Parametric Scheduling for network constraints

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

Abstract

The problem of parametric scheduling is concerned with checking whether a job-set is parametrically schedulable, subject to a set of imposed constraints. In real-time scheduling, parameterization of the schedule plays an important role in extending the flexibility of the scheduler, particularly in the presence of variable execution times. It has been shown that the existence of parametric schedules can be determined in polynomial time when the constraints are restricted to those that can be represented by a network, unimodular matrix. In this paper, we extend the class of constraints for which parametric schedules can be determined efficiently to include *network constraints*, such as weighted sum of completion times.

1 Introduction

Uncertainity in problem parameters is often a feature of real-time scheduling [AB98, AB99, SSRB98]. In the literature, there exist two broad approaches to address uncertainity, viz. stochastic and deterministic. In stochastic scheduling, the goal is to provide probabilistic guarantees that the constraints imposed on the job-set will be met, under the assumption that the non-constant problem paramters belong to a fixed well-understood distribution [Pin95, DLK82, MR, SS]. This approach is not applicable in the case of "hard" real-time systems [SSRB98, Sak94, LTCA89], where the guarantees have to be absolute i.e. there is no room for error. Hard real-time systems are typically composed of mission-critical tasks, wherein the consequences of failure, i.e. a violation of constraints can be catastrophic. Hard real-time systems call for deterministic approaches to the issue of uncertainity. Some of the common approaches include *worst-case* assumptions [Pin95, Bru81], static scheduling [SA00b] and parametric scheduling [GPS95, Cho97]. Worst-case assumptions regarding execution times (say) run the risk of constraint violation at *run-time* [Sub00a] and hence the strategy is not always correct. Static approaches make extremely conservative assumptions about each constraint in order to determine the existence of a feasible schedule. This approach is correct inasmuch as the goal is to provide a set of start-times that cannot cause constraint violation. However, static scheduling is extremely inflexible and even simple constraint sets (see Section §3) may not have static schedules. Parametric scheduling attempts to combine the guarantees provided by static scheduling with the flexibility of stochastic scheduling to enhance the class of constraints for which feasible schedules exist.

In real-time scheduling literature, it has been shown that polynomial time algorithms exist for determining parametric schedules, when the constraints are restricted to be "standard" [GPS95, Cho00]. In this paper we are concerned with the following question: *Can parametric schedulability be determined efficiently for non-standard constraints, with at most 2 tasks per constraint ?* We provide an affimative answer to the above question by designing a polynomial time algorithm for *network constraints*.

The rest of this paper is organized as follows: Section §2 provides a formal description of the problem under consideration. Section §3 discusses the motivation underlying our research while related approaches are detailed in Section §4. Our algorithm and the accompanying analysis are presented in Section §5. We discuss online dispatching techniques in Section §6 and conclude by summarizing our results in Section §7.

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, in which every row represents a *network* constraint (to be defined below),
- $\mathbf{E}$ is the convex set defined by the axis-parallel hyper-rectangle (`aph`)

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

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

The characterization of $\mathbf{E}$ as an `aph` is intended to model the fact that the execution time e_i of job J_i is not a fixed constant, but can assume any value in the pre-specified range $[l_i, u_i]$, depending on factors such as *loop-length*. In real-time systems such as Maruti, a number of runs of the job-set are carried out to statistically estimate the range $[l_i, u_i]$ [MAT90].

Observe that System (1) can be rewritten in the form

$$\mathbf{G}.\vec{\mathbf{s}} + \mathbf{H}.\vec{\mathbf{e}} \leq \vec{\mathbf{b}}, \quad \vec{\mathbf{e}} \in \mathbf{E} \tag{3}$$

Definition: 2.1 *Standard Constraint: A constraint is defined to be standard, if represents a relative separation relationship between at most two tasks, i.e. matrices* $\mathbf{G}$ *and* $\mathbf{H}$ *are flow graph, unimodular, with the added provision, that the entry* $\mathbf{H}[i, j]$ *must equal* $\mathbf{G}[i, j]$ *if it is non-zero.*

For instance a constraint of the form: $s_i - s_j \leq -8$, which specifies that Job J_j must start 8 units after job J_i starts is a standard constraint. A detailed description of standard constraints is available in [GPS95]. Standard constraints can be represented by edges of a flow graph [SA00a, DMP91].

Definition: 2.2 *Network Constraint: A constraint is said to be a network constraint, if it can be expressed in the form*

$$a.s_i + b.s_j \leq c.e_i + d.e_j, a, b, c, d \in \mathbf{Q} \tag{4}$$

i.e. all co-efficients are arbitary rational numbers.

Network constraints also have a graph structure; the "edge" representing the set of constraints between two jobs J_i and J_j form a polyhedron in the 4 variables s_i, e_i, s_j, e_j, with e_i and e_j being universally quantified [AS79, HN94].

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 $\{J_1, J_2, \ldots, J_{a-1}\}$.

Definition: 2.3 *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* $\vec{\mathbf{e}} \in \boldsymbol{\Upsilon}$*.*

The combination of the Job model, Constraint model and the Query model constitutes a scheduling problem specification within the `E-T-C` scheduling framework [Sub00a].

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 (2.3);
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:

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

3 Motivation

The motivation for Parametric Scheduling has been provided in great detail in [GPS95, Sak94, Cho97, Cho00, Sub00a]. The key issue is that a simple constraint system such as:

$$\begin{aligned} s_1 + e_1 &\leq s_2 \\ s_2 &\leq s_1 + e_1 + 1 \\ e_1 &\in [3, 5] \end{aligned}$$

does not have a static schedule i.e. no assignment of rational numbers to the start times can guarantee the meeting of both constraints for all values of e_1 in the range $[3, 5]$. Note that the schedule represented by

$$\begin{aligned} s_1 &= 0 \\ s_2 &= s_1 + e_1 \end{aligned}$$

is a valid, feasible, albeit parametric schedule.

An interesting line of research is obtaining schedules that satisfy certain optimization criteria. The complexity of *Parametric Optimization* for general constraints is not known [Joh]. However, we can approximate optimization functions involving at most two tasks through the use of network constraints. Optimization criteria, formulated as *performance metrics* arise in various situations including Job-shop, Flow-shop and Machine-Shop [Pin95, Bru81]. Typical performance metrics are *Makespan*, *Sum of Completetion times*, *Weighted sum of completion times* (also called aggregate metrics in the Operations Research literature), *Lateness* and *Tardiness*. For instance, the need to minimize the weighted sum of completion times of jobs J_1 and J_2 can be approximated through $w_1.(s_1 + e_1) + w_2.(s_2 + e_2) \leq k$, for suitably chosen k [SSRB98, SR88]. The other performance metrics can be similary approximated.

4 Related Work

The concept of Parametric Scheduling was introduced in [Sak94]. In [GPS95], polynomial time algorithms were presented for the case, when the constraints imposed on the job-set are "standard". [SA00a] argued that "standard" constraints could be represented by a flow, graph (i.e. the matrix $\mathbf{A}$ is network, unimodular). [Cho97] and [Cho00] extend the "standard constraint" model to include the case, in which constraints can exist between adjacent windows. In [Sub00a], it is shown that the problem can be solved in `PSPACE`, when the constraints are arbitrary. However, no hardness result is known for this problem [Joh, All].

A curious feature of the algorithms proposed in the literature for Parametric Schedulability is that none of them exploit the ordering information in the job-set. In both [GPS95] and [Cho00] polynomial time bounds are derived by observing that the number of relative constraints between any two tasks is bounded, if only strict relative constraints are permitted. In this paper, we explicitly take into account the ordering information to develop two new concepts viz. *Constraint Domination* and *Constraint Orientation*, which in turn are used to develop polynomial time algorithms for testing parametric schedulability in arbitrary network constraints.

Linear programs with at most 2 variables per constraint (LI(2)s) have received quite a bit of attention in the Operations Resarch community. [Sho81] was the first to oberve the correspondence between LI(2)s and graphs; [AS79] gave the first polynomial time algorithm for this problem. In [HN94], the Fourier-Motzkin elimination procedure was used to provide a *strongly polynomial* algorithm which to date is the fastest known algorithm for this problem.

5 Algorithms and Complexity

Algorithm (5.1) presents our strategy for testing parametric schedulability.

Function PARAMETRIC-SCHEDULER $(\mathbf{\Upsilon}, \mathbf{A}, \vec{\mathbf{b}})$
1: **for** ($i = n$ **down to** 2) **do**
2: ELIM-UNIV-VARIABLE(e_i)
3: **if** (CHECK-INCONSISTENCY()) **then**
4: **return (false)**
5: **end if**
6: PRUNE-CONSTRAINTS()
7: ELIM-EXIST-VARIABLE(s_i)
8: **if** (CHECK-INCONSISTENCY()) **then**
9: **return (false)**
10: **end if**
11: **end for**
12: ELIM-UNIV-VARIABLE (e_1)
13: **if** ($a \leq s_1 \leq b, \quad a, b \geq 0$) **then**
14: `Valid Parametric Schedule Exists`
15: **return**
16: **else**
17: `No Parametric Schedule Exists`
18: **return**
19: **end if**

Algorithm 5.1: A Quantifier Elimination Algorithm for determining Parametric Schedulability

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 [VR99] represents one implementation of ELIM-EXIST-VARIABLE. In general, any polyhedral projection method suffices. In our work, we assume that the Fourier-Motzkin procedure is used to eliminate the existentially quantified (start-time) variables [1]. When a

[1] A detailed exposition of the Fourier-Motzkin elimination procedure is available in [VR99].

variable (start-time or execution time) is eliminated, inconsistencies and/or redundancies could result. CHECK-INCONSISTENCY() identifies incinsitencies and declares the system to be infeasible, while PRUNE-CONSTRAINTS() identifies redundancies and eliminates them.

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 $e_i \geq ()$
2: Substitute $e_i = u_i$ in each constraint that can be written in the form $e_i \leq ()$

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

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

We point out that Algorithm (5.1) is similar to the ones outlined in the literature. The difference is the implementation of procedures PRUNE-CONSTRAINTS() and the analysis provided in §5.1.

5.1 Analysis

Observe that the procedure ELIM-UNIV-VARIABLE() does not increase the number of constraints. However, ELIM-EXIST-VARIABLE() has the potential to increase the number of constraints substantially. Assuming that the Fourier-Motzkin elimination algorithm is used, the elimination of k start-time variables, could result in the creation of as many as m^{2^k} constraints. One such pathological example is provided in [Sch87]. In [GPS95] and [SA00a], it was pointed out that "standard" constraints are closed under Fourier-Motzkin elimination i.e. the elimination of an existential variable results in the set of constraints staying standard. Using the notation, in [SA00a], this corresponds to saying that contracting a vertex of the flow graph representing the constraint set, does not destroy the its graph structure. Since a graph has at most $O(n^2)$ edges at all times, the polynomiality of the algorithm for standard constraints follows.

In our case though, there is no obvious way to either represent the set of constraints or bound their number under Fourier-Motzkin elimination, since in addition to relative constraints, we also have sum constraints, as discussed in Section §3. We make the following observation:

Observation: 5.1 *Network constraints are closed under Fourier-Motzkin elimination, using Algorithm (5.1).*

Observation (5.1) follows from the fact an existential variable i.e. a start time variable is eliminated only *after* the corresponding execution time variable has been eliminated. Consequently, its elimination results in a network constraint between two other jobs. For instance consider the following constraint set:

1. $s_3 \leq s_1 + e_1 + 14$;
2. $s_1 + s_3 \leq 22$;
3. $s_2 + 22 \leq s_3 + e_3$;
4. $e_3 \in [3, 5]$.

The elimination of e_3 results in:

1. $s_3 \leq s_1 + e_1 + 14$;
2. $s_1 + s_3 \leq 22$;
3. $s_2 + 19 \leq s_3$;

The elimination of s_3 (by pairing off constraints in which s_3 occurs with opposite polarity) gives rise to the following set of constraints:

1. $s_2 + 19 \leq s_1 + e_1 + 14$;

2. $s_2 + 19 \leq 22 - s_1$;

The key point is that the network structure is preserved, under the elimination. Observe that if e_3 were not eliminated, prior to eliminating s_3 the closure claim of Observation (5.1) no longer holds.

Let S_{ij} denote the set of constraints between the two jobs J_i and J_j) $(i < j)$. We now present an informal overview on the nature of constraints $l \in S_{ij}$. Informally, a relative constraint between two jobs either specifies increased separation between them or decreased separation. For instance, the constraint $s_1 + 8 \leq s_2$ specifies increased separation, while the constraint $s_2 \leq s_1 + 17$ specifies decreased separation. An aggregate constraint either pushes the jobs to the left or to the right. For instance the constraint $s_1 + s_2 \leq 7$ pushes jos J_1 and J_2 to the left i.e. towards 0, while the constraint $s_3 + s_4 \geq 8$ pushes jobs J_3 and J_4 towards the right, i.e. towards L.

To proceed with our analysis, we need the following definitions. We associate a type with every constraint, specifying whether it a relative constraint or an aggregate constraint.

Definition: 5.1 *Sum (Aggregate) constraint: A network constraint of the form $a.s_i + b.s_j \leq ()$ is said to be a sum constraint if both a and b have the same sign.*

For instance, $s_1 + s_2 \leq 7$ and $-3.s_1 - 4.s_2 \leq -9$ are sum constraints.

Definition: 5.2 *Difference (Relative) constraint: A network constraint of the form $a.s_i + b.s_j \leq ()$ is said to be a difference constraint, if a and b have opposite signs.*

For instance, the constraint $s_1 - s_2 \leq -4$ is a difference constraint.

Definition: 5.3 *Constraint orientation (Right): A constraint $l \in S_{ij}$ is said to have a* right orientation *if it specifies increased separation between J_i and J_j (in case of difference constraints), or pushes both jobs to the right (in case of sum constraints).*

Definition: 5.4 *Constraint orientation (Left): A constraint $l \in S_{ij}$ is said to have a* left orientation *if it specifies decreased separation between J_i and J_j (in case of difference constraints), or it pushes both jobs to the left (in case of sum constraints).*

For instance, the constraint $s_1 + e_1 + 4 \leq s_2$ specifies that Job J_2 should start at least 4 units *after* J_1 finishes. Since it specifies increased separation, it is has a right orientation. Likewise, the constraint $s_1 + s_3 \leq 12$ requires that J_1 and J_3 move leftward and hence has a left orientation. Using the flow graph terminology in [SA00a], a forward edge in the constraint graph has a right orientation and a backward edge has a left orientation.

Every constraint $l \in S_{ij}, \forall i, j = 1, \ldots, n$ has an orientation, on account of the total ordering on the job-set. Thus a network constraint between job J_1 and J_5 (say) has the effect of either drawing them together or pushing them apart. *This is not true, if there is no ordering on the job-set.* The total ordering on the start time variables implies that all these variables have an interpretaion on the same real axis $[0, L]$. In the absence of the total order, each variable has to be interpreted on its own axis. Also see [HN94].

Definition: 5.5 *Comparable constraints: Two constraints $l_1, l_2 \in S_{ij}$ are said to be comparable if they have the same orientation and type.*

Note that only constraints between the same set of jobs are comparable, i.e. a constraint $l \in S_{13}$ and a constraint $l' \in S_{12}$ are not comparable, regardless of their orientation and type.

Constraint comparability is an *equivalence relation* partitioning the set of constraints between two jobs S_{ij} into the following four categories:

1. Difference Constraint with left orientation (S_{ij}^1);
2. Difference Constraint with right orientation (S_{ij}^2).
3. Sum Constraint with left orientation (S_{ij}^3);

4. Sum Constraint with right orientation (S^4_{ij});

Definition: 5.6 *Constraint domination: A constraint l_1 is said to dominate another constraint l_2 if and only if they are comparable and $l_1 \Rightarrow l_2$ i.e. l_2 is satisfied, whenever l_1 is satisfied ($l_1, l_2 \in S_{ij}$).*

In some sense, the domination relationship attempts to identify constraints which are redundant. For instance, $s_1 - s_2 \leq -4$ is clearly dominated by $s_1 - s_2 \leq -8$, since if the latter constraint is satisfied, the former is trivially met. The interesting case is the comparison between constraints in which there exist execution time variables. Consider the two comparable constraints:

1. $l_1 : s_1 - s_2 \leq -4$
2. $l_2 : s_1 - s_2 \leq -e_1; e_1 \in [3, 5]$.

Observe that l_2 still dominates l_1; the parametric schedulablity query is: $\exists s_1 \forall e_1 \in [3,5] \exists s_2 \ldots \mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}$? Since the query is true for *all values of e_1 in the range $[3, 5]$, the constraint $s_1 - s_2 \leq -4$ is subsumed by the constraint $s_1 - s_2 \leq -e_1; e_1 \in [3, 5]$.* This holds true for every pair of comparable constraints l_1 and l_2, i.e. either $l_1 \Rightarrow l_2$ or $l_2 \Rightarrow l_1$. *In other words, the domination relationship imposes a total order on each set of comparable constraints between two jobs. We use lexicographical ordering to break ties.* It follows that in each equivalence class of the partition imposed by the comparability relationship there is a unique constraint that dominates all other constraints in that class [Kal86].

Definition: 5.7 *The unique elements $\kappa^k_{ij} \in S^k_{ij}, k = 1, \ldots, 4$, which dominate all the other constraints in their respective partitions are called the* dominators *of that partition.*

Lemma: 5.1 *If a constraint l_1 dominates another constraint l_2, then eliminating l_2 from the set of constraints, does not alter the parametric schedulability of the system i.e. if the constraint system has a parametric schedule with l_2 then it has a parametric schedule without l_2 and vice versa.*

In other words, we can eliminate l_2 from the constraint set and test for parametric schedulability on the reduced set of constraints. In fact, it suffices to retain the 4 dominators between each pair of jobs, since all other constraints are redundant.

Proof: *We provide a proof for the case in which both l_1 and l_2 belong to the set S^1_{ij}. The other three cases, viz. $l_1, l_2 \in S^2_{ij}, l_1, l_2 \in S^3_{ij}$, and $l_1, l_2 \in S^4_{ij}$ can be proved in identical fashion.*

1. *Let the initial system have a parametric schedule* - *We need to prove that the system will continue to be parametrically schedulable, even after l_2 is eliminated. But this is obvious, since we are reducing the number of constraints! (If A_1 is the feasible region, before the removal of l_2 and A_2 is the feasible region after the removal of l_2, then $A_1 \subseteq A_2$ [NW88].)*

2. *Let the initial system be parametrically unschedulable* - *We need to show that the removal of l_2 does not make the system schedulable. From our assumption $l_2 \in S^1_{ij}$, i.e. l_2 is a difference constraint that specifies decreased separation between the jobs J_i and J_j. Since l_1 dominates l_2, the separation specified by l_1 is clearly smaller than the separation specified by l_2. Let us assume the contrary and suppose that the constraint system $\mathbf{A}'$ resulting from the elimination of l_2 is parametrically schedulable. This means that there do not exist start times s_i, s_j and execution times e_i which could depend upon s_i and e_j which could depend upon both s_i and s_j, such that a negative cost loop is created. (The system is infeasible if and only if there is such a negative cost loop [CLR92, Sho81, AS79, Sub00a].) Essentially, we are saying that $\ldots \forall s_i \exists e_i \in [l_i, u_i] \forall s_j \exists e_j \in [l_j, u_j] \ldots \neg(\{\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}\} - \{\mathbf{l_2}\})$ is false. This means that the constraints imposed by the sets S^2_{ij}, S^4_{ij} can co-exist with l_1.Now consider what happens when l_2 is added to $\mathbf{A}'$. l_2 cannot decrease the separation between jobs J_i and J_2 any more than what is specified by l_1. Consequently if a negative loop could not be created using l_1, then such a loop definitely cannot be created using l_2, i.e. the constraint system stays parametrically schedulable, contradicting the assumption that it was not.*

□

This leads us directly to:

Lemma: 5.2 *There are at most* 4 *non-redundant constraints between any* 2 *jobs; hence the total number of non-redundant constraints is at most* $O(n^2)$.

It follows that ELIM-EXIST-VARIABLE() takes at most $O(n^2)$ time, since each start-time variable is part of at most $O(n)$ relationships. PRUNE-CONSTRAINTS() basically performs the equivalent of finding the 4 maxima between each pair of start-times and hence the total time taken is proportional to the number of edges, which is $O(n^2)$. Checking the consistency of the resulting constraints can likewise be carried out in $O(n^2)$ time.

Since all the above functions are called at most $O(n)$ times, the above analysis leads to the following conclusion.

Theorem: 5.1 *Algorithm (5.1) can be implemented to run in* $O(n^3)$ *worst-case time.*

Proof: *Follows from the discussion above.* □

6 Online Dispatching

Deciding the parametric schedulability of a given constraint system, is only half the problem. Once it has been decided that the task system has a parametric schedule, it is the function of the dispatcher to compute a valid start time for each job, depending upon the exeuction times of tasks that preceded it. We briefly discuss two strategies for parametric dispatching.

1. Constraint relaxation - The basic idea is to maintain dependency lists for each job, identifying the set of execution times that its start time depends on. When these times are available, they are plugged in and the start time of the current job is computed. This takes at most $O(m)$ time online.

2. Reduction - The parametric schedulability predicate and the dispatching problem can be viewed as the decision and functional counterparts of the same problem. Expressed in strict alternating quantifier form, the problem represents a $2-$person game in which player A outputs start-times, while player B outputs execution times [Pap94, Sub00b]. Since A has a valid move, for every move that B makes, computing the start-time of a job, given the execution times of jobs that completed before it, can be viewed as a `tt-reducibility` problem [DK00]. Using the results from [Sub00a], we know that the start-time computation can be carried out in $O(n^3.\log L)$ time.

7 Conclusions

In this paper, we extended existing polynomial time algorithms for deciding parametric schedulability to include arbitrary network constraints. Network constraints in general and aggregate constraints in particular find application in modeling and approximating performance metrics, such as weighted sum of completion times. We are presently engaged in implementing static and parametric schedulers within the scheduler/allocator block of the Maruti Operating System [STA00].

From a theoretical perspective, we note that existing analyses for the parametric schedulability problem provide worst-case guarantees of $O(n^3)$, independent of the number of constraints. We suspect that our analysis could be improved to provide a worst-case guarantee of $O(m.n)$ time, thereby explicitly accounting for the number of constraints. Finally, we emphasize that our approach yielded a polynomial time bound, because we were able to exploit the ordering information on the jobs. It would be instructive to obtain either polynomial time algorithms or a proof of hardness, in the absence of the total order.

References

[AB98] Alia Atlas and A. Bestavros. Design and implementation of statistical rate monotonic scheduling in kurt linux. In *Proceedings IEEE Real-Time Systems Symposium*, December 1998.

[AB99] Alia Atlas and A. Bestavros. Multiplexing vbr traffic flows with guaranteed application level qos using statistical rate monotonic scheduling. October 1999.

[All] Eric Allender. Personal Communication.

[AS79] Bengt Aspvall and Yossi Shiloach. A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. In *20th Annual Symposium on Foundations of Computer Science*, pages 205–217, San Juan, Puerto Rico, 29–31 October 1979. IEEE.

[Bru81] P. Brucker. *Scheduling.* Akademische Verlagsgesellschaft, Wiesbaden, 1981.

[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.

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

[DK00] Ding-Zhu Du and Ker-I Ko, editors. *Theory of Computational Complexity.* John Wiley and Sons, 2000.

[DLK82] M.A.H. Dempster, J.K. Lenstra, and A.H.G. Rinooy Kan, editors. Reidel, Dordrecht, 1982.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[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.

[Joh] D.S. Johnson. Personal Communication.

[Kal86] Kenneth Kalmanson. *An Introduction to Discrete Mathematics and its Applications.* Addison–Wesley, 1986.

[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.

[MR] R.H. Mohring and F.J. Radermacher. *Lecture Notes in Economics and Mathematical Systems.* Number 240. Springer-Verlag, Berlin.

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

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

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

[SA00a] K. Subramani and A. K. Agrawala. A dual interpretation of standard constraints in parametric scheduling. In *The Sixth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, September 2000.

[SA00b] 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.

[Sho81] Robert Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, October 1981.

[SR88] J. A. Stankovic and K. Ramamritham. *Hard Real-Time Systems.* IEEE Computer Society Press, Los Alamitos, 1988.

[SS] M. Shaked and G. Shanthikumar, editors. *Stochastic Scheduling.* Academic Press, San Diego.

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

[STA00] K. Subramani, Bao Trinh, and A. K. Agrawala. Implementation of static and parametric schedulers in maruti. *Manuscript in Preparation*, March 2000.

[Sub00a] K. Subramani. *Duality in the Parametric Polytope and its Applications to a Scheduling Problem.* PhD thesis, University of Maryland, College Park, July 2000.

[Sub00b] K. Subramani. Parametric schedulability with arbitrary constraint sets is not harder than `pspace`. *Submitted to IPL*, September 2000.

[VR99] V.Chandru and M.R. Rao. Linear programming. In *Algorithms and Theory of Computation Handbook, CRC Press, 1999.* CRC Press, 1999.