece, it follows that there are definite lower and upper bounds on the latency of the machine for that workpiece.

Finally, there is a deadline of D time units for each workpiece; i.e. at the end of D time units, the workpiece should exit the last machine in the assembly line (Lathe 3 in Figure (3)), at which point, the next workpiece will enter Lathe 1 and the cycle repeats. The deadline requirement permits us to regard the system as a periodic real-time system with a period of D time units. Let Ξ denote the function that computes the latency times of the machines for the workpiece at the start of the period, given the resource allocation at that time. Ξ *is the oracle alluded to in Section §2* . The job-shop scheduler is then faced with the following problems:

1. Given upper and lower bounds on the latency of each machine for a workpiece and the temporal constraint set, is there a schedule for all possible latency values ?
2. Given that such a schedule exists, what is the schedule for the current window ? (It is assumed that the controller has access to Ξ.)

Clearly, the requirements of the job-shop scheduling problem can be modeled through the `E-T-C` model, augmented by the co-static scheduling query (2).

Additional application areas include avionics (flight control) [SA00c], machining [Y.K80, Kor83, SE87, SK90], real-time databases [BFW97] and real-time operating systems [LTCA89, DRSK89] and internet scheduling.

References

[Ber95] M. Berkelaar. Linear programming solver. *Software Library for Operations Research, University of Karlsruhe*, 1995.

[BFW97] Azer Bestavros and Victor Fay-Wolfe, editors. *Real-Time Database and Information Systems, Research Advances*. Kluwer Academic Publishers, 1997.

[BSS93] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley, New York, second edition, 1993.

[cFL97] Wu chun Feng and Jane W.-S. Liu. Algorithms for scheduling real-time tasks with input error and end-to-end deadlines. *IEEE Transactions on Software Engineering*, 23(2):93–105, February 1997.

[CH99] V. Chandru and J. N. Hooker. *Optimization Methods for Logical Inference*. Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., 1999.

[Cho97] Seonho Choi. *Dynamic Time-based scheduling for Hard Real-Time Systems.* PhD thesis, University of Maryland, College Park, jun 1997.

[CLK90] J.Y. Chung, J.W.S. Liu, and K.J.Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Transactions on Computers*, 19(9):1156–1173, September 1990.

[CLR92] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press and McGraw-Hill Book Company, 6th edition, 1992.

[CR99] Chandru and Rao. Linear programming. In *Algorithms and Theory of Computation Handbook, CRC Press, 1999.* CRC Press, 1999.

[DE73] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin Elimination and its Dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

[DRSK89] A. Damm, J. Reisinger, W. Schwabl, and H. Kopetz. The Real-Time Operating System of MARS. *ACM Special Interest Group on Operating Systems*, 23(3):141–157, July 1989.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness.* W. H. Freeman Company, San Francisco, 1979.

[GPS95] R. Gerber, W. Pugh, and M. Saksena. Parametric Dispatching of Hard Real-Time Tasks. *IEEE Transactions on Computers*, 1995.

[HN94] Dorit S. Hochbaum and Joseph (Seffi) Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, December 1994.

[HT90] R. Horst and H. Tuy. *Global Optimization.* Springer, Berlin, 1990.

[HuL93] J. B. Hiriart-urruty and C. Lemarechal. *Convex Analysis and Minimization Algorithms.* Springer-Verlag, 1993.

[Kam86] M. Kaminski. Protocols for communication in the factory. *IEEE Spectrum*, April 1986.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 224:1093–1096, 1979. English Translation: *Soviet Mathematics Doklady*, Volume 20, pp. 1093–1096.

[Kor83] Y. Koren. *Computer Control of Manufacturing Systems.* McGraw-Hill, New York, 1983.

[Leu91] Joseph Y-T. Leung. *Research in Real-Time Scheduling*, chapter 2, pages 31–62. Volume 2 of Tilborg and Koob [TK91], 1991.

[LLN87] J.W.S. Liu, K.J. Lin, and S. Natarajan. Scheduling real-time periodic jobs using imprecise results. In *RTSS*, pages 252–260, San Jose, California, 1987.

[LLS+91] J.W.S. Liu, K.J. Lin, W.K. Shih, A.C.Yu, J.Y.Chung, and W. Zhao. *Algorithms for Scheduling Imprecise Computations*, chapter 8, pages 203–249. Volume 2 of Tilborg and Koob [TK91], 1991.

[LTCA89] S. T. Levi, S. K. Tripathi, S. D. Carson, and A. K. Agrawala. The `Maruti` Hard Real-Time Operating System. *ACM Special Interest Group on Operating Systems*, 23(3):90–106, July 1989.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, New York, 1988.

[Pin95] M. Pinedo. *Scheduling: theory, algorithms, and systems.* Prentice-Hall, Englewood Cliffs, 1995.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization.* Prentice Hall, 1982.

[RCGF97] Ismael Ripoll, Alfons Crespo, and Ana Garcia-Fornes. An optimal algorithm for scheduling soft aperiodic tasks in dynamic-priority preemptive systems. *IEEE Transactions on Software Engineering*, 23(6):388–399, June 1997.

[SA00a] K. Subramani and A. K. Agrawala. A dual interpretation of standard constraints in parametric scheduling. Technical Report CS-TR-4112, University of Maryland, College Park, Department of Computer Science, March 2000. Accepted at FTRTFT 2000.

[SA00b] K. Subramani and A. K. Agrawala. The parametric polytope and its applications to a scheduling problem. Technical Report CS-TR-4116, University of Maryland, College Park, Department of Computer Science, March 2000.

[SA00c] K. Subramani and A. K. Agrawala. The static polytope and its applications to a scheduling problem. 3^{rd} *IEEE Workshop on Factory Communications*, September 2000.

[Sch87] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1987.

[SE87] K. Shin and M. Epstein. Intertask communication in an integrated multi-robot system. *IEEE Journal of Robotics and Automation*, 1987.

[SK90] K. Srinivasan and P.K. Kulkarni. Cross-coupled control of biaxial feed drive mechanisms. *ASME Journal of Dynamic Systems, Measurement and Control*, 112:225–232, 1990.

[SSRB98] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo, editors. *Deadline Scheduling for Real-Time Systems*. Kluwer Academic Publishers, 1998.

[Sub00] K. Subramani. *Duality in the Parametric Polytope and its Applications to a Scheduling Problem*. PhD thesis, University of Maryland, College Park, July 2000.

[TK91] A. Tilborg and G. Koob, editors. *Foundations of Real-Time Computing -Scheduling and Resource Management*, volume 2. Kluwer Academic Publishers, 1991.

[Vai87] P. M. Vaidya. An algorithm for linear programming which requires $O(((m+n)n^2+(m+n)^{1.5}n)L)$ arithmetic operations. In Alfred Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 29–38, New York City, NY, May 1987. ACM Press.

[Y.K80] Y.Koren. Cross-coupled biaxial computer control for manufacturing systems. *ASME Journal of Dynamic Systems, Measurement and Control*, 102:265–272, 1980.