PARTIALLY CLAIRVOYANT SCHEDULING FOR AGGREGATE CONSTRAINTS

K. SUBRAMANI

Received 10 February 2004

The problem of partially clairvoyant scheduling is concerned with checking whether an ordered set of jobs, having nonconstant execution times and subject to a collection of imposed constraints, has a partially clairvoyant schedule. Variability of execution times of jobs and nontrivial relationships constraining their executions, are typical features of real-time systems. A partially clairvoyant scheduler *parameterizes* the schedule, in that the start time of a job in a sequence can depend upon the execution times of jobs that precede it, in the sequence. 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 partially clairvoyant schedules can be determined in polynomial time, when the constraints are restricted to be "standard," that is, relative timing constraints. In this paper, we extend the class of constraints for which partially clairvoyant schedules can be determined efficiently, to include *aggregate constraints*. Aggregate constraints form a strict superset of standard constraints and can be used to model performance metrics.

1. Introduction

Variability in the execution times of jobs is a common characteristic in real-time systems. There are a number of reasons for this feature, including input-dependent loops and measuring error [2, 3, 29, 30]. For instance, the running time of a sorting subroutine in a program, depends on the size of the input set, and in general, will be greater for an input set of size 1000, than for an input set of size 10. In the literature, there exist two broad approaches for addressing execution time variability, 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 for the most likely values of execution times. Such a guarantee requires that the execution times belong to a fixed, well-understood distribution [11, 23, 26] and is tempered with the knowledge that there is a finite, nonzero probability that the constraints may be violated at run time. This approach is not applicable in the case of "hard" real-time systems [19, 24, 29], where the guarantees have

Journal of Applied Mathematics and Decision Sciences 2005:4 (2005) 225–240
DOI: 10.1155/JAMDS.2005.225

to be absolute and there is no room for error. Hard real-time systems are typically composed of mission-critical jobs, wherein the consequences of failure, that is, a violation of the imposed constraints, can be catastrophic. Such real-time systems therefore call for deterministic approaches to the issue of uncertainty. Some of the common approaches include *worst-case* (largest value) assumptions [4, 23], Zero clairvoyant scheduling [31] and partially clairvoyant scheduling [6, 13, 32]. Assume that a job J_1 has execution time e_1 in the range $[3,10]$ and it is required that the job finish at or after time $t = 10$, that is, $c_1 : s_1 + e_1 \geq 10$. If we take the largest value of e_1 to satisfy this constraint, then we get $s_1 = 0$ as a possible solution. However, during the actual execution of J_1, e_1 could be 3, in which case the constraint c_1 is broken. Thus, we see that worst-case assumptions regarding execution times, run the risk of constraint violation at *run time* and hence such a strategy is not always correct.

In Zero clairvoyant scheduling [31], we 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. In the above example, a Zero clairvoyant scheduler would substitute $e_1 = 3$ in c_1 to get $s_1 \geq 7$. Observe that this assignment ensures that c_1 is not broken at run time. However, Zero clairvoyant scheduling is extremely inflexible and even simple constraint sets may not have Zero clairvoyant schedules. For instance, consider a pair of jobs $\{J_1, J_2\}$ with start times $\{s_1, s_2\}$ and execution times $\{e_1 \in [3,5],\ e_2 \in [2,4]\}$, with the following two constraints imposed on their execution: $\{c_1 : s_1 + e_1 \leq s_2,\ c_2 : s_2 \leq s_1 + e_1 + 1\}$.

Note that the conservative Zero clairvoyant approach forces the following system to be satisfied

$$s_1 + 5 \leq s_2 \quad s_2 \leq s_1 + 4. \tag{1.1}$$

System (1.1) is clearly infeasible and hence there does not exist a Zero clairvoyant schedule.

On the other hand, consider the schedule described by system (1.2).

$$s_1 = 0 \quad s_2 = s_1 + e_1. \tag{1.2}$$

It is a valid, albeit parametrized schedule; in particular, s_2 depends on the actual value assumed by e_1 during run time.

We see that partially clairvoyant scheduling increases the flexibility of a scheduler, in that a constraint system which cannot be scheduled by a Zero clairvoyant scheduler, may be scheduled by a partially clairvoyant scheduler.

In the real-time scheduling literature, it has been shown that polynomial time algorithms exist for determining the existence of partially clairvoyant schedules, when the constraints are restricted to be "standard" [7, 13]. In this paper, we are concerned with the following question: *can the existence of partially clairvoyant schedules in a constraint system be determined efficiently for nonstandard constraints, with at most two jobs per constraint?* We provide an affirmative answer to the above question by designing a polynomial time algorithm for a class of constraints, termed as *aggregate 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. A quantifier elimination based algorithm for deciding the partially clairvoyant schedulability of a constraint system is presented in Section 5, along with the proof of its correctness and an analysis of its running time. We conclude in Section 6, by summarizing our results in this paper and outlining problems for future research.

2. Statement of problem

In this section, we provide a formal description of the problem under consideration. The specification of a real-time scheduling problem entails the specification of the type of jobs involved, the nature of the constraints that are imposed on them and the type of clairvoyance afforded to the scheduler [30]. This paper is concerned with the partially clairvoyant schedulability of an ordered set of nonpreemptive jobs, subject to a set of aggregate constraints.

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 nonpreemptive, ordered jobs, $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$; these jobs execute in each scheduling window. The jobs have start times $\{s_1, s_2, \ldots, s_n\}$ and execution times $\{e_1, e_2, \ldots, e_n\}$, with (s_i, e_i) denoting the start and execution times of job J_i, respectively. e_i is a range-bound variable, in that during actual execution, it may assume any value in the nonempty range $[l_i, u_i]$. In real-time systems such as Maruti, a number of runs of the job set are carried out to statistically estimate the range $[l_i, u_i]$ for job J_i, with a high degree of confidence [14].

2.2. Constraint model. The constraints on the jobs are described by system (2.1):

$$\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}, \quad \vec{\mathbf{e}} \in \mathbf{E}, \tag{2.1}$$

where,

(i) $\mathbf{A}$ is an $m \times 2 \cdot n$ rational matrix, in which every row represents an aggregate constraint (to be defined below),

(ii) $\mathbf{E}$ is the axis-parallel hyper-rectangle (aph)

$$[l_1, u_1] \times [l_2, u_2] \times \cdots \times [l_n, u_n]. \tag{2.2}$$

(iii) $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$ is the start time vector of the jobs, and

(iv) $\vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E}$ 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*.

Observe that system (2.1) can be rewritten in the form

$$\mathbf{G}\cdot\vec{\mathbf{s}}+\mathbf{H}\cdot\vec{\mathbf{e}}\leq\vec{\mathbf{b}},\quad \vec{\mathbf{e}}\in\mathbf{E}. \tag{2.3}$$

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

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, whereas a constraint of the form $s_i + s_j \geq 9$ is *not* a standard constraint. A detailed description of standard constraints is available in [13]. Standard constraints can be represented by edges of a flow network [10]; these constraints are also called difference constraints and relative timing constraints.

Definition 2.2. Aggregate Constraint—A constraint is said to be an aggregate constraint, if it is a standard constraint *or* it can be expressed in one of the following forms:

(1) $s_i + s_j\{\leq,\geq\}k$,
(2) $(s_i + e_i) + s_j\{\leq,\geq\}k$,
(3) $s_i + (s_j + e_j)\{\leq,\geq\}k$,
(4) $(s_i + e_i) + (s_j + e_j)\{\leq,\geq\}k$.

For instance, the requirement that the sum of the finish times of J_1 and J_2 should not exceed 14, can be modeled by the aggregate constraint $(s_1 + e_1) + (s_2 + e_2) \leq 14$. Aggregate constraints also have a graph structure; the "edge" representing the set of constraints between two jobs J_i and J_j, forms a polyhedron in the four variables s_i, e_i, s_j, e_j with e_i and e_j being universally quantified [1, 15]. It is clear that aggregate constraints are a strict superset of standard constraints.

2.3. Query model. Given the sequence in which the jobs execute, that is, $J_1 < J_2 < \cdots < J_n$, a partially clairvoyant scheduler decides on a value for s_1, so that J_1 can be scheduled. When J_1 finishes execution, the scheduler *knows* the values of e_1; it then uses the value of e_1 to compute s_2; the values of e_1 and e_2 are then used to compute s_3 and this process continues till s_n is decided. The constraints must be respected by the s_i values that are chosen. The goal of the scheduler is to check *in advance* whether such an assignment is possible.

Accordingly, the schedulability query can be described as:

$$\exists s_1\ \forall e_1\in[l_1,u_1]\ \exists s_2\ \forall e_2\in[l_2\cdot u_2],\ldots\exists s_n\ \forall e_n\in[l_n,u_n]\quad \mathbf{A}\cdot[\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathrm{T}}\leq\vec{\mathbf{b}}? \tag{2.4}$$

We note that the "solution" to query (2.4) is not a numeric vector in general, but a vector of Skolemized functions, capturing the dependence of s_i on $\{e_1, e_2, \ldots, e_{i-1}\}$.

It is convenient to think of partially clairvoyant schedulability as an n round, 2-person game. In round i, the start time player guesses a value for s_i, while the execution time player guesses a value for e_i in the range $[l_i, u_i]$. Let $\vec{\mathbf{s}'}$ and $\vec{\mathbf{e}'}$ be the n-vectors guessed

by the 2 players, at the end of n rounds. If $\mathbf{A} \cdot [\vec{\mathbf{s}'} \ \vec{\mathbf{e}'}]^{\mathrm{T}} \leq \vec{\mathbf{b}}$, then the game is a win for the start time player; otherwise it represents a win for the execution time player. The guesses are nondeterministic, in that if there is a strategy for the start time player to win, he will win; similarly if the execution time player has a winning strategy, he will win. Thus, deciding query (2.4), corresponds to checking whether the start time player has a winning strategy. In the succeeding sections, we will argue that if the constraints are restricted to be aggregate constraints and if a winning strategy exists for the start time player, then such a strategy has polynomial size and can be determined in polynomial time.

The combination of the Job model, Constraint model, and the Query model constitutes a scheduling problem specification within the E-T-C scheduling framework [30].

3. Motivation

In Section 1, we argued that partially clairvoyant schedules are more flexible than Zero-clairvoyant schedules, in that constraint sets which are not schedulable by a Zero-clairvoyant scheduler may be scheduled by a partially clairvoyant scheduler. The research upto this point has focussed on "standard" constraints only and established the existence of a polynomial time strategy that determines the existence of a partially clairvoyant schedule. An interesting line of research is obtaining schedules that satisfy certain optimization criteria. The complexity of Partially Clairvoyant Optimization for general constraints, is not known [17]. However, we can approximate optimization functions involving at most two jobs through the use of aggregate constraints. Optimization criteria, formulated as *performance metrics* arise in various situations including Job-shop, Flow-shop and Machine-Shop [4, 23]. Typical performance metrics are *Makespan*, *Sum of Start times* and *Sum of Completion times*. For instance, the need to minimize the sum of completion times of jobs J_1 and J_2 can be approximated through $(s_1 + e_1) + (s_2 + e_2) \leq k$, for suitably chosen k [28, 29]. Likewise the Makespan performance metric can be modeled through $s_n \leq k$. (Recall that job J_n finishes last in the sequence.)

4. Related work

Applications in which partially clairvoyant scheduling plays a role, arise in two, somewhat unrelated fields, viz., Operating Systems scheduling and AI planning.

From the Operating Systems perspective, the concept of partially clairvoyant scheduling was introduced in [24] (where it was called parametric scheduling). In [13], polynomial time algorithms were presented for the case, when the constraints imposed on the job-set are "standard." [32] argued that "standard" constraints could be represented by a flow, graph. [6, 7] extend the "standard constraint" model to include the case, in which constraints can exist between adjacent scheduling windows.

In AI planning, dealing with uncertainty in problem parameters is a fundamental issue, as evidenced by [10, 20, 21, 22, 33, 34]. In a typical AI planner, timing constraints are used to ensure temporal consistency of plans. We will now briefly describe the various types of constraint networks and their relationship to each other, in terms of expressibility.

The temporal constraint network structure was introduced in [10]. In this structure, the only constraints that are permitted are those of the form $x_i - x_j \leq [a,b]$, $a < b$, where a and b are fixed numbers and x_i and x_j are points in time, which are controlled by the agent. Note that this constraint can be expressed by the pair of relationships (conjunction): $x_i - x_j \leq b$, $x_j - x_i \leq -a$. If there is precisely one constraint between every pair of time points, then the resultant constraint system is a conjunction of difference constraints, which can be solved in polynomial time, using a Bellman-Ford type propagation algorithm [8]. Such a system is also called a Simple Temporal Constraint Network, since such a system can be represented as a directed graph with the interval $[a,b]$ representing two opposite arcs or links [10].

As was pointed out in [34], simple temporal constraint networks are not adequate to model real-world situations, in which certain links may not be under the control of the agent. For instance, a given interval constraint may represent the time taken for a job to complete execution and it may vary in a range, depending upon external factors. Accordingly, the simple temporal constraint networks with uncertainty (STNU) framework was proposed in [22, 34]. In this framework, some of the links (constraints) are *free*, while the rest of the links are contingent. The duration of a free link is chosen by the agent, whereas the duration of a contingent link is chosen by the environment. A contingent link always relates two strictly ordered time points, with the latter point being determined by the environment.

Our notion of partially clairvoyant schedulability corresponds to their notion of *Dynamic Controllability*, in that the plans change on the fly, depending on the outcome of previous events; however there exist the following important differences.

(1) We explicitly accommodate *aggregate* or *sum* constraints. These constraints cannot be represented in Constraint Networks, since by definition, the nodes of the network represent points in time and the links represent constraints on their difference. Accordingly, our constraint framework is a *strict* superset of the constraint framework in [22].

(2) In [22, 34], the condition that two contingent arcs cannot have the same finish time, is explicitly enforced. In our case, this condition is implicitly enforced as follows: We have nonzero execution times for each job and there is a total ordering on the job execution sequence.

(3) Consider a constraint of the following form: the finish time of J_2 is at most 5 units, from the finish time of J_1. This constraint can be easily represented in our framework as: $s_2 + e_2 \leq s_1 + e_1 + 5$. However, this constraint cannot be represented in the framework of [22], since the link from s_1 to s_2 depends on both e_1 and e_2. While s_2 can legitimately depend upon the value of e_1, it cannot depend upon the value of e_2. Thus, there is no way to designate this link as a contingent link in their framework, since the value of this link can be determined completely, only *after* s_2 has been assigned.

(4) In the AI applications described in [20], constraints are of the form $s_1 + e_1 - s_2 \in [-5,5]$, that is, the environment decides what value in $[-5,5]$ is actually assumed by the link. It is not clear that this constraint can be put within our scheduling framework.

(5) It is also important to note that the algorithm in [22], to determine dynamic controllability bears no resemblance whatsoever, to our quantifier elimination approach. Indeed, they have used *Triangular Reductions* as the basis of their approach, to determine

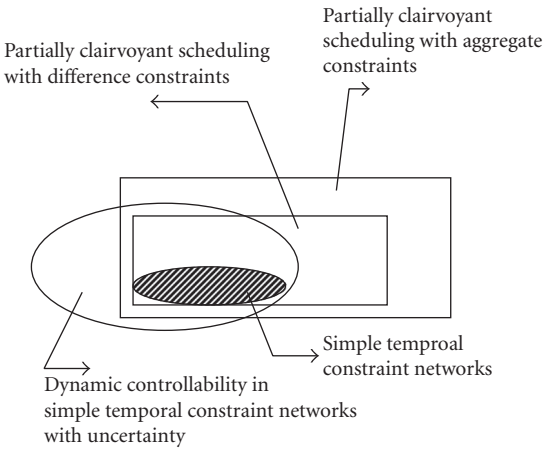

Figure 4.1. Scheduling with uncertainty.

dynamic controllability, whereas quantifier elimination over conjunctions of aggregate constraints forms the basis for our algorithm.

Figure 4.1 shows how uncertainty has been incorporated into various scheduling frameworks and the relationships among them.

In both [7] and [13], polynomial time bounds are derived by observing that the number of standard constraints between any two jobs is bounded, if only strict relative constraints are permitted (in fact, at most 8 nonredundant constraints can exist). In this paper, we develop two new concepts viz. *Constraint Domination* and *Constraint Orientation*, which in turn are used to develop polynomial time algorithms for testing partially clairvoyant schedulability in the presence of aggregate constraints.

Linear programs with at most 2 variables per constraint (LI(2)s) have received quite a bit of attention in the Operations Research community. [27] was the first to observe the correspondence between LI(2)s and graphs; [1] gave the first polynomial time algorithm for this problem. In [15], the Fourier-Motzkin elimination procedure was used to provide a *strongly polynomial* algorithm, which to date is the fastest known algorithm for this problem. Our constraints are similar to those found in LI(2)s; however the coefficient of a nonzero variable is either 1 or -1 in aggregate constraints.

5. Algorithms and complexity

Algorithm 5.1 presents our strategy for testing partially clairvoyant schedulability. The strategy is based on quantifier elimination over a conjunction of linear constraints. To start with, the innermost quantified variable (e_n) in specification (2.4) is eliminated, followed by the next variable and so on. The quantifier elimination process is solution preserving, that is, if the constraint system has a solution prior to the elimination then the constraint system that results after the elimination, also has a solution and vice versa. The elimination process reduces a $2 \cdot n$ variable constraint system to a constraint system

```
Function PART-CLAIR-SCHED (E, A, b⃗)
  (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 ≤ s_1 ≤ b, a, b ≥ 0)
  (14)   Valid partially clairvoyant schedule exists.
  (15)   return
  (16) else
  (17)   A partially clairvoyant schedule does not exist.
  (18)   return
  (19) end if
```

Algorithm 5.1. A quantifier elimination algorithm for determining partially clairvoyant schedulability.

in one variable, viz., s_1. If it is possible to satisfy this trivial constraint system, then the original constraint system is satisfiable. As the quantifier string unwinds, that is, variables are eliminated, it is possible that inconsistencies and redundancies occur; the CHECK-INCONSISTENCY() procedure handles inconsistencies, while the PRUNE-CONSTRAINTS() procedure eliminates redundancies. The following types of inconsistencies and redundancies could occur:

(1) A constraint $s_i \le a$ and another constraint $s_i \ge b$, where $b \ge a + 1$—In this case the system is declared infeasible.
(2) A constraint of the form $e_i \le g$—If $g \ge u_i$, we declare the constraint redundant; otherwise the constraint system is declared infeasible.
(3) A constraint of the form $e_i \ge r$—If $r \le l_i$, we declare the constraint redundant; otherwise the constraint system is declared infeasible.

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 [5] 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. A brief description of the Fourier-Motzkin procedure is available in the appendix.

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 $e_i \in [l_i, u_i]$.

5.1. Proof of correctness. Algorithm 5.1 is basically a quantifier unrolling algorithm; it eliminates one quantifier at a time, while preserving the solution space; we provide an inductive proof sketch of its correctness. Observe that Algorithm 5.1 works correctly, when there is only one job, that is, the base case is trivial. Assume that Algorithm 5.1 works correctly, when there are at most $(n-1)$ jobs in the constraint system. Now consider the case in which the algorithm is presented with a constraint system on n jobs.

We focus on the elimination of the nth execution time variable e_n; it is the first variable to be eliminated.

Let $\mathbf{A} \cdot [\vec{\mathbf{s}'}\ \vec{\mathbf{e}'}]^{\mathbf{T}} \leq \vec{\mathbf{b}'}$ be the system that results after calling Elim-Univ-Variable(e_n), where $\vec{\mathbf{s}'} = [s_1, s_2, \ldots, s_n]^T$ and $\vec{\mathbf{e}'} = [e_1, e_2, \ldots, e_{n-1}]^T$. We need to show that

$$\exists s_1\ \forall e_1 \in [l_1, u_1]\ \exists s_2 \forall e_2 \in [l_2, u_2], \ldots \exists s_n\ \forall e_n \in [l_n, u_n] \quad \mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}$$

$$\iff \exists s_1\ \forall e_1 \in [l_1, u_1]\ \exists s_2\ \forall e_2 \in [l_2, u_2],\ \forall e_{n-1} \in [l_{n-1}, u_{n-1}] \cdots \exists s_n \quad \mathbf{A}' \cdot \left[\vec{\mathbf{s}'}\ \vec{\mathbf{e}'}\right]^{\mathbf{T}} \leq \vec{\mathbf{b}'}. \tag{5.1}$$

We focus on those constraints which contain e_n; the constraints which do not contain e_n occur identically in both systems. Consider a constraint of the form $g_1 : e_n \leq f()$ in $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}$.

Note that a solution to system $\mathbf{A}' \cdot [\vec{\mathbf{s}'}\ \vec{\mathbf{e}'}]^{\mathbf{T}} \leq \vec{\mathbf{b}'}$ (with the quantifier specification) must satisfy the constraint $u_n \leq f()$. This solution clearly satisfies g_1 since during execution the maximum value that e_n may take is u_n. Likewise, any solution to the system $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}$ must ensure that the constraint $u_n \leq f()$ is satisfied; if such is not the case, then e_n can take the value u_n at run time and break the constraint g_1. In essence, we are taking the intersection of the polyhedron $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}$ with the polyhedron $e_n \leq u_n$, $e_n \geq l_n$. Preservation of the solution space using Fourier-Motzkin elimination has been argued in [9, 25]. We have thus shown that Algorithm 5.1 eliminates e_n and s_n while preserving the solution space; after the elimination of these two variables, we are left with a constraint system on $(n-1)$ jobs. It follows by the inductive hypothesis that Algorithm 5.1 is correct.

5.2. 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 quadratically, each time it is called. 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 [25]. In [6, 13], it was pointed out that "standard" constraints are closed under Fourier-Motzkin elimination, that is, the elimination of an existential variable results in the set of constraints staying standard. Using the notation, in [32], 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; indeed Algorithm 5.1 runs in time $O(n^3)$ on a constraint set on n jobs.

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 aggregate constraints, as discussed in Section 3. We make the following observation.

Observation 5.1. *Aggregate constraints are closed under Fourier-Motzkin elimination, using Algorithm 5.1.*

Observation 5.1 follows from the fact an existentially quantified variable, that is, 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 aggregate constraint 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 jobs J_1 and J_2 to the left, that is, towards 0, while the constraint $s_3 + s_4 \geq 8$ pushes jobs J_3 and J_4 towards the right, that is, 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 5.2. 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 aggregate constraints).

Definition 5.3. 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 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 [32], 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 an aggregate 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 interpretation on the same real axis $[0, L]$. In the absence of the total order, each variable has to be interpreted on its own axis (see [15]).

Definition 5.4. Comparable constraints: Let c_1 and c_2 denote two constraints of the constraint system $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathrm{T}} \leq \vec{\mathbf{b}}$ that express relationships between J_i and J_j. These constraints are said to be comparable if and only if they have the same orientation and type and additionally the coefficients of s_i, s_j, e_i, and e_j are identical in both constraints,

For instance, $s_1 + e_1 + 4 \leq s_2$ and $s_1 + 8 \leq s_2$ are not comparable, since the coefficient of e_1 is 1 in the first constraint and 0 in the second constraint.

Note that only constraints between the same set of jobs are comparable, that is, 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 sixteen categories as follows, characterized by the following features:

(1) left oriented or right oriented (L or R),
(2) sum or difference (S or D),
(3) e_i present or absent, (i_1 or i_0),
(4) e_j present or absent, (j_1 or j_0).

We use $S_{ij}^{LSi_1j_0}$ to denote constraints defined between J_i and J_j that are left-oriented, sum, with e_i present and e_j absent; the other 15 equivalence classes are described similarly.

Definition 5.5. Constraint domination. A constraint c_1 is said to dominate another constraint c_2, if and only if they are comparable and $c_1 \Rightarrow c_2$, that is, c_2 is satisfied, whenever c_1 is satisfied.

In some sense, the domination relationship attempts to identify constraints that 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: Observe that c_2 still dominates c_1; the schedulability query is: $\exists s_1\ \forall e_1 \in [3,5]\ \exists s_2 \cdots \mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}$? Since the query is true for *all values of* e_1 *in the range* $[3,5]$, *the constraint* $s_1 - s_2 \leq -e_1$ *is subsumed by the constraint* $s_1 - s_2 \leq -e_1 - 4$; $e_1 \in [3,5]$. This holds true for every pair of comparable constraints c_1 and c_2, that is, either $c_1 \Rightarrow c_2$ or $c_2 \Rightarrow c_1$. *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 exists a unique constraint that dominates all other constraints in that class.

Definition 5.6. The unique elements which dominate all the other constraints in their respective partitions are called the *dominators* of that partition.

Theorem 5.7. *Let the constraint system contain two constraints* c_1 *and* c_2, *such that* c_1 *dominates* c_2. *Eliminating* c_2 *from the set of constraints, does not alter the partially clairvoyant schedulability of the system, that is, if the constraint system has a partially clairvoyant schedule with* c_2, *then it has such a schedule without* c_2; *likewise, if it does not have a partially clairvoyant schedule, in the absence of* c_2, *then it does not have one, in the presence of* c_2 *either.*

In other words, we can eliminate c_2 from the constraint set and test for partially clairvoyant schedulability on the reduced set of constraints. In fact, it suffices to retain the 16 dominators between each pair of jobs, since all other constraints are redundant.

Proof. We provide a proof for the case in which both c_1 and c_2 belong to the set $S_{ij}^{\mathrm{LS}\,i_1 j_1}$. The other fifteen cases can be argued in similar fashion.

(1) Assume that the system containing both c_1 and c_2 has a partially clairvoyant schedule. We need to prove that the system will continue to have such a schedule after c_2 is eliminated. But this is obvious, since we are reducing the number of constraints; indeed any solution to the old system (with c_2 present) is also a solution to the new system (with c_2 eliminated).

(2) Consider the case in which the system containing both constraints does not have a partially clairvoyant schedule. We need to show that the removal of c_2 does not make the system schedulable. Without loss of generality, we assume that c_1 is $(s_i + e_i) + (s_j + e_j) \leq k_1$ and c_2 is $(s_i + e_i) + (s_j + e_j) \leq k_2$, where $k_1 \leq k_2$, since c_1 dominates c_2. Let us assume that the system has a partially clairvoyant schedule, when c_2 is removed. Now consider the n round 2-person game played by the start time and execution time players. At the end of n rounds, constraint c_1 must be met; accordingly, $(s_i' + e_i') + (s_j' + e_j') \leq k_1$, which means that $(s_i' + e_i') + (s_j' + e_j') \leq k_2$ and hence the start time player could have won the game, even in the presence of c_2, thereby contradicting the hypothesis that the system did not have a partially clairvoyant schedule, in the presence of both c_1 and c_2. □

This leads us directly to the following theorem.

Theorem 5.8. *There are at most* 16 *nonredundant constraints between any* 2 *jobs; hence the total number of nonredundant 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 16 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 5.9. *Algorithm 5.1 can be implemented to run in $O(n^3)$ worst-case time.*

Proof. Follows from the discussion above. □

6. Conclusions

In this paper, we extended an existing polynomial time algorithm for deciding partially clairvoyant schedulability, to include aggregate constraints. Aggregate constraints find applications in modeling and approximating performance metrics, such as sum of completion times. One of our current projects is to implement a partially clairvoyant scheduler within the frameworks of existing real-time operating systems such as Maruti.

From a theoretical perspective, we note that existing analyses for the partially 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 $O(m \cdot n)$ time, thereby explicitly accounting for the number of constraints.

Appendix

Fourier-Motzkin elimination

The Fourier-Motzkin elimination procedure is an elegant, syntactic, variable elimination scheme to solve constraint systems that are comprised of linear inequalities. It was discovered initially by Fourier [12] and later by Motzkin [9], who used it to solve general purpose linear programs.

The key idea in the elimination procedure is that a constraint system in n variables (i.e., $\Re^n$), can be projected onto a space of $n-1$ variables (i.e., $\Re^{n-1}$), without altering the solution space. In other words, polyhedral projection of a constraint set is solution preserving. This idea is applied recursively, till we are left with a single variable (say x_1). If we have $a \leq x_1 \leq b$, $a \leq b$, then the system is consistent, for any value of x_1 in the interval $[a,b]$. Working backwards, we can deduce the values of all the variables $x_2,\ldots,x_n$. If $a > b$, we conclude that the system is infeasible.

Algorithm A.1 is a formal description of the above procedure.

Though elegant, this syntactic procedure suffers from an exponential growth in the constraint set, as it progresses. This growth has been observed both in theory [25] and in practice [16, 18]. By appropriately choosing the constraint matrix **A**, it can be shown that eliminating k variables causes the size of the constraint set to increase from m to $O(m^{2^k})$ [25]. Algorithm A.1 remains useful though as a tool for proving theorems on polyhedral spaces [5]. [25] gives a detailed exposition of this procedure.

```
Function Fourier-Motzkin elimination (A, b⃗)
(1) for (i = n down to 2)
(2)    Let I+ = {set of constraints that can be written in the form x_i ≥ ()}
(3)    Let I− = {set of constraints that can be written in the form x_i ≤ ()}
(4)    for (each constraint k ∈ I+) do
(5)     for (each constraint l ∈ I−) do
(6)       Add k ≤ l to the original constraints
(7)     end for
(8)    end for
(9)    Delete all constraints containing x_i
(10) end for
(11) if (a ≤ x_1 ≤ b, a ≤ b) then
(12)    Linear program is consistent
(13)    return
(14) else
(15)    Linear program is inconsistent
(16)    return
(17) end if
```

Algorithm A.1. The Fourier-Motzkin elimination procedure.

Acknowledgment

We are grateful to the anonymous referee who pointed out [20], thereby enabling us to present our work within the framework of the AI community, in addition to the OR community.

References

[1] B. Aspvall and Y. Shiloach, *A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality*, Proc. 20th Annual Symposium on Foundations of Computer Science (Puerto Rico, 1979), IEEE Computer Society, California, 1979, pp. 205–217.

[2] A. Atlas and A. Bestavros, *Design and implementation of statistical rate monotonic scheduling in KURT Linux*, Proc. IEEE Real-Time Systems Symposium (Arizona, 1999), IEEE Computer Society, California, 1999, pp. 272–276.

[3] ———, *Multiplexing VBR traffic flows with guaranteed application-level QoS using statistical rate monotonic scheduling*, Proc. 8th International Conference on Computer Communications and Networks (Massachusetts, 1999), IEEE Computer Society, California, 1999, pp. 282–287.

[4] P. Brucker, *Scheduling Algorithms*, 2nd ed., Springer, Berlin, 1998.

[5] V. Chandru and M. R. Rao, *Linear programming*, Algorithms and Theory of Computation Handbook, CRC Press, Florida, 1999, pp. 31-1–31-37.

[6] S. Choi, *Dynamic time-based scheduling for hard real-time systems*, Ph.D. thesis, University of Maryland, Maryland, 1997.

[7] ———, *Dynamic time-based scheduling for hard real-time systems*, Journal of Real-Time Systems **19** (2000), no. 1, 5–40.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 2nd ed., MIT Press, Massachusetts; McGraw-Hill, New York, 1992.

[9] G. B. Dantzig and B. C. Eaves, *Fourier-Motzkin elimination and its dual*, J. Combinatorial Theory Ser. A **14** (1973), 288–297.

[10] R. Dechter, I. Meiri, and J. Pearl, *Temporal constraint networks*, Artificial Intelligence **49** (1991), no. 1-3, 61–95.

[11] M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan (eds.), *Deterministic and Stochastic Scheduling*, Proc. NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems (Durham, 1981), NATO Advanced Study Institute Series C: Mathematical and Physical Sciences, vol. 84, D. Reidel, Dordrecht, 1982.

[12] J. B. J. Fourier, *Reported in: Analyse de travaux de l'Academie Royale des Sciences, pendant l'annee 1824, Partie Mathematique, Historyde l'Academie Royale de Sciences de l'Institue de France 7 (1827) xlvii-lv. (Partial English translation in: D. A. Kohler, Translation of a Report by Fourier on his work on Linear Inequalities. Opsearch 10 (1973) 38-42.)*, Academic Press, 1824.

[13] R. Gerber, W. Pugh, and M. Saksena, *Parametric dispatching of hard real-time tasks*, IEEE Trans. Computers **44** (1995), no. 3, 471–479.

[14] O. Gudmundsson, D. Mosse, A. K. Agrawala, and S. K. Tripathi, *MARUTI: a hard real-time operating system*, Proc. IEEE Workshop on Experimental Distributed Systems (Alabama, 1990), IEEE Computer Society, California, 1990, pp. 29–34.

[15] D. S. Hochbaum and J. Naor, *Simple and fast algorithms for linear and integer programs with two variables per inequality*, SIAM J. Comput. **23** (1994), no. 6, 1179–1192.

[16] T. Huynh, C. Lassez, and J.-L. Lassez, *Fourier algorithm revisited*, Algebraic and logic programming (Nancy, 1990), Lecture Notes in Comput. Sci., vol. 463, Springer, Berlin, 1990, pp. 117–131.

[17] D. S. Johnson, personal communication.

[18] J.-L. Lassez and M. J. Maher, *On Fourier's algorithm for linear arithmetic constraints*, J. Automat. Reason. **9** (1992), no. 3, 373–379.

[19] S. T. Levi, S. K. Tripathi, S. D. Carson, and A. K. Agrawala, *The MARUTI hard real-time operating system*, Operating Systems Review **23** (1989), no. 3, 90–105.

[20] P. H. Morris, N. Muscettola, and T. Vidal, *Dynamic control of plans with temporal uncertainty*, Proc. 17th International Joint Conference on Artificial Intelligence (Washington, 2001), 2001, pp. 494–502.

[21] N. Muscettola, P. H. Morris, B. Pell, and B. Smith, *Issues in temporal reasoning for autonomous control systems*, Proc. 2nd International Conference on Autonomous Agents (Minnesota, 1998), 1998, pp. 362–368.

[22] N. Muscettola, B. Smith, S. Chien, C. Fry, G. Rabideau, K. Rajan, and D. Yan, *In-board planning for autonomous spacecraft*, Proc. 4th International Symposium on Artificial Intelligence, Robotics, and Automation for Space (Tokyo, 1997), 1997.

[23] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice-Hall, New Jersey, 1995.

[24] M. Saksena, *Parametric scheduling in hard real-time systems*, Ph.D. thesis, University of Maryland, Maryland, 1994.

[25] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley-Interscience Series in Discrete Mathematics, John Wiley & Sons, New York, 1986.

[26] M. Shaked and G. Shanthikumar (eds.), *Stochastic Scheduling*, Academic Press, California, 1994.

[27] R. Shostak, *Deciding linear inequalities by computing loop residues*, J. Assoc. Comput. Mach. **28** (1981), no. 4, 769–779.

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

[29] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems. EDF and Related Algorithms*, The Kluwer International Series in Engineering and Computer Science, Kluwer Academic, Dordrecht, 1998.

[30] K. Subramani, *A specification framework for real-time scheduling*, Proc. 29th Annual Conference on Current Trends in Theory and Practice of Informatics (Milovy, 2002) (W. I. Grosky and F. Plasil, eds.), Lecture Notes in Computer Science, vol. 2540, Springer, London, 2002, pp. 195–207.

[31] ———, *An analysis of zero-clairvoyant scheduling*, Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (Grenoble, 2002) (J.-P. Katoen and P. Stevens, eds.), Lecture Notes in Computer Science, vol. 2280, Springer, 2002, pp. 98–112.

[32] ———, *An analysis of partially clairvoyant scheduling*, Journal of Mathematical Modelling and Algorithms **2** (2003), no. 2, 97–119.

[33] T. Vidal and H. Fargier, *Handling contingency in temporal constraint networks: from consistency to controllabilities*, Journal of Experimental and Theoretical Artificial Intelligence **11** (1999), no. 1, 23–45.

[34] T. Vidal and M. Ghallab, *Dealing with uncertain durations in temporal constraint networks dedicated to planning*, Proc. 12th European Conference on Artificial Intelligence (Budapest, 1996), 1996, pp. 48–52.

K. Subramani: Lane Department of Computer Science and Electrical Engineering (LDCSEE), West Virginia University, Morgantown, WV 26506, USA

E-mail address: ksmani@csee.wvu.edu