On the complexity of Parametric Scheduling

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

Abstract

Parametric Scheduling is a scheduling technique used to partially address the inflexibility of static scheduling in real-time systems. Real-Time scheduling confronts two issues not addressed by traditional scheduling models viz. parameter variability and the existence of complex relationships constraining the execution of tasks. Accordingly, modeling becomes crucial in the specification of scheduling problems. In this paper, we study the complexity of parametric scheduling within the `E-T-C` scheduling framework introduced in [Sub00]. We establish that (a) The parametric schedulability query is in `PSPACE` for arbitrary constraint sets, and (b) Parametric dispatching is not harder than than deciding the schedulability query. To the best of our knowledge, our results are the first of their kind for this problem. [1]

1 Introduction

An important feature in Real-time systems is *parameter impreciseness*, i.e. the inability to accurately determine certain parameter values. The most common such parameter is *task execution time*. A second feature is the presence of complex relationships between tasks that constrain their execution. Traditional models do not accomodate either feature completely: (a) Variable execution times are modeled through a fixed value (*worst-case*), and (b) Relationships are limited to those that can be represented by precedence graphs. We present a task model that effectively captures *variable task execution time*, while simultaneously permitting arbitrary linear relationships between tasks. Real-Time scheduling problems can be classified as *static, co-static or parametric* depending upon the information available at dispatching [Sub00]. In this paper we focus on real-time systems in which the only information available before a job is to be dispatched is the *actual execution time of every job sequenced before it.* The primary scheduling goal is to provide an offline guarantee that the input constraints will be met at run-time. Such systems are termed as *Parametrically Specified systems* or *Parametric Systems*, since the dispatch time of a job will, in general, be a parameterized function of the start and execution times of jobs sequenced before it.

The parametric scheduling problem is concerned with the following two issues:

1. Deciding the schedulability predicate for a parametrically specified system (§2), and
2. Determining the dispatch time of a job, given the start and execution times of all jobs sequenced before it.

The rest of this chapter is organized as follows: We introduce the parametric scheduling problem in Section §2 and state the schedulability query. Section §3 motivates the necessity for the parametric schedulability specification through examples from real-time design. Previous work in the design of parametrically specified systems is detailed in Section §4. The complexity of the parametric schedulability query is analyzed in Section §5; we show that for arbitrary constraint systems the query is `PSPACE-easy`. Section §6 provides a dual interpretation of the parametric schedulability algorithm in [Sak94]. Online dispatching algorithms for arbitrarily constrained parametric systems are discussed in Section §7. Section §8 summarizes our contributions in this chapter and tabulates our results.

[1] The results presented in this chapter also appear in Chapter 5 of [Sub00].

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 $[l_i, u_i]$;
- $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$ 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

Suppose that job J_a has to be dispatched. We assume that the dispatcher has access to the start times $\{s_1, s_2, \ldots, s_{a-1}\}$ and execution times $\{e_1, e_2, \ldots, e_{a-1}\}$ of the jobs $\{J_1, J_2, \ldots, J_{a-1}\}$.

Since the actual execution times of the previously executed jobs are required, the execution time domain must perforce be an axis-parallel hyper-rectangle (`aph`). For the rest of this paper, we assume that the domain $\mathbf{E}$ in System (1) is an `aph` represented by:

$$\mathbf{\Upsilon} = [l_1, u_1] \times [l_2, u_2] \times \ldots [l_n, u_n] \tag{2}$$

Definition: 2.1 *A parametric schedule of an ordered set of jobs, in a scheduling window, is a vector* $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$*, where* s_1 *is a rational number and each* $s_i, i \neq 1$ *is a function of the start time and execution time variables of jobs sequenced prior to job* J_i*, i.e.* $\{s_1, e_1, s_2, e_2, \ldots, s_{i-1}, e_{i-1}\}$*. Further, this vector should satisfy the constraint system (1) for all execution time vectors* $\vec{\mathbf{e}} \in \mathbf{\Upsilon}$*.*

The Parametric Scheduling problem is concerned with the following two issues:

1. Determining whether the given job-set has a parametric schedule, i.e. a schedule as defined in Definition (2.1);
2. Computing the start-time of a job in each scheduling window, assuming that
   (a) The parametric schedulability query has been decided affirmatively, and
   (b) The start and execution times of all jobs sequenced before it are provided.

   This corresponds to the online dispatching phase.

The discussion above directs us to the following formulation of the parametric schedulability query:

$$\exists s_1 \; \forall e_1 \in [l_1, u_1] \; \exists s_2 \; \forall e_2 \in [l_2.u_2], \ldots \exists s_n \; \forall e_n \in [l_n, u_n] \quad \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}} \quad ? \tag{3}$$

3 Motivation

Parametric scheduling provides a means of addressing the lack of flexibility in static scheduling.

Example (1): Consider the two job system $J = \{J_1, J_2\}$*, with start times* $\{s_1, s_2\}$*, execution times in the set* $\{(e_1 \in)[2,4] \times (e_2 \in)[4,5]\}$ *and the following set of constraints:*

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

Clearly a static approach would declare the constraint system to be infeasible i.e. there do not exist rational $\{s_1, s_2\}$ which can satisfy the constraint set for all execution time vectors [SA00]. Now consider the following start time dispatch vector:

$$\vec{\mathbf{s}} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 0 \\ s_1 + e_1 \end{bmatrix} \tag{4}$$

This assignment clearly satisfies the input set of constraints and is hence a valid dispatch schedule. The key feature of the schedule provided by Equation (4) is that the start time of job J_2 is no longer an absolute time, but a (parameterized) function of the start and execution times of job J_1. Thus, the parametric schedulability query provides the flexibility to partially address the loss of schedulability phenomenon.

Parametric schedulability is particularly useful in real-time Operating Systems such as Maruti [LTCA89, MAT90] and MARS [DRSK89], wherein program specifications can be efficiently modeled through constraint matrices, and interactions between processes are permitted through linear relationships between their start and execution times. The flexibility of parametric schedulability greatly enhances the schedulability of a job-set, at the expense of computing dispatch vectors online.

4 Related Work

The *parametric model* for `axis-parallel hyper-rectangle` domains was proposed in [Sak94]. In [GPS95], polynomial time algorithms were presented for the case, in which the constraints are restricted to be standard (network, unimodular). The principal technique used in their algorithm was the Fourier- Motzkin elimination procedure to eliminate existentially quantified variables [DE73]. They showed that when the constraints are standard, the elimination procedure does not lead to an exponential increase in the set of resolvent constraints, a phenomenon observed when the constraints are arbitrary [HJLL90]. In Section §6, we shall provide a dual interpretation of their algorithm. [Cho97] and [Cho00] extend the results in [GPS95] to handle the case, in which inter-period constraints are permitted in the job-set. In [WDL91], a dynamic scheduling scheme is presented; however no offline guarantees are provided. Relative separation constraints, but only in restricted forms, are considered in [HL92b] and [HL92a]; in their model, certain distance constraints require to be satisfied between successive invocations of a job.

Our work here represents the first attempt to study the parametric scheduling problem from a computational complexity perspective and we show that the parametric schedulability query can be decided in polynomial space for arbitrary constraint sets.

5 Complexity of Parametric Schedulability

Prior to analyzing the complexity of the query, let us discuss a simple algorithm to decide the query. Quantifier elimination procedures help us to work through query (3), by eliminating one quantified variable at a time.

Algorithm (5.2) describes the procedure for eliminating the universally quantified execution variable $e_i \in [l_i, u_i]$. The Fourier-Motzkin elimination technique discussed in [CR99] represents one implementation of ELIM-EXIST-VARIABLE. In general, any polyhedral projection method suffices.

The correctness of Algorithm (5.2) has been argued in [GPS95], while the correctness of the Fourier-Motzkin procedure is discussed in [Sch87].

```
Function PARAMETRIC-SCHEDULER ($\Upsilon, \mathbf{A}, \vec{\mathbf{b}}$)
 1: for ( $i = n$ down to 2 ) do
 2:     ELIM-UNIV-VARIABLE($e_i$)
 3:     ELIM-EXIST-VARIABLE($s_i$)
 4: end for
 5: ELIM-UNIV-VARIABLE ($e_1$)
 6: if ( $a \le s_1 \le b, \quad a, b \ge 0$ ) then
 7:     Valid Parametric Schedule Exists
 8:     return
 9: else
10:     No Parametric Schedule Exists
11:     return
12: end if
```

Algorithm 5.1: A Quantifier Elimination Algorithm for determining Parametric Schedulability

```
Function ELIM-UNIV-VARIABLE ($\mathbf{A}, \vec{\mathbf{b}}$)
 1: Substitute $e_i = l_i$ in each constraint that can be written in the form $e_i \ge ()$
 2: Substitute $e_i = u_i$ in each constraint that ca be written in the form $e_i \le ()$
```

Algorithm 5.2: Eliminating Universally Quantified variable $e_i \in [l_i, u_i]$

Observation: 5.1 *Eliminating a universally quantified execution time variable does not increase the number of constraints.*

Observation: 5.2 *Eliminating an existentially quantified variable s_i in general, leads to a quadratic increase in the number of constraints, i.e. if there are m constraints, prior to the elimination, there could be $O(m^2)$ constraints after the elimination (See [DE73]). Thus, the elimination of k existential quantifiers could increase the size of the constraint set to $O(m^{2^k})$ [CR99]. Clearly, the exponential size blow-up, makes the Algorithm (5.1) impractical for general constraint sets.*

Lemma: 5.1 *Query (3) is locally convex over the execution time variables, i.e. if the query is true for $e_i = a_1$ and $e_i = b_1$, then it is true for $e_i = \lambda.a_1 + (1 - \lambda).b_1, 0 \le \lambda \le 1$.*

Proof: *Follows from the correctness of Algorithm (5.2).* □

Lemma (5.1) allows us to restate the parametric schedulability query as:

$$\exists s_1 \, \forall e_1 \in \{l_1, u_1\} \, \exists s_2 \, \forall e_2 \in \{l_2.u_2\}, \ldots \exists s_n \, \forall e_n \in \{l_n, u_n\} \quad \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \le \vec{\mathbf{b}} \quad ? \tag{5}$$

We shall refer to query (5) as the parametric schedulability query in subsection §5.1. Note that query (3) is equivalent to query (5) in that query (3) is true if and only if query (5) is true. The advantage of the discrete form is that it simplifies the complexity analysis of the query.

5.1 Parametric Schedulability as a 2−person game

We show that query (5) can be modeled as a 2−person game; the modeling helps us to establish that the query is `PSPACE-easy`.

Let $\mathcal{A}$ and $\mathcal{B}$ denote the two players involved in the following 2−person game:

1. The two players move alternatingly, with $\mathcal{A}$ making the first move, $\mathcal{B}$ making the second move and so on;
2. The game has $2.n$ moves;
3. In its i^{th} move, $\mathcal{A}$ guesses a valid start time s_i for Job J_i, where the guess is valid, if the guessed value lies in the scheduling window i.e $[0, L]$;

4. In its $(i+1)^{st}$ move, $\mathcal{B}$ guesses a valid execution time e_i, for Job J_i, where a guess is valid if $e_i \in \{l_i, u_i\}$;
5. At the end of $2.n$ moves $\mathcal{A}$ has guessed $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$, while $\mathcal{B}$ has guessed $\vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n]$;
6. If $\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \le \vec{\mathbf{b}}$ (See System (5)), the game is considered *a win for* $\mathcal{A}$; otherwise the game is considered *a win for* $\mathcal{B}$.

Definition: 5.1 *An Alternating Turing Machine (ATM) is a non-deterministic Turing machine $N = (K, \Sigma, \Delta, s)$, in which the set of states K is partitioned into two sets K_{AND} and K_{OR}, i.e. $K = K_{AND} \cup K_{OR}$. Let x be an input and consider the tree of computations of N on x. Each node in the tree is a configuration of the precise machine at that state and includes the step number of the machine. Starting from the leaves of the tree, and going up, we recursively define a subset of these configurations, called* eventually accepting configurations *as follows: First, all leaf configurations with state "yes" are eventually accepting. A configuration C in state K_{AND} is accepting, if and only if all its successor configurations are eventually accepting. A configuration C in state K_{OR} is accepting, if and only if at least one of its successor configurations is eventually accepting. Finally, we say that N accepts x, if the initial configuration is eventually accepting. We say that an Alternating Turing Machine N decides a language $\mathcal{L}_\infty$, if and only if N accepts all $x \in \mathcal{L}_\infty$ and rejects all strings $x \notin \mathcal{L}_\infty$.*

Definition: 5.2 $\mathtt{ATIME}(f(n))$ *is defined as the class of languages that can be decided by an Alternating Turing Machine, in at most $f(n)$ steps, on an input of size n* [2].

It is thus clear that

Lemma: 5.2 *The parametric schedulability query can be decided in* $\mathtt{ATIME}(m^2)$.

Proof: *Algorithm (5.3) describes the Alternating Turing machine that decides query (5). The only non-trivial computation is at the leaf of the tree and this computation involves the multiplication of a $m \times 2n$ matrix with a $2.n \times 1$ vector, giving a running time of* $\mathtt{ATIME}(m.n)$, *which can be expressed as* $\mathtt{ATIME}(m^2)$, *since $m > n$.* □

```
Function PARAM-DECIDE($\mathbf{A}, \vec{\mathbf{b}}, \Upsilon$)
1: for ( $i = 1$ to $n$ ) do
2:    Guess a value for $s_i \in [0, L]$ in state $K_{OR}$, of the computation tree
3:    Guess a value for $e_i \in \{l_i, u_i\}$ in state $K_{AND}$, of the computation tree
4: end for{ We now have both $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$ and $\vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n]$. }
5: if ( $\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \le \vec{\mathbf{b}}$ ) then
6:    There exists a parametric schedule
7: else
8:    There does not exist a parametric schedule
9: end if
```

Algorithm 5.3: An Alternating Turing Machine to decide Parametric Schedulability

Corollary: 5.1 *Parametric schedulability is* `PSPACE-easy`.

Proof: *From [Pap94], we know that,* $\mathtt{PSPACE} = \cup_{i=0}^{\infty} \mathtt{ATIME}(m^i)$. □

5.2 Parametric Co-scheduling

As part of our complexity analysis, we briefly study the complement of the parametric schedulability query; this query called the *Parametric Co-Scheduling query* is instrumental in explaining the difference between parametric scheduling and co-static scheduling.

The complement of the parametric schedulability query is given by:

[2] For details on Turing Machines, configurations and a more detailed exposition of Alternating Turing Machines, see [Pap94].

$$\neg(\ \exists s_1 \ \forall e_1 \in [l_1, u_1] \ \exists s_2 \ \forall e_2 \in [l_2.u_2], \ldots \exists s_n \ \forall e_n \in [l_n, u_n] \quad \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}} \ \ ? \ ) \tag{6}$$

which can be rewritten as:

$$\forall s_1 \ \exists e_1 \in [l_1, u_1] \ \forall s_2 \ \exists e_2 \in [l_2, u_2] \ldots \forall s_n \ \exists e_n \in [l_n, u_n] \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \not\leq \vec{\mathbf{b}} \quad ? \tag{7}$$

Query (7), called the *parametric co-scheduling query*, asks whether there exists an execution time vector in the form

$$\vec{\mathbf{e}} = \begin{bmatrix} e_1 = g_1(s_1) \\ e_2 = g_1(s_1, s_2) \\ e_3 = g_3(s_1, s_2, s_3) \\ \vdots \\ e_n = g_n(s_1, s_2, \ldots, s_n) \end{bmatrix} \tag{8}$$

where $g_i(s_1, s_2, \ldots, s_i)$ are functions which specify that the e_i could depend upon the particular start time chosen, such that the constraint system (1) is violated. Such a vector is called the break vector of the parametric system or a parametric break vector. *The difference between a parametric break vector and a co-static break vector is that in the co-static break vector all component elements are rational numbers, while in the parametric break vector, the i^{th} component element $e_{param}(i)$ could depend upon the start times $s_1, s_2, \ldots, s_i$ (see [Sub00].* It follows that the parametric schedulability specification is stronger than the co-static schedulability specification, inasmuch as parametric schedulability requires the absence of a parametric break vector, and not merely the absence of a co-static break vector, with elements in $\boldsymbol{\Upsilon}$.

A constraint system which is co-statically schedulable, but not parametrically schedulable is given in Example (2).

Example (2):

$$\begin{aligned} s_1 + e_1 &\leq s_2 \\ s_2 + e_2 &\leq s_1 + e_1 + 4 \\ s_2 + e_2 &\geq 8 \\ s_1 &\leq 4 \\ e_1 \in [1, 2], \qquad e_2 &\in [2, 4] \end{aligned}$$

We show that if query (7) is true, then we can restrict the range of each $g_i()$ to $\{l_i, u_i\}$ i.e. there exists a parametric break vector with component values in $\{l_i, u_i\}$. Thus, we can regard the $g_i()$ as *bi-valued functions* with $g_i(s_1, s_2, \ldots, s_i) = l_i$ or u_i, depending upon the values assumed by $s_1, s_2, \ldots, s_i$.

Lemma: 5.3 *Query (7) is true if and only if*

$$\forall s_1 \ \exists e_1 \in \{l_1, u_1\} \ \forall s_2 \ \exists e_2 \in \{l_2, u_2\} \ldots \forall s_n \ \exists e_n \in \{l_n, u_n\} \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \not\leq \vec{\mathbf{b}} \quad ? \tag{9}$$

Proof: *Consider a start time vector $\vec{\mathbf{s'}} = [s'_1, s'_2, \ldots, s'_n]$, such that Lemma (5.2) does not hold. Let e_a be the first job, where $e_a \in [l_a, u_a], e_a \neq l_a, u_a$. In the linear system (1), one or more of the constraints are violated. We show that e_a can be set to either l_a or e_a, maintaining the truth of query (7). Using induction, it follows that we can use the same argument for all e_i which do not belong to $\{l_i, u_i\}$. Setting $e_a = l_a$ causes one of the following things to happen:*

(a) The constraint system is still violated, in which case, the assignment preserves the answer to query (7), or

(b) The constraint system is satisfied, in which case set $e_a = u_a$. Clearly the constraint system must be violated, or else the constraint system is valid for all $e_a \in [l_a, u_a]$, violating the premise.

□

Using an argument that parallels the discussion of Lemma (5.2), it is clear that the Parametric Co-Scheduling query can also be decided in PSPACE. This is not surprising, since we know that PSPACE = co-PSPACE [Pap94].

6 Special Case Analysis

We analyze the problem `<aph|stan|param>` i.e. the schedulability problem in which the system is *parametric*, the constraints are standard (see Appendix (A) and the execution time domain is an axis-parallel hyper-rectangle. We provide a dual interpretation of the quantifier elimination algorithm, discussed in [Sak94].

Given an instance of `<aph|stan|param>`, we use the procedure in Appendix §A to construct the dual graph. Algorithm (6.1) takes the dual graph as input and decides the schedulability query over the dual. The principal difference between this algorithm and Algorithm (5.1) lies in the implementation of the existential variable elimination procedure. When the constraints are standard, existential variable elimination can be performed through vertex contraction, in the manner discussed in Algorithm (6.2). The exponential increase in the number of constraints that is a feature of the Algorithm (5.1) is prevented in this case; Algorithm (6.3) deletes redundant edges (constraints) as new edges (constraints) are created as a result of vertex contraction, thereby bounding the total number of constraints at any time in the computation.

```
Function PARAM-SCHEDULE-STANDARD ($G =< V, E >$)
 1: for ( $i = n$ down to 2 ) do
 2:   Substitute $e_i = u_i$ on all edges where $e_i$ is prefixed with a negative sign
 3:   Substitute $e_i = l_i$ on all other edges { We have now eliminated $e_i$ in $\forall e_i \in [l_i, u_i]$ }
 4:   $G' =< V', E' >$=VERTEX-CONTRACT( $s_i$ )
 5: end for
 6: Substitute $e_1 = u_1$ on all edges, where $e_1$ is prefixed with a negative sign
 7: Substitute $e_1 = l_1$ on all other edges { At this point, the only edges in the graph are between $s_1$ and $s_0$ and
    the weights on these edges are rational numbers.}
 8: Find the edge $s_1 \leadsto s_0$ with the smallest weight, say $u$.  { $u \geq 0$ }
 9: Find the edge $s_0 \leadsto s_1$ with the largest weight in magnitude, say $l$.  { $l \leq 0$ }
10: if ( $-l \leq u$ ) then
11:   return $[-l, u]$ as the range in which $J_1$ can begin execution
12: else
13:   There is no Parametric Schedule
14:   return
15: end if
```

Algorithm 6.1: Implementing quantifier elimination over the dual graph

Observation: 6.1 *For deleting redundant edges between a vertex s_a and another vertex s_b, where $a > b$, we use a separate function $\geq_{back}$ (e_{old}, e_{new}), which retains the edge with the smaller (positive) weight. See also Observation (A.4);*

Observation: 6.2 *The number of edges from any vertex s_a to any other vertex s_b (say $a < b$) does not exceed four at any time; in fact, after the elimination of e_b, the number of edges does not exceed two.*

Observation: 6.3 *The class of standard constraints is closed under execution time variable elimination, i.e. the elimination of the execution time variables does not alter the network structure of the graph;*

Observation: 6.4 *The class of standard constraints is closed under vertex contraction. A naive implementation of* VERTEX-CONTRACT() *would cause the number of edges between the two vertices to increase quadratically; our observations in Appendix §A provide us with a means of eliminating redundant edges on the fly, in $O(1)$ time;*

Observation: 6.5 *The only manner in which infeasibility is detected is through the occurrence of a negative weight self-loop on any vertex (Step (5) of Algorithm (6.2) and Steps (7-13) of Algorithm (6.1).) These loops could occur in two ways:*

1. *The contraction of a vertex results in a self-loop with a negative rational number on another vertex; e.g. in the constraint set $s_1 + 8 \leq s_2, s_2 \leq s_1 + 7$, contraction of s_2 results in a self-loop at s_1 of weight -1;*

```
**Function** VERTEX-CONTRACT ($G =< V, E >, s_i$)
for each edge $s_j \leadsto s_i$, with weight $w_{ji}$ do
  for each edge $s_i \leadsto s_k$, with weight $w_{ik}$ do
    Add an edge ( say $e_{new}$ ) $s_j \leadsto s_k$ with weight $w_{ji} + w_{ik}$
    if $j = k$ then
      if ( $w_{ji} + w_{ik} < 0$ ) then
        No Parametric Schedule Exists { See Observation (6.5)}
        return( false )
      end if{ Eliminating self-loops }
    else
      Discard $e_{new}$
      continue {We do not add self-loops to the edge set}
    end if
    $E' = E \cup e_{new}$
    REMOVE-REDUNDANT( $G =< V, E' >, s_j, s_k, e_{new}$ )
  end for
  $E' = E' - (s_j \leadsto s_i)$
end for
for each edge $s_i \leadsto s_k$, with weight $w_{ik}$ do
  $E' = E' - (s_i \leadsto s_k)$
end for
$V' = V - \{s_i\}$ { We have now eliminated $s_i$ in $\exists s_i$ }
```

Algorithm 6.2: Vertex Contraction

```
**Function** REMOVE-REDUNDANT($G =< V, E >, s_j, s_k, e_{new}$)
{ W.l.o.g. assume that $j < k$.}
{ $e_{new}$ is the new edge that is added from $s_j$ to $s_k$. }
Let $t_o$ = type of $e_{new}${ $t_o$ must be one of the four edge types discussed in (A.2)}
if ( there does not exist an edge from $s_j$ to $s_k$ having type $t_o$ ) then
  Retain $e_{new}$ {The addition of $e_{new}$ does not cause the number of edges from $s_j$ to $s_k$ to increase beyond
  four.}
else
  Let $e_{old}$ be the existing edge with the same type as $e_{new}$
  if ( $\geq_{for}$ ($e_{old}, e_{new}$ ) then
    Discard $e_{new}$
  else
    Discard $e_{old}$
  end if
end if
```

Algorithm 6.3: Redundant edge elimination

```
**Function** $\geq_{for}(e_{old}, e_{new})$
Symbolically compare weights on both edges
if ( $e_{old}$ has the more negative weight ) then
  return( true )
else
  return( false )
end if
```

Algorithm 6.4: Implementation of $\geq_{for}$

2. *The contraction of a vertex results in a self-loop of the following form: $e_a - 7$, on vertex s_a. In this case, either $l_a \geq 7$, in which case the edge can be discarded (redundant), or the system is infeasible.*

6.1 Correctness and Complexity

The correctness of the algorithm follows from the correctness of the quantifier elimination algorithm (5.1).

The elimination of an universally quantified execution time variable e_i takes time proportional to the degree of vertex s_i, since e_i occurs only on those edges that represent constraints involving s_i. Hence eliminating e_i takes time $O(n)$ in the worst case. The total time taken for execution time variable elimination over all n vertices is thus $O(n^2)$. The contraction of a single vertex takes time $O(n^2)$ in the worst-case, since every pair of incoming and outgoing edges has to be combined. In fact $O(n^2)$ is a lower-bound on the contraction technique. *However, the total number of edges in the graph is always bounded by* $O(n^2)$; the total time spent in vertex contraction is therefore $O(n^3)$.

Thus the complexity of `<aph|stan|param>` is $O(n^3)$.

7 Complexity of Online Dispatching

In [Sak94] and [Cho97], query (3) was addressed by providing a parametrized list of linear functions for the start time of each job, as shown in Table (1). During actual execution, s_1 can take on any value in the range $[a, b]$. Upon termination of job J_1, we know e_1 which along with s_1 can be plugged into $f_1()$ and $f_1'()$, thereby providing a range $[a', b']$ for s_2 and so on, till job J_n is scheduled and completes execution.

Lower bound function	$\leq$ Start time $\leq$	Upper bound function
a	s_1	b
$f_1(s_1, e_1)$	s_2	$f_1'(s_1, e_1)$
$f_2(s_1, e_1, s_2, e_2)$	s_3	$f_2'(s_1, e_1, s_2, e_2)$
$\vdots$	$\vdots$	$\vdots$
$f_{n-1}(s_1, e_1, s_2, e_2, \ldots, s_{n-1}, e_{n-1})$	s_n	$f_{n-1}'(s_1, e_1, s_2, e_2, \ldots, s_{n-1}, e_{n-1})$

Table 1: List of parametric functions

The principal problem with the creation of the function lists is that we cannot *a priori* bound the length of these lists. We argue here that explicit construction of the parameterized function list is unnecessary. Determination of feasibility is sufficient, thereby eliminating the need for storing the parameterized function list. Observe that at any point in the scheduling window, the first job that has not yet been scheduled has a start time that is independent of the start and execution times of all other jobs. Once this job is executed, we can determine a rational range, e.g.$[a'', b'']$ for the succeeding job and the same argument applies to this job. In essence, all that is required to be determined is *the start time of the first unexecuted job in the sequence.*

Let us assume the existence of an oracle $\mathbf{\Omega}$ that decides query (3) in time $T(\mathbf{\Omega})$. Algorithm (7) can then be used to determine the start time of the first unexecuted job (say J_ρ [3]) in the schedule :

The end of the period L is the deadline for all jobs in the job-set. We must have $0 \leq s_\rho \leq L$. The goal is to determine the exact value that can be safely assigned to s_ρ without violating the current set. Let

$$\mathbf{G}^\rho.\vec{\mathbf{s}^\rho} + \mathbf{H}^\rho.\vec{\mathbf{e}^\rho} \leq \vec{\mathbf{b}^\rho} \tag{10}$$

denote the current constraint set, where

- $\mathbf{G}^\rho$ is obtained from $\mathbf{G}$, by dropping the first $(\rho - 1)$ columns; $\mathbf{G}^{\mathbf{1}-\rho}$ represents the first $(\rho - 1)$ columns of $\mathbf{G}$,
- $\mathbf{H}^\rho$ is obtained from $\mathbf{H}$, by dropping the first $(\rho - 1)$ columns; $\mathbf{H}^{\mathbf{1}-\rho}$ represents the first $(\rho - 1)$ columns of $\mathbf{H}$,

[3] At commencement, $\rho = 1$

- $\vec{\mathbf{s}^{\rho}} = [s_\rho, s_{\rho+1}, \ldots, s_n]$; $\vec{\mathbf{s}^{1-\rho}} = [s_1, s_2, \ldots s_{\rho-1}]$;
- $\vec{\mathbf{e}^{\rho}} = [e_\rho, e_{\rho+1}, \ldots, e_n]$; $\vec{\mathbf{e}^{1-\rho}} = [e_1, e_2, \ldots e_{\rho-1}]$, and
- $\vec{\mathbf{b}^{\rho}} = \vec{\mathbf{b}} - (\mathbf{G}^{1-\rho}.\vec{\mathbf{s}^{1-\rho}} + \mathbf{H}^{1-\rho}.\vec{\mathbf{e}^{1-\rho}})$.

```
Function Determine-Start-Time ($\mathbf{G}^\rho, \mathbf{H}^\rho, \vec{\mathbf{b}^\rho}, [a_l, a_h]$)
1:  { Initially $[a_l, a_h] = [0, L]$; the interval is reduced to half its original length at each level of the recursion}
2:  Let $m' = \frac{a_h - a_l}{2}$
3:  if ($\Omega(\mathbf{G}^\rho, \mathbf{H}^\rho, \vec{\mathbf{b}^\rho}, s_\rho \geq m')$) then
4:    {We now know that there is a valid assignment for $s_\rho$ in the interval $[m', a_h]$; the exact point in time needs
      to be determined}
5:    if ( $\Omega(\mathbf{G}^\rho, \mathbf{H}^\rho, \vec{\mathbf{b}^\rho}, s_\rho = m')$ ) then
6:      $s_\rho = m'$
7:      return
8:    else
9:      { $m'$ is not a valid point; however we can still recurse on the smaller interval}
10:     Determine-Start-Time ($\mathbf{G}^\rho, \mathbf{H}^\rho, \vec{\mathbf{b}^\rho}, [m', a_h]$)
11:   end if
12: else
13:   { We know that the valid assignment for $s_\rho$ must lie in the interval $[a_l, m']$ }
14:   Determine-Start-Time ($\mathbf{G}^\rho, \mathbf{H}^\rho, \vec{\mathbf{b}^\rho}, m'$)
15: end if
```

Algorithm 7.1: Parametric Dispatcher to determine s_ρ

Algorithm (7.1) exploits the local convexity of s_ρ, i.e. if $s_\rho \geq a$ is valid and $s_\rho \leq b$ is valid, then any point $s_\rho = \lambda.a + (1-\lambda).b, 0 \leq \lambda \leq 1$ is valid. The cost of this strategy is $O(\log L)$ calls to the oracle $\mathbf{\Omega}$, i.e. $O(T(\mathbf{\Omega}).\log L)$. We have thus established that the principal complexity of the parametric scheduling problem is in deciding query (3). *This result is significant because it decouples dispatching complexity from decidability. In many complexity analyses involving* `PSPACE` *problems, the size of the output is used to provide a lower bound for the running time of the problem, e.g. if we can provide a problem instance where the start time of a job must have exponentially many dependencies, due to the nature of the constraints, then we have an exponential lower bound for the dispatching scheme in [GPS95]. Algorithm (7.1) assures us that efficient dispatching is contingent only upon efficient decidability.*

8 Summary

In this paper, we studied the computational complexity of the parametric schedulability query. A naive implementation of the quantifier elimination algorithm, as suggested in [Sak94], requires exponential space and time to decide the query. Our work establishes that the problem is `PSPACE-easy` for arbitrary constraint classes.

We also analyzed the dual of `<aph|stan|param>`; the analysis enabled us to design an $O(1)$ dynamic dispatching algorithm, using one additional processor per job. In real-time systems, more often than not, it is preferable to achieve timing goals at the expense of other resources (increased processor requirements, in our case) and our algorithm provides a marked improvement over the dispatching algorithms in the literature [Sak94, Cho97].

Table (2) summarizes the results discussed in this paper.

A Construction of Dual Graph for Standard Constraints

The purpose of this appendix is to provide a step-by-step procedure for constructing the dual graph, when the system constraints are standard.

	`<aph|arb|param>`	`<aph|stan|param>`	`<aph|net|param>`
Schedulability	PSPACE-easy	$O(n^3)$	open
Online Dispatching	$O(T(\mathbf{\Omega}).\log L)$	$O(1)$	$O(T(\mathbf{\Omega}).\log L)$

Table 2: Summary of Results for Parametric Scheduling

Given a set of n jobs, with standard constraints imposed on their execution, we create a graph $G =< V, E >$, where V is the set of vertices and E is the set of edges.

1. $V =< s_0, s_1, s_2, \ldots, s_n >$, i.e. one node for the start times of each job, and node s_0 which is used for handling absolute constraints;
2. For every constraint of the form: $s_i + k \leq s_j$, construct and arc $s_i \rightsquigarrow s_j$, with weight $-k$;
3. For every constraint of the form: $s_i + e_i \leq s_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $k - e_i$;
4. For every constraint of the form: $s_i \leq s_j + e_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $e_j + k$;
5. For every constraint of the form: $s_i + e_i \leq s_j + e_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $e_j - e_i + k$;
6. For every constraint of the form: $s_i \leq c$, construct an arc $s_i \rightsquigarrow s_0$, with weight c;
7. For every constraint of the form: $s_i \geq c$, construct an arc $s_0 \rightsquigarrow s_i$, with weight $-c$;
8. For every constraint of the form: $s_i + e_i \leq c$, construct an arc $s_i \rightsquigarrow s_0$, with weight $c - e_i$;
9. For every constraint of the form: $s_i + e_i \geq c$, construct an arc $s_0 \rightsquigarrow s_i$, with weight $e_i - c$;
10. Finally construct arc $s_o \rightsquigarrow s_1$ with weight 0, since $s_1 \geq 0$ and arc $s_n \rightsquigarrow s_0$ with weight $L - e_n$, since all jobs have to be completed by the end of the current scheduling window.

Observation: A.1 *In the dual graph, there are $n + 1$ vertices and m edges, corresponding to a job-set with n jobs and m standard constraints on their execution.*

Observation: A.2 *There are at most 4 edges from node s_i and s_j; we classify them as:*

1. Type 1 - *An edge $s_i \rightsquigarrow s_j$ with weight k_1, representing temporal distance between the start times of J_i and J_j;*

2. Type 2 - *An edge $s_i \rightsquigarrow s_j$ with weight $-e_i + k_2$, representing temporal distance between the finish time of J_i and the start time of J_j;*

3. Type 3 - *An edge $s_i \rightsquigarrow s_j$ with weight $e_j + k_3$, representing temporal distance between the start time of J_i and the finish time of J_j;*

4. Type 4 - *An edge $s_i \rightsquigarrow s_j$ with weight $e_j - e_i + k_4$, representing temporal distance between the finish times of J_i and J_j.*

where, $k_1, k_2, k_3, k_4 \in \Re$.

Observation: A.3 *There are at most 2 edges between any job node s_i and the node s_0.*

Corollary: A.1 *In case of standard constraints, the dual graph has at most $O(n^2)$ edges.*

Proof: *Follows from the fact there are at most $(n + 1).n$ vertex pairs, with at most 4 edges between them.* □

Observation: A.4 *A forward edge, i.e. an edge $s_i \rightsquigarrow s_j, i < j$, dictates the degree of separation required between J_i and J_j , whereas a backward edge $s_j \rightsquigarrow s_i, j > i$ dictates the degree of closeness required between them.*

Example (3): We construct the dual graph for a 4−job set $\{J_1, J_2, J_3, J_4\}$, subject to a set of standard constraints.

$$\begin{aligned} 4 \le e_1 \le 8,\ 6 \le e_2 \le 11,\ 10 \le e_3 \le 13,\ 3 \le e_4 \le 9 \\ s_4 + e_4 \le 56 \\ s_4 + e_4 \le s_3 + e_3 + 12 \\ s_2 + e_2 + 18 \le s_4 \\ s_3 + e_3 \le s_1 + e_1 + 31 \\ 0 \le s_1,\ s_1 + e_1 \le s_2,\ s_2 + e_2 \le s_3, s_3 + e_3 \le s_4 \end{aligned} \tag{11}$$

Figure (1) represents the corresponding dual graph.

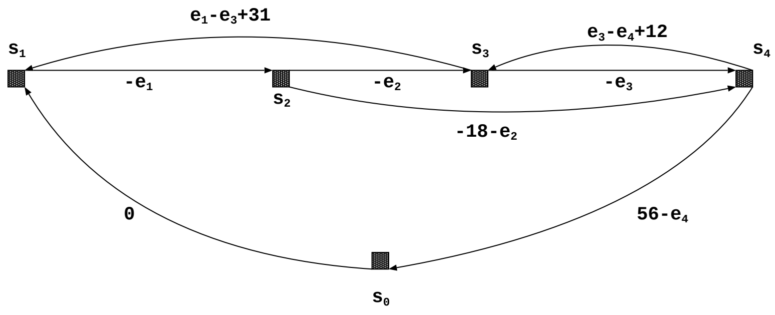


Figure 1: Dual graph of System (11)

B Illustration of Algorithm (6.1)

Example (4): Consider an instance of `<aph|stan|param>`*, in which the underlying constraint system is represented by Figure (1) and the parametric specification is given by query (12). Figures (2-4) display the application of Algorithm (6.1) to the dual graph.*

$$\exists s_1\ \forall e_1 \in [4,8]\ \exists s_2\ \forall e_2 \in [6,11]\ \exists s_3\ \forall e_3 \in [10,13]\ \exists s_4\ \forall e_4 \in [3,9] \quad \{(11)\} \quad ? \tag{12}$$

The final output is $0 \le s_1 \le 10$.

References

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

[Cho00] Seonho Choi. Dynamic dispatching of cyclic real-time tasks with relative time constraints. *JRTS*, pages 1–35, 2000.

[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.

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

[HJLL90] Huynh, Joskowicz, Lassez, and Lassez. Reasoning about linear constraints using parametric queries. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 10, 1990.

[HL92a] C. C. Han and K. J. Lin. Scheduling Distance-Constrained Real-Time Tasks. In *Proceedings, IEEE Real-time Systems Symposium*, pages 300–308, Phoenix, Arizona, December 1992.

[HL92b] C. C. Han and K. J. Lin. Scheduling real-time computations with separation constraints. *Information Processing Letters*, 12:61–66, May 1992.

[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.

[MAT90] D. Mosse, Ashok K. Agrawala, and Satish K. Tripathi. Maruti a hard real-time operating system. In *Second IEEE Workshop on Experimental Distributed Systems*, pages 29–34. IEEE, 1990.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, New York, 1994.

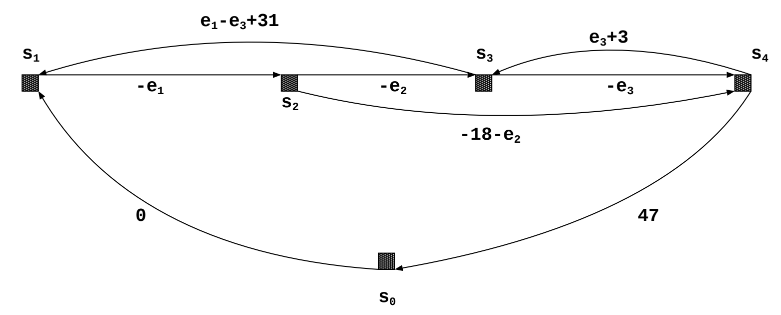
[SA00] K. Subramani and A. K. Agrawala. The static polytope and its applications to a scheduling problem. 3^{rd} *IEEE Workshop on Factory Communications*, September 2000.

[Sak94] Manas Saksena. *Parametric Scheduling in Hard Real-Time Systems.* PhD thesis, University of Maryland, College Park, June 1994.

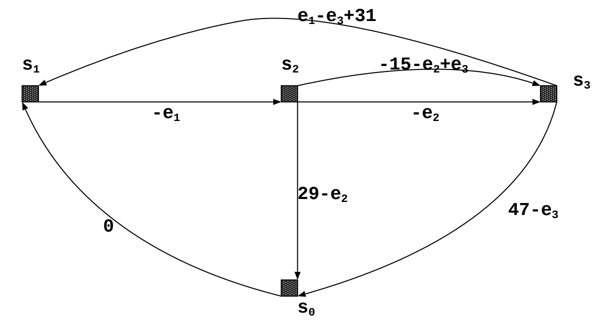
[Sch87] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1987.

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

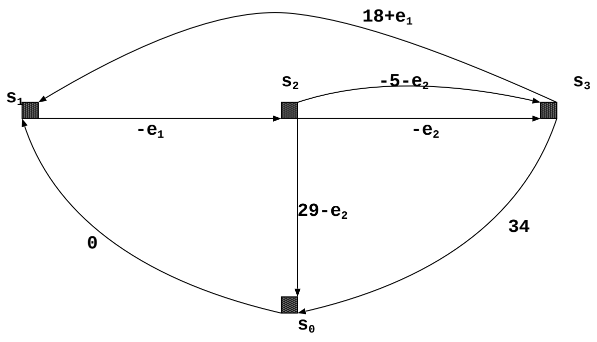
[WDL91] V. Wolfe, S. Davidson, and I. Lee. Rtc: Language support for real-time concurrency. In *Proceedings IEEE Real-Time Systems Symposium*, pages 43–52, December 1991.



Eliminating e_4



Eliminating s_4



Eliminating e_3

Figure 2: Algorithm (6.1) on query (12)

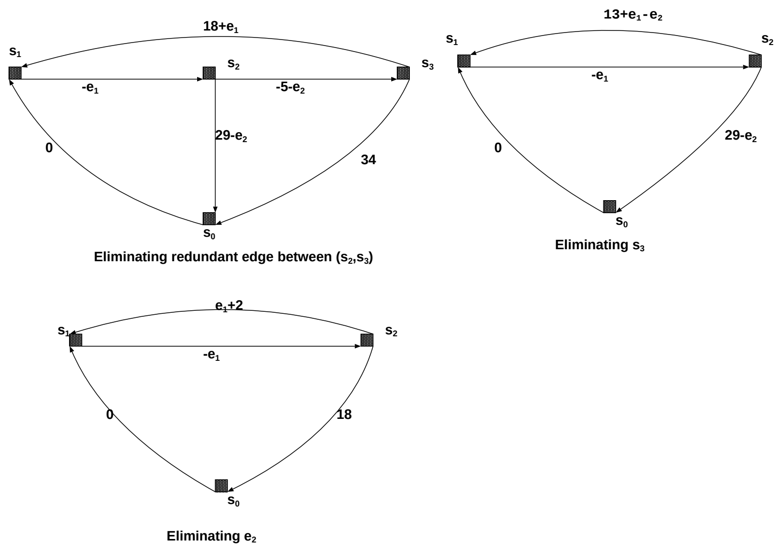


Figure 3: Algorithm (6.1) on query (12) (contd.)

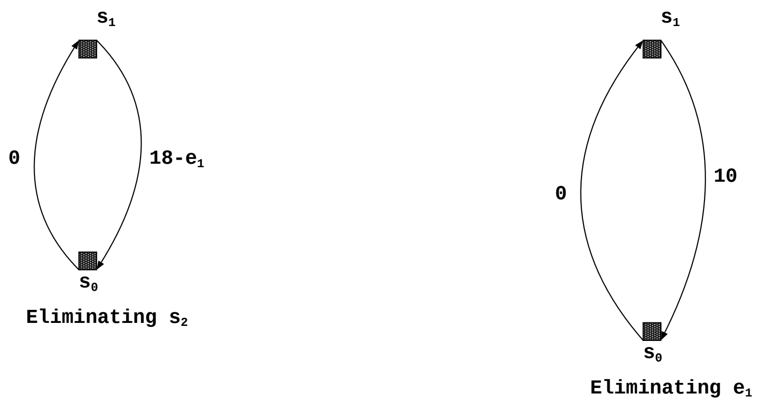


Figure 4: Algorithm (6.1) on query (12) (contd.)