Journal of Mathematical Modelling and Algorithms **2:** 97–119, 2003.

An Analysis of Partially Clairvoyant Scheduling

K. SUBRAMANI*
LCSEE, West Virginia University, Morgantown, WV 26506, U.S.A. e-mail: ksmani@csee.wvu.edu

(Received: 5 March 2002)

Abstract. Real-time scheduling problems confront two issues not addressed by traditional scheduling models, viz., parameter variability and the existence of complex relationships constraining the executions of jobs. Accordingly, modeling becomes crucial in the specification of scheduling problems in such systems. In this paper, we analyze scheduling algorithms in *Partially Clairvoyant* Real-time scheduling systems and present a new dual-based algorithm for the feasibility problem in the case of *strict relative* constraints. We also study the problem of online dispatching in Partially Clairvoyant systems and show that the complexity of dispatching is logarithmically related to the complexity of the schedulability problem.

Mathematics Subject Classifications (2000): 90C99, 90B35, 91A05.

Key words: real time, partial clairvoyance, execution time variability, timing constraints.

1. Introduction

An important feature in Real-time systems is *parameter imprecision*, i.e., the inability to accurately determine certain parameter values. The most common such parameter is job execution time. A second feature, which is prevalent in Real-time systems, is the presence of complex relationships between jobs that constrain their execution. Traditional scheduling models ([13]) do not accommodate 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. The worst-case assumption for execution times is unduly pessimistic; further depending upon the constraint involved, the worst-case may be either the smallest value or the largest value for the execution time of the job. Note that precedence graphs cannot capture relationships involving relative timing constraints.

Real-time systems (and the associated scheduling problems) can be classified as *Zero-Clairvoyant, Partially Clairvoyant* or *Totally Clairvoyant*, depending upon the information available at dispatching [18]. In this paper, we focus on *Partially Clairvoyant* Real-time systems, wherein the dispatch time of the current job may depend upon the *actual execution time of every job sequenced before it*, i.e., it

* This research was conducted in part at Aalborg Universitet, where the author was supported by a CISS Faculty Fellowship.

will be a parameterized function of the execution times of jobs sequenced before it. The primary scheduling goal is to provide an offline guarantee that the input constraints will be met at run-time, regardless of the actual execution times of the jobs at run-time.

The scheduling problem for Partially Clairvoyant systems is concerned with the following two issues.

(1) Deciding the schedulability predicate for a specified Partially Clairvoyant system (Section 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 paper is organized as follows: We introduce the Partially Clairvoyant scheduling problem in Section 2 and state the schedulability query. Section 3 motivates the necessity for the schedulability specification, through an example from Real-time design. Previous work in the design of Partially Clairvoyant systems is detailed in Section 4. Section 5 describes our dual-based approach to solve the Partially Clairvoyant schedulability problem for the special case in which all constraints are strictly relative. Online dispatching algorithms for arbitrarily constrained Partially Clairvoyant systems are discussed in Section 6. Section 7 summarizes our contributions in this paper and discusses directions for future research.

2. Statement of Problem

2.1. JOB MODEL

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

2.2. CONSTRAINT MODEL

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

$$\mathbf{A} \cdot [\mathbf{s} \;\; \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}, \quad \mathbf{e} \in \mathbf{E}, \tag{1}$$

where

- $\mathbf{A}$ is an $m \times 2 \cdot n$ rational matrix; unless explicitly stated otherwise, we assume no restrictions on the entries in $\mathbf{A}$, i.e., they represent arbitrary constraint sets.
- $\mathbf{E}$ is an axis-parallel hyper-rectangle (aph) represented by:

$$\mathbf{E} = [l_1, u_1] \times [l_2, u_2] \times \cdots \times [l_n, u_n]. \tag{2}$$

  The aph $\mathbf{E}$ models the fact that the execution time of job J_i can assume any value in the range $[l_i, u_i]$, i.e., it is not constant.

- $\mathbf{s} = [s_1, s_2, \ldots, s_n]$ is the start time vector of the jobs.
- $\mathbf{e} = [e_1, e_2, \ldots, e_n] \in \mathbf{E}$ is the execution time vector of the jobs.

Constraints can express relationships between the start times of jobs, or their finish times (the finish time of job J_i is $s_i + e_i$), but since the jobs are nonpreemptive, the addition of finish time variables does not enhance the expressiveness of the constraint model. We also note that the jobs are ordered, i.e., $s_i + e_i \leqslant s_{i+1}, i = 1, 2, \ldots, n-1$; the ordering constraints are part of the constraint system $\mathbf{A} \cdot [\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}$.

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 2.1. A Partially Clairvoyant schedule (or parametric schedule) of an ordered set of jobs, in a scheduling window, is a vector $\mathbf{s} = [s_1, s_2, \ldots, s_n]$, where each s_i, $1 \leqslant i \leqslant n$, is a function of the execution time variables of jobs sequenced prior to job J_i, i.e., $\{s_1, e_1, s_2, e_2, \ldots, s_{i-1}, e_{i-1}\}$.

Note that s_1 must be numeric, since J_1 is the first job in the sequence.

DEFINITION 2.2. A Partially Clairvoyant schedule $\mathbf{s}$ for the constraint system (1) is said to be feasible, if for all sequences $b_{\mathrm{seq}} = \langle s'_1, e'_1, s'_2, e'_2, \ldots, s'_n, e'_n \rangle$, where s'_i is chosen as per $\mathbf{s}$ and $e_i \in [l_i, u_i]$, we have, $\mathbf{A} \cdot [\mathbf{s}'\ \ \mathbf{e}']^{\mathrm{T}} \leqslant \mathbf{b}$, where $\mathbf{s}'$ and $\mathbf{e}'$ are the numeric vectors, corresponding to the sequence b_{seq}.

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

$$\exists s_1 \forall e_1 \in [l_1, u_1] \exists s_2 \forall e_2 \in [l_2, u_2], \ldots \exists s_n \forall e_n \in [l_n, u_n] \quad \mathbf{A} \cdot [\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}? \tag{3}$$

Query (3) is called the Partially Clairvoyant schedulability query. The combination of the Job model, Constraint model and the Query model constitutes a scheduling problem specification within the `E-T-C` scheduling framework [18]. The function capturing the dependence of s_i on $\{s_1, e_1, s_2, e_2, \ldots, s_{i-1}, e_{i-1}\}$ is called the *dispatch function* of job J_i.

3. Motivation

A Partially Clairvoyant system has the ability to schedule at least some job-constraint sets, which would be declared infeasible, if the system had no clairvoyance at all, i.e., if it was Zero-Clairvoyant.

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

(1) Job J_1 must finish before job J_2 commences; i.e., $s_1 + e_1 \leqslant s_2$;
(2) Job J_2 must commence within 1 unit of J_1 finishing; i.e., $s_2 \leqslant s_1 + e_1 + 1$.

A Zero-Clairvoyant 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 [17]. This is because, in order to satisfy the first constraint for all values of e_1, we must choose $e_1 = 4$, while to satisfy the second constraint for all values of e_1, we must choose $e_1 = 2$. The resulting constraint set $\{s_1 + 4 \leqslant s_2, s_2 \leqslant s_1 + 2 + 1\}$ is unsatisfiable. Now consider the following start time dispatch vector:

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

This assignment satisfies the input set of constraints, for all values of $(e_1, e_2) \in [2, 4] \times [4, 5]$ and is hence a valid 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 execution time of job J_1. This phenomenon wherein a Zero-Clairvoyant scheduler declares a constraint system infeasible, when a Partially Clairvoyant schedule exists is called *Loss of Schedulability*. Thus, Partially Clairvoyant scheduling systems have more flexibility than systems which cannot compute schedules online. Partially Clairvoyant schedulability is particularly useful in Real-time Operating Systems such as Maruti [10, 11] and MARS [5], 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. In passing, we note that the constraints in Example 1 cannot be modeled through precedence graphs, since they would create a cycle.

4. Related Work

The Partially Clairvoyant scheduling problem was proposed for the first time in [14]; they used the term 'Parametric Scheduling'; the term *Partially Clairvoyant* was proposed in [18] to represent parametric scheduling as one among 3 possible scheduling systems. In [6], a polynomial time algorithm was presented for the case in which the constraints are restricted to be 'standard', i.e., strict difference constraints. The principal technique used in their algorithm was the Fourier–Motzkin elimination procedure to eliminate existentially quantified variables [4]. 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 [7]. In Section 5, we shall provide a dual-based algorithm that represents systems of difference constraints as constraint graphs. [1] and [2] extend the results in [6] to handle the case, in which inter-period constraints (constraints across scheduling windows) are permitted in the job set. In [20], a dynamic scheduling scheme is presented; however no offline guarantees are provided. Relative separation constraints, but only in restricted forms, are considered in [9] and [8]; in their model, certain distance constraints must be satisfied between successive invocations of a job.

The chief contributions of this paper are as follows:

(1) Development of a 'dual' notion of feasibility, in the case of Partially Clairvoyant scheduling systems with standard constraints,
(2) Using the dual notion, to develop a new algorithm for the feasibility problem, and
(3) Studying the complexity of Online Dispatching.

5. Primal and Dual Algorithms

Algorithm 5.1 represents a simple, deterministic procedure that uses Quantifier elimination techniques to work through query (3), by eliminating one quantified variable at a time. This algorithm was first proposed in [6] and will henceforth be referred to as the *Primal Algorithm*.

Algorithm 5.2 describes the procedure for eliminating the universally quantified execution variable $e_i \in [l_i, u_i]$. ELIM-EXIST-VARIABLE() is implemented through the Fourier–Motzkin elimination technique discussed in [19].

Note that $\mathbf{A} \cdot [\mathbf{s}\ \ \mathbf{e}] \leqslant \mathbf{b}$ is a polyhedron in $2{\cdot}n$ dimensions the procedures ELIM-EXIST-VARIABLE() and ELIM-UNIV-VARIABLE() each reduce the dimension of this polyhedron by 1, while preserving the solution space. Thus, in order to establish the correctness of Algorithm 5.1, all that we need to show is that the procedures ELIM-UNIV-VARIABLE(e_n) and ELIM-EXIST-VARIABLE(s_n) preserve the solution space. We can then use induction to establish the correctness of Algorithm 5.1. The correctness of Algorithm 5.2 to eliminate the last variable if it is universal, has been argued in [15], while the correctness of the Fourier–Motzkin elimination procedure to eliminate the last variable if it is existential is discussed in [4] and [12].

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. Thus, the elimination of k existential quantifiers could increase the size of the constraint set to $O(m^{2^k})$ [15]. Clearly, the exponential size blow-up, makes the Algorithm 5.1 impractical for general constraint sets.

LEMMA 5.1 *Algorithm* 5.1 *correctly decides the Partially Clairvoyant schedulability query* (3).

```
Function PARTIALLY-CLAIRVOYANT-SCHEDULER (E, A, b)
for (i = n down to 2) do
    ELIM-UNIV-VARIABLE(e_i)
    ELIM-EXIST-VARIABLE(s_i)
    REMOVE-REDUNDANCIES()
    if (CHECK-INCONSISTENCY()) then
      return( false)
    end if
end for
ELIM-UNIV-VARIABLE (e_1)
REMOVE-REDUNDANCIES()
if (CHECK-INCONSISTENCY()) then
    return(false)
end if
if (a ⩽ s_1 ⩽ b, a, b ⩾ 0) then
    return(A Partially Clairvoyant schedule exists)
else
    return(A Partially Clairvoyant schedule does not exist)
end if
```

Algorithm 5.1. A quantifier elimination algorithm for deciding Partially Clairvoyant schedulability.

```
Function ELIM-UNIV-VARIABLE (A, b)
Substitute e_i = l_i in each constraint that can be written in the form e_i ⩾ ()
Substitute e_i = u_i in each constraint that can be written in the form e_i ⩽ ()
```

Algorithm 5.2. Eliminating universally quantified variable $e_i \in [l_i, u_i]$.

Proof. Follows from the discussion above. □

For the rest of this section, we confine our discussion to the class of standard constraints; this class was introduced in [6] to describe strict relative constraints between jobs.

DEFINITION 5.1. A constraint is said to be standard, if it represents a strict difference constraint between exactly 2 jobs.

As per Definition 5.1, the relationships between job J_i and job J_j are standard, if they fall into one of the following categories:

(1) A difference constraint between the start time of J_i and the start time of J_j, e.g., $s_i \leqslant s_j + c$.
(2) A difference constraint between the start time of J_i and the finish time of J_j, e.g., $s_i \leqslant s_j + e_j + c$.
(3) A difference constraint between the finish time of J_i and the start time of J_j, e.g., $s_i + e_i \leqslant s_j + c$.

(4) A difference constraint between the finish time of J_i and the finish time of J_j, e.g., $s_i + e_i \leqslant s_j + e_j + c$.

Note that *absolute constraints*, i.e., constraints of the form $s_i \geqslant a$ can also be treated as relative constraints through the addition of a dummy job J_0, with start-time s_0 and execution time $e_0 \in [0, 0]$. Without loss of generality, we assume that all constraints are strictly relative; doing so, keeps the analysis uniform.

The above restriction is called `<aph|stan|param>` within the `E-T-C` scheduling framework [18].

Observe that standard constraints are in fact difference constraints between jobs; consequently, they do have a constraint graph structure [3]. In Section 5.1, we shall show how to construct the constraint graph corresponding to a set of standard constraints.

5.1. CONSTRUCTION OF THE CONSTRAINT GRAPH FOR STANDARD CONSTRAINTS

Given a set of n jobs, with standard constraints imposed on their execution, we construct a graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$, where $\mathbf{V}$ is the set of vertices and $\mathbf{E}$ is the set of edges.

(1) $\mathbf{V} = \langle s_1, s_2, \ldots, s_n \rangle$, i.e., one node for the start time of each job,
(2) For every constraint of the form: $s_i + k \leqslant s_j$, construct an arc $s_i \rightsquigarrow s_j$, with weight $-k$,
(3) For every constraint of the form: $s_i + e_i \leqslant s_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $k - e_i$,
(4) For every constraint of the form: $s_i \leqslant s_j + e_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $e_j + k$,
(5) For every constraint of the form: $s_i + e_i \leqslant s_j + e_j + k$, construct an arc $s_i \rightsquigarrow s_j$, with weight $e_j - e_i + k$.

OBSERVATION 5.3. In the constraint graph, there are n vertices and m edges, corresponding to a job set with n jobs and m standard constraints on their execution.

OBSERVATION 5.4. There are at most 4 edges from node s_i to 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.

COROLLARY 5.1. *In the case of standard constraints, the constraint graph has at most* $O(n^2)$ *edges.*

Proof. Follows from the fact there are exactly $n \cdot (n-1)$ vertex pairs, with at most 4 edges between each vertex pair. □

EXAMPLE 2. 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.

$$\begin{aligned} 4 \leqslant e_1 \leqslant 8,\ 6 \leqslant e_2 \leqslant 11,\ 10 \leqslant e_3 \leqslant 13,\ 3 \leqslant e_4 \leqslant 9 \\ s_4 + e_4 \leqslant 56 \\ s_4 + e_4 \leqslant s_3 + e_3 + 12 \\ s_2 + e_2 + 18 \leqslant s_4 \\ s_3 + e_3 \leqslant s_1 + e_1 + 31 \\ 0 \leqslant s_1,\ s_1 + e_1 \leqslant s_2,\ s_2 + e_2 \leqslant s_3, s_3 + e_3 \leqslant s_4 \end{aligned} \tag{5}$$

Figure 1 represents the corresponding constraint graph.

Given an instance of <aph|stan|param>, we use the procedure in Section 5.1 to construct the constraint graph, which is provided as input to Algorithm 5.3.

OBSERVATION 5.5. 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; likewise, 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; however, observe that there can exist precisely one nonredundant constraint of each of the 4 types between any pair of nodes in the constraint graph. Further, the redundant edge can be identified and eliminated in O(1) time, when a new edge is created, as demonstrated by Algorithm 5.5.

OBSERVATION 5.6. The only manner in which infeasibility is detected is through the occurrence of a negative cost self-loop on any vertex (step (5) of Algorithm 5.4). These loops could occur in two ways:

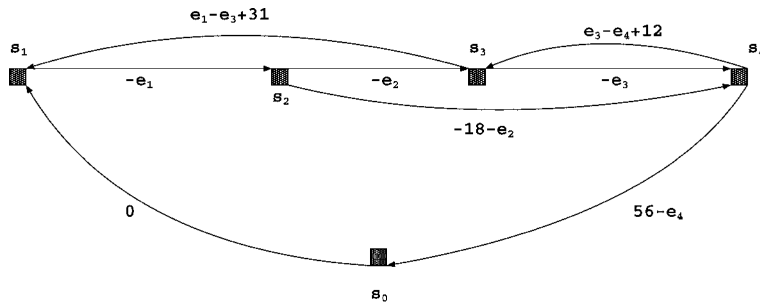

Figure 1. Constraint graph of system (5).

Function PARTIALLY-CLAIRVOYANT-STANDARD ($G = \langle V, E \rangle$)

for ($i = n$ **down to** 1) **do**
    Substitute $e_i = u_i$ on all edges where e_i is prefixed with a negative sign
    Substitute $e_i = l_i$ on all other edges {We have now eliminated e_i in $\forall e_i \in [l_i, u_i]$}
    $G' = \langle V', E' \rangle$ = VERTEX-CONTRACT(s_i)
end for
return(`A Partially Clairvoyant schedule exists`)

Algorithm 5.3. The dual-based algorithm for `<aph|stan|param>`.

Function VERTEX-CONTRACT ($G = \langle V, E \rangle, s_i$)

for each edge $s_j \rightsquigarrow s_i$, with weight w_{ji} **do**
  for each edge $s_i \rightsquigarrow s_k$, with weight w_{ik} **do**
    Add an edge (say e_{new}) $s_j \rightsquigarrow s_k$ with weight $w_{ji} + w_{ik}$
    if $j = k$ **then**
      if $(w_{ji} + w_{ik} < 0)$ **then**
        return(`A Partially Clairvoyant schedule does not exist`) {See Observation 5.6}
      end if {Eliminating self-loops}
    else
      Discard e_{new}
      continue {We do not add self-loops to the edge set}
    end if
    $E' = E \cup e_{\text{new}}$
    REMOVE-REDUNDANT($G = \langle V, E' \rangle, s_j, s_k, e_{\text{new}}$)
  end for
  $E' = E' - (s_j \rightsquigarrow s_i)$
end for
for each edge $s_i \rightsquigarrow s_k$, with weight w_{ik} **do**
  $E' = E' - (s_i \rightsquigarrow s_k)$
end for
$V' = V - \{s_i\}$ {We have now eliminated s_i in $\exists s_i$}

Algorithm 5.4. Vertex contraction.

(1) The contraction of a vertex results in a negative cost self-loop on another vertex. For instance, consider the constraint graph, corresponding to the constraint set $\{s_1 + 8 \leqslant s_2,\ s_2 \leqslant s_1 + 7\}$; the contraction of vertex s_2 results in a self-loop at s_1 of weight -1;

(2) The contraction of a vertex results in a self-loop of the following form: $-e_a + c$ (or $e_a - c$), on vertex s_a. For instance, consider the constraint graph corresponding to the constraint set $\{s_1 + e_1 + 7 \leqslant s_2,\ s_2 \leqslant s_1 + 12\}$; the contraction of vertex s_2 results in the self-loop: $-e_1 + 5$. For this loop to have nonnegative

Function REMOVE-REDUNDANT($G = \langle V, E' \rangle, s_j, s_k, e_{\text{new}}$)

{e_{new} is an edge from s_j to s_k, with weight $w_{ji} + w_{jk}$}
Let t denote the type of e_{new} {Recall that any edge in the constraint graph is one of the four types described in Observation 5.4}
if (there is precisely one edge between s_j and s_k in G of type t) **then**
    {In this case, e_{new} is the first edge of type t between s_j and s_k and hence there is nothing to be done.}
    return
end if
{In this case, there are two edges between s_j and s_k of type t; one of the edges existed prior to the contraction of vertex s_i and e_{new} is the new edge.}
Retain the edge with the smaller numeric coefficient and delete the other edge {For instance, let the 2 edges be of Type 4, with l_1 having weight $e_k - e_j + k_1$ and l_2 having weight $e_k - e_j + k_2$. If $k_1 \leqslant k_2$ retain l_1, otherwise retain l_2.}

Algorithm 5.5. Removing redundant edges in the constraint graph.

cost, we must have $-e_1 + 5 \geqslant 0$, i.e., $e_1 \leqslant 5$. In this case, either $u_1 \leqslant 5$, in which case the edge can be discarded (since it is redundant), or $u_1 > 5$ in which case, the system is infeasible.

5.2. CORRECTNESS

In order to prove the correctness of Algorithm 5.3, we need to develop a few concepts.

DEFINITION 5.1. Let

$$\exists s_1 \forall e_1 \in [l_1, u_1]\ \exists s_2 \forall e_2 \in [l_2, u_2], \ldots \exists s_n \forall e_n \in [l_n, u_n] \quad \mathbf{A} \cdot [\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b} \quad (6)$$

represent a Partially Clairvoyant system of standard constraints. Let $G = \langle V, E \rangle$ represent the constraint graph of this constraint system, constructed as per the discussion in Section 5.1. Let C denote a simple, directed cycle in G, on the vertices $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ with $\{i_1, i_2, \ldots, i_k\} \in \{1, 2, \ldots, n\}$. Without loss of generality, we assume that $i_1 < i_2 < i_3 < \cdots < i_k$. Note that an edge in C can exist between any pair of vertices. The Partially Clairvoyant cost of C is defined as the numeric value returned by Algorithm 5.6, with C as input.

It is not hard to see that Algorithm 5.6 is similar to Algorithm 5.3; the input to Algorithm 5.6 must be a cycle and it computes and returns the Partially Clairvoyant cost of that cycle. Note that the ordering information is crucial, in that the vertices must be eliminated in the order $\{s_{i_k}, s_{i_{k-1}}, \ldots, s_{i_1}\}$. It is understood that when the execution time variables are eliminated through substitution, the weights on the edges are adjusted accordingly.

Function COMPUTE-PARTIALLY-CLAIRVOYANT-COST (C, $\{i_1, i_2, \ldots, i_k\}$)

{The list $\langle i_1, i_2, \ldots, i_k \rangle$ is a list of vertex indices, with each $i_j \in \{1, 2, \ldots, n\}$, $j = 1, 2, \ldots, k$. Without loss of generality, we assume that $i_1 < i_2 < \cdots < i_k$.}
if ($k = 2$) **then**
  {There are precisely 2 vertices and 2 edges in the cycle C; recall that C is a simple cycle in G.}
  {Since C is a simple cycle, there is precisely one edge into vertex s_{i_2} and one edge into s_{i_1}.}
  Adjust weight $w_{i_2 i_1}$ to reflect the substitution $e_{i_2} = u_{i_2}$ and weight $w_{i_1 i_2}$ to reflect the substitution $e_{i_2} = l_{i_2}$.
  Let cost $= w_{i_1 i_2} + w_{i_2 i_1}$.
  Adjust cost to reflect the substitution $e_{i_1} = u_{i_1}$ if cost is a decreasing function of e_{i_1} and $e_{i_1} = l_{i_1}$ otherwise.
  {It is important to note that if e_{i_1} appears in cost, it is either as e_{i_1} or as $-e_{i_1}$.}
  return(cost)
else
  {We eliminate s_{i_k} from the cycle.}
  Let s_{i_p} and s_{i_q} denote the vertices in C to which s_{i_k} is connected; further, we assume that the edges of C are $s_{i_p} \rightsquigarrow s_{i_k}$ and $s_{i_k} \rightsquigarrow s_{i_q}$.
  Adjust $w_{i_p i_k}$ to reflect the substitution $e_{i_k} = l_{i_k}$ and $w_{i_k i_q}$ to reflect the substitution $e_{i_k} = u_{i_k}$.
  Create a new edge $s_{i_p} \rightsquigarrow s_{i_q}$ having weight $w_{i_p i_q} = w_{i_p i_k} + w_{i_k i_q}$.
  {Since C is a cycle, there did not exist an edge from s_{i_p} to s_{i_q} prior to the above step.}
  Let C' denote the new cycle, thus created.
  return(COMPUTE-PARTIALLY-CLAIRVOYANT-COST (C, $\{i_1, i_2, \ldots, i_{k-1}\}$).)
end if

Algorithm 5.6. Computing the Partially Clairvoyant cost of a simple, directed cycle.

For the rest of this section, we assume that $\mathbf{Q}(\mathbf{s},\ \mathbf{e})\ \mathbf{A} \cdot [\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}$ is a Partially Clairvoyant system of standard constraints, where

$$\mathbf{Q}(\mathbf{s},\ \mathbf{e}) = \exists s_1 \forall e_1 \in [l_1, u_1] \exists s_2 \forall e_2 \in [l_2, u_2] \ldots \exists s_n \forall e_n \in [l_n, u_n]$$

and G is the corresponding constraint graph.

When G is presented to Algorithm 5.3, steps (2:) and (3:) eliminate variable e_n and step (4:) eliminates vertex s_n to give a new constraint graph G'.

LEMMA 5.2 *The elimination of variable e_n through steps* (2:) *and* (3:) *of Algorithm* 5.3 *preserves simple cycles having negative Partially Clairvoyant cost, i.e., the constraint graph G has a simple cycle having negative Partially Clairvoyant cost before the execution of steps* (2:) *and* (3:) *if and only if it has a simple cycle having negative Partially Clairvoyant cost, after the execution of steps* (2:) *and* (3:).

Proof. Let C denote a simple, directed cycle in G, having negative Partially Clairvoyant cost. We first observe that if C does not include s_n, then the theorem is trivially true, since edges of cycles not involving s_n, cannot have weights depending on e_n (as per our definition of relative timing constraints) and hence the execution of steps (2:) and (3:) leaves C unaltered. Now consider the case in which C does pass through s_n. From Algorithm 5.6, it is clear that any negative cost Partially Clairvoyant cycle through s_n, must have e_n set to u_n on all edges where e_n is prefixed with a negative sign and to l_n on all other edges, i.e., C is retained. For the same reason, if G does not have a negative cost Partially Clairvoyant cycle, the execution of steps (2:) and (3:) of Algorithm 5.3 cannot create one. □

We now assume that steps (2:) and (3:) of Algorithm 5.3 have been executed on G.

LEMMA 5.3 *There exists a negative cost Partially Clairvoyant cycle through s_n if and only if either there exists a negative cost Partially Clairvoyant cost cycle in the graph G' returned by step* (4:) *of Algorithm* 5.3 *or Algorithm* 5.4 *executes Step* (6:).

Proof. Let C be a simple, directed cycle in G, having negative Partially Clairvoyant cost through vertex s_n. Consider the case in which C does pass through vertex s_n. The following cases arise.

(1) C consists of 2 vertices, i.e., vertex s_n and some other vertex, say s_p. When s_n is contracted C becomes a loop around vertex s_p and this loop has negative Partially Clairvoyant cost. Consequently, this loop is detected in steps (4:)–(6:) of Algorithm 5.4.

(2) C consists of more than 2 vertices. Note that step (3:) of Algorithm 5.4 combines edge-pairs going through s_n, so C exists in the constraint graph after the execution of step (3:). However, an edge created in step (3:) could be thrown out at step (13:), if it is deemed redundant (see Figure 2), thereby destroying C.

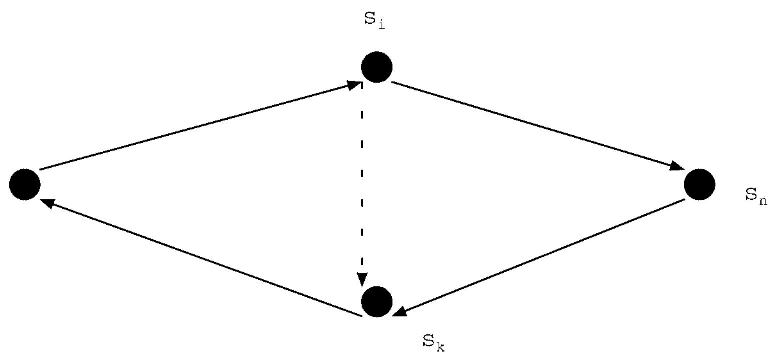

Figure 2. Contracting vertex s_n.

But this means that there is a simple, directed cycle in G having Partially Clairvoyant cost, even lower than C (the cycle that includes the dashed edge from s_i to s_k, that shortcuts s_n); it follows that negative Partially Clairvoyant cost cycles are preserved. For the same reason, it follows that contracting s_n does not create negative cost Partially Clairvoyant cycles, if none exist. □

Although Lemmas 5.2 and 5.3 were proved for the elimination of e_n and s_n respectively from the constraint graph, it is easy to see that the argument can be applied inductively to conclude that

THEOREM 5.1 *Algorithm* 5.3 *returns* (`A Partially Clairvoyant schedule does not exist`) *if and only if the constraint graph G has a simple cycle having negative Partially Clairvoyant cost.*

Proof. Note that any cycle in the constraint graph, including one having negative Partially Clairvoyant cost, has length at most n. Steps (2:)–(4:) of Algorithm 5.3 preserve negative cost Partially Clairvoyant cycles, while reducing the number of vertices in the constraint graph by 1. Let C be a simple cycle in G having negative Partially Clairvoyant cost; as discussed in Lemma 5.3, when the length of C is reduced to 2, it is detected by the VERTEX-CONTRACT() operation.

If there is no negative cost Partially Clairvoyant cycle in G, then Algorithm 5.3 falls through to step (6:) and returns (`A Partially Clairvoyant schedule exists`). □

THEOREM 5.2 *A Partially Clairvoyant system of standard constraints, as specified in system* (6) *has a feasible schedule, if and only if the corresponding constraint graph does not have a simple cycle having negative Partially Clairvoyant cost.*

Proof. Let $\mathbf{Q}(\mathbf{s},\ \mathbf{e})\ \mathbf{A}\cdot[\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}$ represent the system of standard constraints, where

$$\mathbf{Q}(\mathbf{s},\ \mathbf{e}) = \exists s_1 \forall e_1 \in [l_1, u_1] \exists s_2 \forall e_2 \in [l_2, u_2] \ldots \exists s_n \forall e_n \in [l_n, u_n]$$

and let G denote the corresponding constraint graph.

We first assume that G has a simple, directed cycle C on the vertices $\{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$, having negative Partially Clairvoyant cost, where $i_1 < i_2 < \cdots < i_k$. Observe that as per the construction procedure in Section 5.1, the subset of constraints in the constraint system $\mathbf{A}\cdot[\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}$, corresponding to C can be represented as:

$$\begin{aligned}
& \exists s_{i_1} \forall e_{i_1} \in [l_{i_1}, u_{i_1}] \exists s_{i_2} \forall e_{i_2} \in [l_{i_2}, u_{i_2}] \ldots \exists s_{i_k} \forall e_{i_k} \in [l_{i_k}, u_{i_k}] \\
s_{i_1} - s_{i_2} &\leqslant f_1(e_{i_1}, e_{i_2}) \\
s_{i_2} - s_{i_3} &\leqslant f_2(e_{i_2}, e_{i_3}) \\
\vdots\ &\ \vdots \\
s_{i_{k-1}} - s_{i_k} &\leqslant f_{k-1}(e_{i_{k-1}}, e_{i_k}) \\
s_{i_k} - s_{i_1} &\leqslant f_k(e_{i_k}, e_{i_1}). \qquad (7)
\end{aligned}$$

The notation $f_1(e_{i_1}, e_{i_2})$ represents the fact that the weight of the edge between vertex s_{i_1} and s_{i_2} is, in general, a function of e_{i_1} and e_{i_2}; $f_2()$, $f_3()$, ..., $f_k()$ have similar explanations. Let us say that the constraint system (7) is provided as input to Algorithm 5.1. Algorithm 5.1 proceeds by eliminating e_{i_k} from the last two constraints and then adding them together to eliminate s_{i_k}. In the succeeding iteration, $e_{i_{k-1}}$ is eliminated from the last two constraints in the resultant constraint set and then $s_{i_{k-1}}$ is eliminated by adding them together. This process continues, till we reach the contradiction $0 \leqslant -a$, where $a > 0$; we must reach this contradiction, since the cycle C has negative Partially Clairvoyant cost. Thus, Algorithm 5.1 would declare the constraint system corresponding to C to be infeasible. However, the addition of constraints to an infeasible constraint system cannot make it feasible. It therefore, follows that the initial system of standard constraints, viz., $\mathbf{Q}(\mathbf{s},\ \mathbf{e})\ \mathbf{A} \cdot [\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}$ does not have a Partially Clairvoyant schedule.

We now assume that G does not contain a cycle having negative Partially Clairvoyant cost. We use induction on the number of jobs n to argue that the system $\mathbf{Q}(\mathbf{s},\ \mathbf{e})\ \mathbf{A} \cdot [\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}$ has a Partially Clairvoyant schedule.

It is clear that the base case of the induction is $n = 2$, since if there is only one job, there cannot be any constraints; recall that we allow only strict difference constraints. Accordingly, we denote the Partially Clairvoyant system as:

$$\exists s_1 \forall e_1 \in [l_1, u_1] \exists s_2 \forall e_2 \in [l_2, u_2]\ \ \mathbf{A} \cdot [\mathbf{s}\ \ \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}. \tag{8}$$

The corresponding constraint graph G has 2 nodes s_1 and s_2 with the edges between them, representing the constraints on the jobs. The hypothesis assumes that there are no negative cost Partially Clairvoyant cycles in G. Observe that e_2 can be eliminated from G, using steps (2:) and (3:) of Algorithm 5.3, without creating negative cost Partially Clairvoyant cycles. Let the resultant constraint graph be denoted by G' and the corresponding Partially Clairvoyant specification be denoted by:

$$\exists s_1 \forall e_1 \in [l_1, u_1] \exists s_2\ \ \mathbf{A}' \cdot [\mathbf{s}\ \ e_1]^{\mathrm{T}} \leqslant \mathbf{b}'. \tag{9}$$

Let S_{in} denote the set of constraints which are represented by edges going from s_1 to s_2 in G'. Note that each constraint $l_i \in S_{\mathrm{in}}$ can be written in the form $s_1 - s_2 \leqslant f_i()$, i.e., in the form $s_2 \geqslant s_1 - f_i()$, for appropriately chosen $f_i()$. Similarly, let S_{out} denote the set of constraints which are represented by edges going from s_2 to s_1 in G'. Note that each constraint $m_j \in S_{\mathrm{out}}$ can be written in the form $s_2 - s_1 \leqslant g_j()$, i.e., in the form $s_2 \leqslant s_1 + g_j()$, for appropriately chosen $g_j()$. Observe that the $f_i()$ and $g_j()$ are functions of e_1 only.

Consider the point

$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 0 \\ \max_{S_{\mathrm{in}}}\{s_1 - f_i()\} \leqslant s_2 \leqslant \min_{S_{\mathrm{out}}}\{s_1 + g_j()\} \end{bmatrix}.$$

We claim that $\mathbf{s}$ represents a feasible schedule for the Partially Clairvoyant system (9).

Assume the contrary and let $e_1' \in [l_1, u_1]$ be an execution time such that for $s_1 = 0$ and $e_1 = e_1'$, System (9) cannot be satisfied by any value of s_2. This means that at $e_1 = e_1'$, we have $\max_{S_{\text{in}}}\{s_1 - f_i()\}|_{e_i=e_1'} > \min_{S_{\text{out}}}\{s_1 + g_j()\}|_{e_1=e_1'}$. It immediately follows that $\max_{S_{\text{in}}}\{f_i()\}|_{e_1=e_1'} + \min_{S_{\text{out}}}\{g_j()\}|_{e_1=e_1'} < 0$, i.e., we have a simple negative cost cycle in G'. From Algorithm 5.6, it is clear, that the existence of a simple negative cost cycle implies the existence of a simple negative cost Partially Clairvoyant cycle. However, this violates the hypothesis, which assumed that G' did not have a simple cycle of negative Partially Clairvoyant cost. Thus, **s** is a feasible schedule for the Partially Clairvoyant system (9).

We now need to show that **s** is also a feasible schedule for system (8). Let $\mathbf{s}_{\mathbf{e}_1'}$ be the (numeric) vector corresponding to $e_1' \in [l_1, u_1]$. Observe that a constraint in system (8) that can be written in the form $e_2 \leqslant ()$ is satisfied by $\mathbf{s}_{\mathbf{e}_1'}$ with $e_2 = u_2$ and hence for all values of $e_2 \in [l_2, u_2]$. Likewise, a constraint in system (8) that can be written in the form $e_2 \geqslant ()$ is satisfied by $\mathbf{s}_{\mathbf{e}_1'}$ with $e_2 = l_2$ and hence for all values of $e_2 \in [l_2, u_2]$. It follows that **s** represents a feasible schedule for the Partially Clairvoyant system (8). The base case of the induction is proven.

We now assume that Theorem 5.2 is true for all job sets of size at most k. Now consider a job set of size $k + 1$. Accordingly, the Partially Clairvoyant system is:

$$\begin{aligned} &\exists s_1 \forall e_1 \in [l_1, u_1] \exists s_2 \forall e_2 \in [l_2, u_2] \ldots \\ &\quad \exists s_{k+1} \forall e_{k+1} \in [l_{k+1}, u_{k+1}] \;\; \mathbf{A} \cdot [\mathbf{s} \;\; \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}. \end{aligned} \tag{10}$$

Let G denote the corresponding constraint graph. By Lemma 5.2, we know that e_{k+1} can be eliminated from G without creating negative cost Partially Clairvoyant cycles. Let G' denote the resulting constraint graph and the corresponding Partially Clairvoyant specification is denoted by:

$$\exists s_1 \forall e_1 \in [l_1, u_1] \exists s_2 \forall e_2 \in [l_2, u_2] \ldots \exists s_{k+1} \;\; \mathbf{A}' \cdot [\mathbf{s} \;\; \mathbf{e}']^{\mathrm{T}} \leqslant \mathbf{b}', \tag{11}$$

where $\mathbf{e}' = [e_1, e_2, \ldots, e_k]^{\mathrm{T}}$.

Let S_{in} denote the set of constraints which are represented by edges going into s_{k+1} and let S_{out} denote the set of constraints which are represented by edges going out of s_{k+1}. Observe that each constraint $l_i^a \in S_{\text{in}}$ can be written in the form $s_a - s_{k+1} \leqslant f_i^a()$, i.e., in the form $s_{k+1} \geqslant s_a - f_i^a()$, for suitable $a \in \{1, 2, \ldots, k\}$ and suitably chosen $f_i^a()$. Likewise, every constraint $m_j^b \in S_{\text{out}}$ can be written in the form $s_{k+1} - s_b \leqslant g_j^b()$, i.e., $s_{k+1} \leqslant s_b + g_j^b()$, for suitable $b \in \{1, 2, \ldots, k\}$ and suitably chosen $g_j()$. Note that the indices a and b will change depending upon the vertex from which the edge originates or ends. Fix s_{k+1} as

$$\max_{S_{\text{in}}}\{s_a - f_i^a()\} \leqslant s_{k+1} \leqslant \min_{S_{\text{out}}}\{s_b + g_j^b()\}. \tag{12}$$

Now contract vertex s_{k+1} and eliminate the redundant constraints as described in Algorithm 5.4 to get a new constraint graph G''; the corresponding Partially Clairvoyant constraint system is denoted as:

$$\begin{aligned} &\exists s_1 \forall e_1 \in [l_1, u_1] \exists s_2 \forall e_2 \in [l_2, u_2] \ldots \\ &\quad \exists s_k \forall e_k \in [l_k, u_k] \;\; \mathbf{A}' \cdot [\mathbf{s}' \;\; \mathbf{e}']^{\mathrm{T}} \leqslant \mathbf{b}'', \end{aligned} \tag{13}$$

where $\mathbf{s}' = [s_1, s_2, \ldots, s_k]^{\mathrm{T}}$. Observe that G'' cannot contain a negative cost Partially Clairvoyant cycle, as per Lemma 5.3. By the inductive hypothesis, system (13) has a Partially Clairvoyant schedule, $\mathbf{s}_{\mathrm{sol}}$, which can be recursively constructed as follows: $s_{\mathrm{sol}}[1] = 0$, while $s_{\mathrm{sol}}[i]$, $2 \leqslant i \leqslant k$, is a parameterized function of the execution times $\{e_1, e_2, \ldots, e_{i-1}\}$, constructed in precisely the same manner as s_{k+1}, in relation (12). It is clear from this description that each $s_{\mathrm{sol}}[i]$, $2 \leqslant i \leqslant k$, evaluates to a nonempty, numeric interval, when $s_1, e_1, \ldots, s_{i-1}$ and e_{i-1} are provided.

Now consider the point

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}_{\mathrm{sol}} \\ s_{k+1} \end{bmatrix} \tag{14}$$

with s_{k+1} constructed as per relation (12). We claim that $\mathbf{s}$ is a Partially Clairvoyant schedule for system (11).

Assume the contrary and let it be the case that $\mathbf{s}$ is not a valid Partially Clairvoyant schedule. It follows that there is a sequence $b_{\mathrm{seq}} = \langle s_1, e_1, s_2, e_2, \ldots, s_k, e_k, s_{k+1}\rangle$, where the s_is are chosen according to system (14) and $e_i \in [l_i, u_i]$, such that the constraint system $\mathbf{A}' \cdot [\mathbf{s}\ \ \mathbf{e}'] \leqslant \mathbf{b}'$ in system (11) is violated. From the manner in which the $s_i s$ are recursively constructed, it must the case that there exists a first job J_j, such that the interval to choose s_j is empty. We first observe that $j \not\leqslant k$, since that would violate the inductive hypothesis which assumed that $\mathbf{s}_{\mathrm{sol}}$ was a valid Partially Clairvoyant schedule for system (13). Therefore, $j = k+1$ and the interval to choose s_{k+1} is empty.

Let $l_i^a \in S_{\mathrm{in}}$ be the constraint which is maximized by b_{seq}; likewise, let $m_j^b \in S_{\mathrm{out}}$ be the constraint which is minimized by b_{seq}. Observe that as a result of the VERTEX-CONTRACT() operation, the edge in G'' corresponding to l_i^a is merged with the edge corresponding to constraint m_j^b to obtain an edge e_{ab} between vertex s_a and s_b in G''. Let l_{ab} denote the corresponding constraint between jobs J_a and J_b. Since G'' does not have a negative cost Partially Clairvoyant cycle, by the inductive hypothesis, b_{seq} must respect the constraint l_{ab}. (If edge e_{ab} is deemed redundant, then b_{seq} respects an even stronger constraint!) This means that $\max_{S_{\mathrm{in}}}\{s_a - f_i^a()\}|_{b\mathrm{seq}} \leqslant \min_{S_{\mathrm{out}}}\{s_b + g_j^b()\}|_{b\mathrm{seq}}$, i.e., there is a nonempty interval to choose s_{k+1}.

Finally, we need to show that $\mathbf{s}$ is also a valid, Partially Clairvoyant schedule for system (10); we use the same argument that was used in the base case. For each sequence $b_{\mathrm{seq}} = \langle s_1, e_1, \ldots, s_k, e_k, s_{k+1}\rangle$, the constraint in system (10) that contains e_{k+1} in the form $e_{k+1} \geqslant ()$ is met with $e_{k+1} = l_{k+1}$ and therefore for all values of $e_{k+1} \in [l_{k+1}, u_{k+1}]$; likewise, the constraint in system (10) that contains e_{k+1} in the form $e_{k+1} \leqslant ()$ is met $e_{k+1} = u_{k+1}$ and therefore for all values of $e_{k+1} \in [l_{k+1}, u_{k+1}]$.

By applying the principle of mathematical induction, we conclude that if the constraint graph does not have a negative cost Partially Clairvoyant cycle, the corresponding constraint system has a Partially Clairvoyant schedule. $\square$

The correctness of Algorithm 5.3 follows immediately from Theorems 5.1 and 5.2.

5.3. COMPLEXITY

The elimination of a 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, for appropriately chosen constraint sets (see [16]). *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 Algorithm 5.3 is $O(n^3)$. Note that constraint sets can be chosen so that the running time of Algorithm 5.3 is $\Omega(n^3)$.

5.4. DIFFERENCES BETWEEN THE PRIMAL AND DUAL ALGORITHMS

The principal differences between Algorithm 5.3 and Algorithm 5.1 are as follows:

(1) The primal algorithm operates by eliminating one column of the constraint matrix after another; columns are eliminated for both execution time variables and start time variables. In contrast, the elimination of an execution time variable in the dual algorithm, does not affect the structure of the constraint graph, while the elimination of a start time variable results in the elimination of a vertex and the possible creation of new edges. The primal algorithm requires space $\Omega(n^3)$ on a constraint set having n jobs and $O(n^2)$ constraints, whereas the dual algorithm can be implemented in $O(n^2)$ space on all constraint sets, having n jobs.

(2) Implementation of existential variable elimination – Algorithm 5.1 eliminates an existentially quantified variable, through pivot operations, whereas Algorithm 5.3 eliminates existentially quantified variables by vertex contraction; this is a graph operation that can be implemented in time proportional to the product of the in-degree and out-degree of the vertex being contracted;

(3) Checking inconsistencies – In the primal approach, an inconsistency is identified when we have a pair of constraints of the form: $s_i \leqslant 3, s_i \geqslant 4$; in the dual algorithm, the focus is on *Partially Clairvoyant negative cost loops*. There are exactly two types of loops:

   (a) **E**-domain loops – Suppose that vertex s_i and s_j $i < j$ are constrained as: $s_i + e_i + 8 \leqslant s_j, s_j \leqslant s_i + 10$; the contraction of s_j results in a loop with weight $2 - e_i$. Such a loop (called an **E**-domain loop) is either redundant or inconsistent.

(b) **S**-domain loops – Suppose that the vertex s_i has constraints of the form $s_i \geqslant 7, s_i \leqslant 5$. These constraints are edges of the form $s_0 - s_i \leqslant -7, s_i - s_0 \leqslant 5$. When s_i is contracted, we get a loop of cost -2, which indicates infeasibility.

In other words, the dual algorithm is a (negative) loop identification algorithm. This dual characterization of Partially Clairvoyant infeasibility may have additional applications.

Remark 5.1. The dual-based algorithm is applicable only in case of difference constraints; it is not known whether arbitrarily constrained sets have duals.

5.5. ILLUSTRATION OF THE DUAL ALGORITHM

EXAMPLE 3. Consider an instance of <aph|stan|param>, in which the underlying constraint system is represented by Figure 1 and the schedulability specification is given by query (15). Figures 3–5 display the application of Algorithm 5.3 to the constraint graph.

$$\begin{aligned}&\exists s_1\ \forall e_1 \in [4, 8]\ \exists s_2\ \forall e_2 \in [6, 11]\\ &\quad \exists s_3\ \forall e_3 \in [10, 13]\ \exists s_4 \forall e_4 \in [3, 9] \quad \{(5)\} \quad ?\end{aligned} \tag{15}$$

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

6. Complexity of Online Dispatching

As discussed in the previous section, the start time of each job is a parameterized function of the start and execution times of the jobs sequenced before it. (Table (I) provides a typical example.) 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.

The principal problem with the creation of the function lists is that we cannot *a priori* bound the length of these lists. In the case of standard constraints, it can be argued that the length of these lists is at most $O(n)$. However, there appears to be no easy way to bound the length of the function lists, when the constraint matrix is arbitrary. We argue here that explicit construction of the parameterized function lists is unnecessary; determination of feasibility is sufficient, thereby eliminating the need for storing the parameterized function lists. 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, (say) $[a', b']$ 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*.

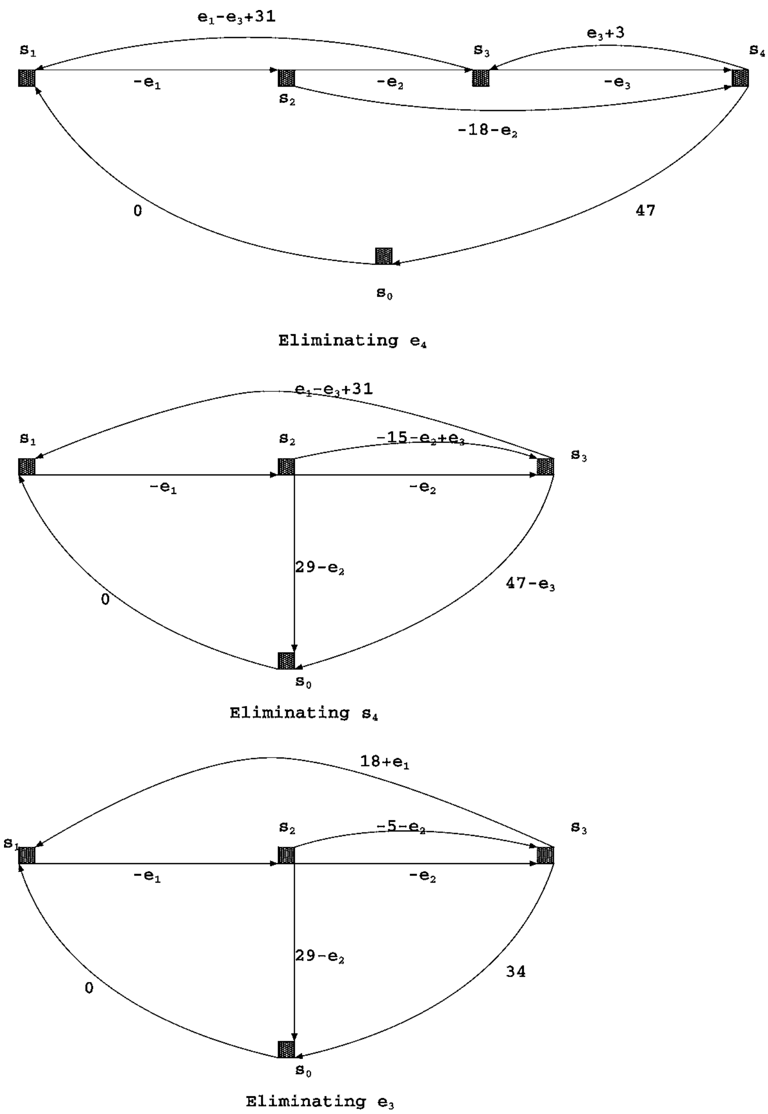

Figure 3. Algorithm 5.3 on query (15).

Let us assume the existence of an oracle, $\mathbf{\Delta}$, that decides query (3) in time $T(\mathbf{\Delta})$. Algorithm 6.1 can then be used to determine the start time of the first unexecuted job (say J_ρ) in the schedule. Note that at commencement, $\rho = 1$.

The end of the period, L, is the deadline for all jobs in the job set. We must have $0 \leqslant s_\rho \leqslant L$. The goal is to determine the exact value that can be safely assigned to s_ρ without violating the current constraint set. Observe that the constraint system $\mathbf{A} \cdot [\mathbf{s} \;\; \mathbf{e}]^{\mathrm{T}} \leqslant \mathbf{b}$ can also be written as: $\mathbf{G} \cdot \mathbf{s} + \mathbf{H} \cdot \mathbf{e} \leqslant \mathbf{b}$. Let

$$\mathbf{G}^\rho.\mathbf{s}^\rho + \mathbf{H}^\rho.\mathbf{e}^\rho \leqslant \mathbf{b}^\rho \tag{16}$$

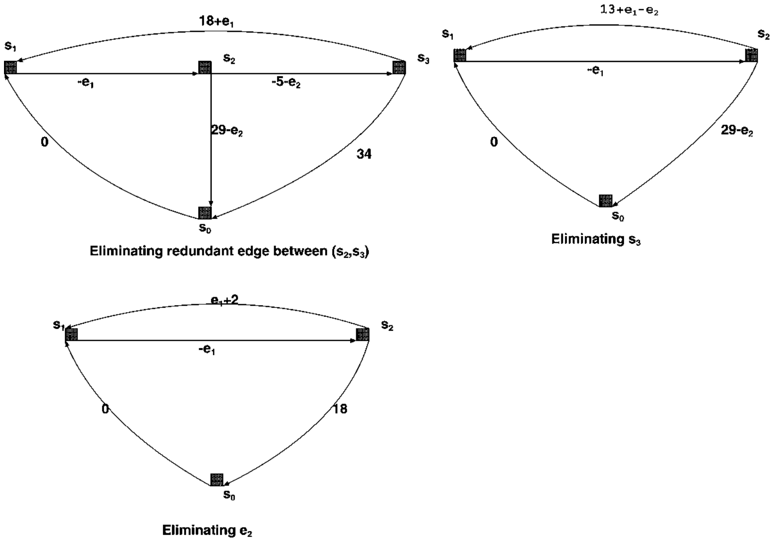

Figure 4. Algorithm 5.3 on query (15) (contd.).

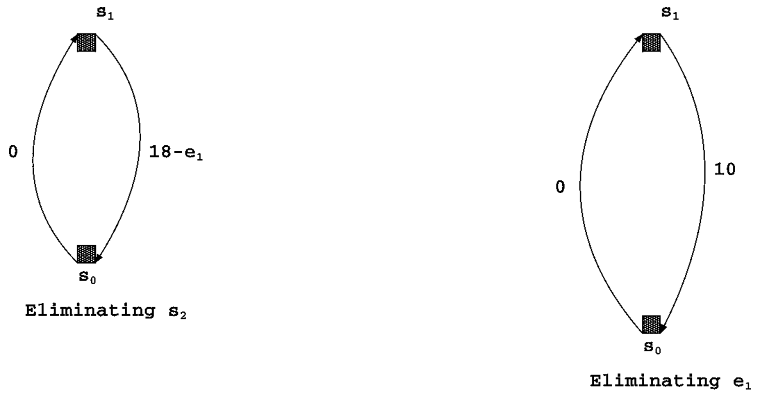

Figure 5. Algorithm 5.3 on query (15) (contd.).

Table I. List of parametric functions

Lower bound function	$\leqslant$ Start time $\leqslant$	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	$f_{n-1}'(s_1, e_1, s_2, e_2, \ldots, s_{n-1}, e_{n-1})$

Function DETERMINE-START-TIME $(\mathbf{G}^{\rho}, \mathbf{H}^{\rho}, \mathbf{b}^{\rho}, [a_l, a_h])$

{Initially $[a_l, a_h] = [0, L]$; the interval is reduced to half its original length at each level of the recursion}
Let $m' = \frac{a_h + a_l}{2}$
if $(\mathbf{\Delta}(\mathbf{G}^{\rho}, \mathbf{H}^{\rho}, \mathbf{b}^{\rho}, s_{\rho} \geqslant m'))$, **then**
  {We now know that there is a valid assignment for s_{ρ} in the interval $[m', a_h]$; the exact point in time needs to be determined}
  if $(\mathbf{\Delta}(\mathbf{G}^{\rho}, \mathbf{H}^{\rho}, \mathbf{b}^{\rho}, s_{\rho} = m'))$, **then**
    $s_{\rho} = m'$
    return
  else
    if $(a_l = a_h)$ **then**
      {The recursion has bottomed out; there does not exist a valid time to assign to s_{ρ}.}
      return('s_{ρ} cannot be assigned')
    end if
    {m' is not a valid point; however we can still recurse on the smaller interval}
    DETERMINE-START-TIME $(\mathbf{G}^{\rho}, \mathbf{H}^{\rho}, \mathbf{b}^{\rho}, [m', a_h])$
  end if
else
  {We know that the valid assignment for s_{ρ} must lie in the interval $[a_l, m']$}
  DETERMINE-START-TIME $(\mathbf{G}^{\rho}, \mathbf{H}^{\rho}, \mathbf{b}^{\rho}, [a_l, m'])$
end if

Algorithm 6.1. Partially Clairvoyant Dispatcher to determine s_{ρ}.

denote the current constraint system, 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 $\mathbf{H}$,
- $\mathbf{s}^{\rho} = [s_{\rho}, s_{\rho+1}, \ldots, s_n]^{\mathrm{T}}$; $\mathbf{s}^{1-\rho} = [s_1, s_2, \ldots s_{\rho-1}]^{\mathrm{T}}$,
- $\mathbf{e}^{\rho} = [e_{\rho}, e_{\rho+1}, \ldots, e_n]^{\mathrm{T}}$; $\mathbf{e}^{1-\rho} = [e_1, e_2, \ldots e_{\rho-1}]^{\mathrm{T}}$, and
- $\mathbf{b}^{\rho} = \mathbf{b} - (\mathbf{G}^{1-\rho} \cdot \mathbf{s}^{1-\rho} + \mathbf{H}^{1-\rho} \cdot \mathbf{e}^{1-\rho})$.

Algorithm 6.1 exploits the local convexity of s_{ρ}, i.e., if $s_{\rho} \geqslant a$ is valid and $s_{\rho} \leqslant b$ is valid, then any point $s_{\rho} = \lambda \cdot a + (1 - \lambda) \cdot b, 0 \leqslant \lambda \leqslant 1$ is valid. The cost of this strategy is $\mathrm{O}(\log L)$ calls to the oracle $\mathbf{\Delta}$, i.e., $\mathrm{O}(T(\mathbf{\Delta}) \cdot \log L)$. We have thus established that the principal complexity of the Partially Clairvoyant scheduling problem is in deciding query (3). This result is significant because it decouples dispatching complexity from decidability, i.e., Algorithm 6.1 assures us that efficient dispatching is contingent only upon efficient decidability.

7. Conclusion

In this paper, we presented a new dual-based algorithm for the problem of deciding whether a system of strictly relative constraints has a Partially Clairvoyant schedule. The basis of the dual-based algorithm was Theorem 5.1, which is the full first-order logic equivalent of the theorem in [3] for a system of simple, difference constraints.

The analysis of the dual-based algorithm provided new insights into the implementation of Partially Clairvoyant schedulers; in particular, we showed, through a reduction that the dispatching problem was not harder than the schedulability problem. This result is important in situations in which it is not known how to *a priori* bound the length of the dispatch function lists.

Some of the important open problems in Partially Clairvoyant scheduling are:

(1) Does there exist an algorithm that runs in $O(m \cdot n)$, for the problem of deciding whether a system of relative constraints has a Partially Clairvoyant schedule? Both the primal and dual algorithms take have running time $\Omega(n^3)$ on appropriately chosen constraint sets; it follows that a new approach is required to improve the running time.

(2) A detailed implementation profile of the primal and dual algorithms on various classes of constraint sets.

References

1. Choi, S.: Dynamic time-based scheduling for hard real-time systems, PhD thesis, University of Maryland, College Park, June 1997.
2. Choi, S.: Dynamic time-based scheduling for hard real-time systems, *J. Real-Time Systems*, 2000.
3. Cormen, T. H., Leiserson, C. E. and Rivest, R. L.: *Introduction to Algorithms*, 6th edn, MIT Press and McGraw-Hill, 1992.
4. Dantzig, G. B. and Eaves, B. C.: Fourier–Motzkin elimination and its dual, *J. Combin. Theory (A)* **14** (1973), 288–297.
5. Damm, A., Reisinger, J., Schwabl, W. and Kopetz, H.: The real-time operating system of MARS, *ACM Special Interest Group on Operating Systems* **23**(3) (1989), 141–157.
6. Gerber, R., Pugh, W. and Saksena M.: Parametric dispatching of hard real-time tasks, *IEEE Trans. Comput.*, 1995.
7. Huynh, Joskowicz, Lassez and Lassez: Reasoning about linear constraints using parametric queries, *FSTTCS: Foundations of Software Technology and Theoretical Computer Science* **10** (1990).
8. Han, C. C. and Lin, K. J.: Scheduling distance-constrained real-time tasks, In: *Proceedings, IEEE Real-time Systems Symposium*, Phoenix, Arizona, December 1992, pp. 300–308.
9. Han, C. C. and Lin, K. J.: Scheduling real-time computations with separation constraints, *Inform. Process. Lett.* **12** (1992), 61–66.
10. Levi, S. T., Tripathi, S. K., Carson, S. D. and Agrawala, A. K.: The `Maruti` hard real-time operating system, *ACM Special Interest Group on Operating Systems* **23**(3) (1989), 90–106.
11. Mosse, D., Agrawala, A. K. and Tripathi, S. K.: Maruti a hard real-time operating system, In: *Second IEEE Workshop on Experimental Distributed Systems*, IEEE, 1990, pp. 29–34.

12. Nemhauser, G. L. and Wolsey, L. A.: *Integer and Combinatorial Optimization*, Wiley, New York, 1999.
13. Pinedo, M.: *Scheduling: Theory, Algorithms and Systems*, Prentice-Hall, Englewood Cliffs, 1995.
14. Saksena, M.: Parametric scheduling in hard real-time systems, PhD thesis, University of Maryland, College Park, June 1994.
15. Schrijver, A.: *Theory of Linear and Integer Programming*, Wiley, New York, 1987.
16. Subramani, K. and Kovalchick, L.: Contraction versus relaxation: A comparison of two approaches for the negative cost cycle detection problem, In: P. M. A. Sloot *et al.* (eds), *Proceedings of the 3rd International Conference on Computational Science (ICCS)*, Lecture Notes in Comput. Sci., Springer-Verlag, June 2003.
17. Subramani, K.: An analysis of zero-clairvoyant scheduling, In: J.-P. Katoen and P. Stevens (eds), *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction of Systems (TACAS)*, Lecture Notes in Comput. Sci. 2280, Springer-Verlag, April 2002, pp. 98–112.
18. Subramani, K.: A specification framework for real-time scheduling, In: W. I. Grosky and F. Plasil (eds), *Proceedings of the 29th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, Lecture Notes in Comput. Sci. 2540, Springer-Verlag, November 2002, pp. 195–207.
19. Chandru, V. and Rao, M. R.: Linear programming, In: *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.
20. Wolfe, V., Davidson, S. and Lee, I.: Rtc: Language support for real-time concurrency, In: *Proceedings IEEE Real-Time Systems Symposium*, December 1991, pp. 43–52.