REPORT DOCUMENTATION PAGE

AFRL-SR-AR-TR-03-

0084

reviewing
information

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		01 JAN 02 - 31 DEC 02

4. TITLE AND SUBTITLE
TOALY CLAIRVOYANT SCHEDULING WITH RELATIVE TIMING CONSTRAINTS

5. FUNDING NUMBERS
F49620-02-1-0043

6. AUTHOR(S)
K. SUBRAMANI

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
WEST VIRGINIA UNIVERSITY
MORGANTOWN, WV

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
4015 Wilson Blvd, Room 713
Arlington, VA 22203-1954

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-02-1-0043

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Traditional scheduling models assume that the execution time of a job in a periodic job-set is constant in every instance of its execution. This assumption does not hold in real-time systems wherein job execution time is known to vary. A second feature of traditional models is their lack of expressiveness, in that constraints more complex than precedence constraints (for instance, relative timing constraints) cannot be modeled. Thirdly, the schedulability of a real-time system depends upon the degree of clairvoyance afforded to the dispatcher. In this paper, we shall discuss Totally Clairvoyant Scheduling, as modeled within the E-T-C scheduling framework. We show that instantiation of the scheduling framework captures the central issues in a real-time flow-shop scheduling problem and design a polynomial time sequential algorithm for the same. We also introduce an error-minimizing performance metric called Violation Degree and establish that optimizing this metric in a Totally Clairvoyant Scheduling System in NP-Hard.

20030326 017

14. SUBJECT TERMS	15. NUMBER OF PAGES
	23
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Final Report on Grant F49620-02-1-0043

K. Subramani
LDCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Introduction

Our proposal to the Air Force Office of Scientific Research (AFOSR), entitled "The Application of Quantified Linear Programs to Scheduling problems in Real-Time Systems" was approved for the calendar year January 2002 to December 2002. In [Subb], we formulate a flow-shop scheduling problem as a Totally Clairvoyant Scheduling problem in the E-T-C scheduling framework [Sub02]. We provide polynomial time algorithms for the case in which the constraints between jobs are strict difference constraints, also called "standard constraints". We also show that a Performance Metric Optimization problem is NP-Hard.

[Suba] represents an offshoot of the main thrust of our work. While working on short refutations for Totally Clairvoyant Scheduling systems, it occurred to us that tree-like resolution refutations of 2SAT formulas could be determined in polynomial time.

We have recently proposed the addition of optimization functions to Clairvoyant Scheduling, as well as parallel strategies for large scale problems. This proposal is currently under consideration for funding by AFOSR.

2 Acknowledgements

We thank Major Juan Vasquez for his encouragement and willingness to explore new ideas in basic research. We hope that the attached papers will vindicate his faith in us.

References

[Suba] K. Subramani. Optimal length tree-like resolution refutations for 2sat formulas. *ACM Transactions on Computational Logic.* Accepted.

[Subb] K. Subramani. Totally clairvoyant scheduling with relative timing constraints. Submitted to CIAC 2003.

[Sub02] K. Subramani. A specification framework for real-time scheduling. In W.I. Grosky and F. Plasil, editors, *Proceedings of the* 29^{th} *Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, volume 2540 of *Lecture Notes in Computer Science*, pages 195–207. Springer-Verlag, November 2002.

Totally Clairvoyant Scheduling with Relative Timing Constraints

K. Subramani *
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

Abstract

Traditional scheduling models assume that the execution time of a job in a periodic job-set is constant in every instance of its execution. This assumption does not hold in real-time systems wherein job execution time is known to vary. A second feature of traditional models is their lack of expressiveness, in that constraints more complex than precedence constraints (for instance, relative timing constraints) cannot be modeled. Thirdly, the schedulability of a real-time system depends upon the degree of clairvoyance afforded to the dispatcher. In this paper, we shall discuss *Totally Clairvoyant Scheduling*, as modeled within the E-T-C scheduling framework. We show that this instantiation of the scheduling framework captures the central issues in a real-time flow-shop scheduling problem and design a polynomial time sequential algorithm for the same. We also introduce an error-minimizing performance metric called *Violation Degree* and establish that optimizing this metric in a Totally Clairvoyant Scheduling System is NP-Hard.

1 Introduction

Real-time scheduling is concerned with the scheduling of computer jobs which are part of periodic job-sets. The execution time of these jobs is known to vary, as we move from one period to the next [AB98]. The most common cause for this feature is the presence of input-dependent loops in the program; the time taken to execute the loop structure **for(i=1 to N)** will in general be lesser when **N=10** than when **N=1000**. A second reason for this variance is the statistical error associated with measuring execution times [LTCA89]. Consequently the traditional approach of assuming a fixed execution time for jobs [Pin95a, Bru81] may not be appropriate in real-time situations. Traditional models suffer from a second drawback, viz. the inability to specify complex constraints such as relative timing constraints. The literature on deterministic scheduling focuses almost exclusively on ready-time, deadline and precedence constraints [GLLK79, SSRB98]. In real-time applications though, there is often the necessity to constrain jobs through relationships of the form: *Start Job* J_5 *at least* 5 *units after Job* J_2 *terminates, Start Job* J_5 *within* 12 *units of Job* J_2 *starting.* Such relationships cannot be captured through precedence graphs, which are by definition, Directed Acyclic Graphs, whereas systems of relative timing constraints in real-time scheduling do contain cycles. Relative timing constraints are considered in [HL92], but with constant execution times. In this paper, we discuss a scheduling model that addresses both the above issues simultaneously.

An important feature of any scheduling model is the schedulability predicate, i.e. what it means for a job-set to be schedulable. In the case of a traditional scheduling model, the obvious definition, viz. all jobs start after their ready-times and complete before their deadlines, is sufficient. When there are relative constraints between the jobs and fixed times can be assumed for job execution, the requirement that the polyhedron defining the constraint system is non-empty, suffices. However, when the execution time of a job is an interval and there exist relative constraints, the definition of the schedulability predicate is not so obvious. There are two broad categories of providing guarantees in the presence of uncertainty, viz. Stochastic and Deterministic. A Stochastic guarantee is probabilistic in nature; for instance, in a particular application, we could declare a job-set to be schedulable if

*This research is supported in part by the Air Force Office of Scientific Research under Grant F49620-02-1-0043

it meets the imposed constraints for the most likely values of execution times [SS]. These guarantees are useful when the underlying distribution of execution time variation is well-understood and a finite (arguably small) probability of constraint violation is tolerable. Such guarantees though are always tempered with a probability (however small) that the constraints may not hold; consequently, they are not useful in the design of "hard" real-time systems in which the error probability must be zero [SR88]. Within the category of deterministic guarantees, there are 3 sub-categories, viz. Zero-Clairvoyant (Static), Partially Clairvoyant (Parametric) and Totally Clairvoyant (Co-Static) (see [Sub00, Sub02]); the usefulness of each guarantee depends upon the type of application involved. In fact, the complexity of the scheduling problem under consideration is determined in large part by the type of guarantee that we wish to provide. In this paper, our focus is on providing a polynomial time algorithm for Totally Clairvoyant Scheduling in the presence of relative timing constraints.

The rest of this paper (main body) is organized as follows: Section §2 provides a formal definition of the problem under consideration. In Section §3, we analyze the complement of the problem described in Section §2 and show that it is equivalent to checking whether a network has a negative cost cycle of a particular form. An algorithm for deciding the complement problem is outlined in Section §4; its correctness and complexity are also analyzed and we show that it converges in polynomial time. We conclude in Section §5 by summarizing our results and discussing problems for future research. Due to space limitations, some of the significant sections of this paper have been moved to the Appendix, which is organized as follows: Section §A discusses the design of a real-time system for a flow-shop problem; the discussion motivates the necessity for Totally Clairvoyant scheduling. Related approaches and models for scheduling under uncertainty are detailed in Section §B. A numerical example demonstrating the conversion of a constraint system to a constraint graph is detailed in Section §C. In Section §D, we introduce an error-minimizing metric, viz. *Violation Degree* and show that optimizing this metric over an infeasible Totally Clairvoyant Scheduling system is `NP-Hard`.

2 Statement of Problem

In this section, we detail a formal description of the problem under consideration.

2.1 Job Model

Assume an infinite time-axis divided into windows of length L, starting at time $t = 0$. These windows are called *periods* or *scheduling windows*. There is a set of non-preemptive, ordered jobs, $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ that executes in each scheduling window. The occurrences of the same job in different windows are referred to as *instances* of that job. The jobs must execute in the sequence $J_1, J_2, \ldots, J_n$.

2.2 Constraint Model

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

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

where,

- $\mathbf{A}$ is an $m \times 2.n$ rational matrix, $\vec{\mathbf{b}}$ is a rational $m-$ vector, $(\mathbf{A}, \vec{\mathbf{b}})$ is called the initial constraint matrix;
- $\mathbf{E}$ is an axis-parallel hyper-rectangle (aph) which is represented as the product of n closed intervals $[l_i, u_i]$, i.e.,

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

  We are modeling the fact that the execution time of a task can take any value in the range $[l_i, u_i]$ during actual execution and is not a fixed constant. Observe that $\mathbf{E}$ can be represented as a polyhedral system $\mathbf{M} \cdot \vec{\mathbf{e}} \leq \vec{\mathbf{m}}$, having $2 \cdot n$ constraints and n variables;
- $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$ is the start time vector of the jobs, and

- $\vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E}$ is the execution time vector of the jobs. We reiterate that $\vec{\mathbf{e}}$ could be different in different windows, i.e. different task instances of the same job could have different execution times (within the specified interval $[l_i, u_i]$ for J_i) in different scheduling windows. However, in any particular period, the execution time of the job is fixed and known at the start of the period. The execution time variables are referred to as *interval variables* since they can assume any value within a pre-specified interval;

Observation 2.1 *The jobs are non-preemptive; hence the finish time of job J_i (with start time s_i) is $s_i + e_i$. The expressive power of the scheduling framework is therefore not enhanced by introducing separate finish time variables to model constraints.*

Observation 2.2 *The ordering on the jobs is achieved by the constraint set: $s_i + e_i \leq s_{i+1}\ \forall i = 1, 2, \ldots, n-1$; these constraints are part of the $\mathbf{A}$ matrix.*

The following types of constraints are permitted:

1. Absolute constraints - These constraints are of the form: $s_i \leq a, s_i \geq a$ or $s_i + e_i \leq a, s_i + e_i \geq a$, where a is some positive integer; these relationships express the requirements on a single job. Note that deadline and ready-time constraints of traditional scheduling can be represented as absolute constraints.

2. Relative constraints - These constraints are of the form: $s_i + e_i \leq s_j + e_j + a$, $s_i + e_i \leq s_j + a$, $s_i \leq s_j + e_j + a$, $s_i \leq s_j + a$, where a is an arbitrary rational number; these relationships express relative timing (distance) constraints between the jobs J_i and J_j. As indicated, the constraints can exist between start or finish times of the jobs.

Assumption 2.1 *We assume that all constraints are between pairs of jobs, i.e. there are no absolute constraints. Since absolute constraints can be modeled as relative constraints by using an additional job J_0 with start time s_0 and execution time $e_0 \in [0, 0]$, our assumption does not limit the expressiveness of the constraint system; however it does simplify the analysis, by keeping it uniform, thereby obviating the need for considering absolute and relative constraints separately.*

Observation 2.3 *The entries in the constraint matrix $\mathbf{A}$ belong to the set $\{0, 1, -1\}$.*

Observation 2.4 *In each row of $\mathbf{A}$, the coefficient of the execution time variable tracks the coefficient of the corresponding start-time variable. If the coefficient of s_i is 0, so is the coefficient of e_i; otherwise the coefficient of e_i is either 0 or equal to the coefficient of s_i.*

Observation 2.5 *At most two start-time variables in any constraint (row) can have non-zero coefficients; if two start time variables s_i and s_j are non-zero, then $s_i = -s_j$.*

The above constraints have also been called "standard" constraints [GPS95] and are a subset of the monotone constraints discussed in [HN94]. We shall be using the terms "standard constraint" and relative constraint interchangeably, for the rest of the discussion.

2.3 Query Model

In the real-time applications that we consider such as Flow-Shop (see Section §A), it is possible to calculate with sufficient accuracy the execution times of the jobs in the current period and a few periods into the future. Totally Clairvoyant Scheduling assumes knowledge of the execution time of every job in the job-set, at the start of each scheduling window; the execution time vector may be different in different windows.

We are now in a position to formally state the Totally Clairvoyant schedulability query:

$$\mathbf{Q} : \forall \vec{\mathbf{e}} = [e_1, e_2, \ldots, e_n] \in \mathbf{E} \quad \exists \vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n] \quad \mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}} \quad ? \tag{3}$$

The focus of this paper is on the design of a polynomial time procedure to decide Query (3) (henceforth $\mathbf{Q}$).

Remark 2.1 *The combination of the Job Model, Constraint Model and Query Model represents an instance of* `<aph|stan|co-stat>` *in the* `E-T-C` *scheduling framework [Sub02].*

3 The Complement Problem

A simple technique to test the schedulability of a Totally Clairvoyant system is as follows: Let $\vec{\mathbf{e_1}}, \vec{\mathbf{e_2}}, \ldots \vec{\mathbf{e_l}}$ be the extreme points of $\mathbf{E}$. Substitute each extreme point of $\mathbf{E}$ in the constraint system $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}$ and declare $\mathbf{Q}$ to be true if and only if each of the resulting linear systems (in the start-time variables) is feasible. This is because any execution time vector $\vec{\mathbf{e'}} \in \mathbf{E}$ can be expressed as a convex combination of the extreme points of $\mathbf{E}$ and thus

$$\begin{aligned}
\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e'}}]^{\mathbf{T}} &= \mathbf{A} \cdot [\vec{\mathbf{s}}\ \sum_{i=1}^{l} \lambda_i \cdot \vec{\mathbf{a_i}}]^T,\ \sum_{i=1}^{l} \lambda_i = 1, \lambda_i \in [0,1] \\
&= \sum_{i=1}^{l} \mathbf{A} \cdot [\vec{\mathbf{s}}\ \lambda_i \vec{\mathbf{a_i}}]^T,\ \sum_{i=1}^{l} \lambda_i = 1, \lambda_i \in [0,1] \\
&= \sum_{i=1}^{l} \lambda_i \cdot \mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{a_i}}]^{\mathbf{T}},\ \sum_{i=1}^{l} \lambda_i = 1, \lambda_i \in [0,1] \\
&\leq \sum_{i=1}^{l} \lambda_i \cdot \vec{\mathbf{b}},\ \sum_{i=1}^{l} \lambda_i = 1, \lambda_i \in [0,1] \\
&= (\sum_{i=1}^{l} \lambda_i) \cdot \vec{\mathbf{b}},\ \sum_{i=1}^{l} \lambda_i = 1, \lambda_i \in [0,1] \\
&= \vec{\mathbf{b}}
\end{aligned}$$

Unfortunately such a strategy takes $\Omega(2^n)$ time, since $\mathbf{E}$ has 2^n extreme points. In this section, we shall study the complement of Query (3); our insights into the complement problem will be used to develop a polynomial time algorithm in Section §4.

Let us rewrite Query (3) as:

$$\forall \vec{\mathbf{e}} \in \mathbf{E}\ \exists \vec{\mathbf{s}}\ \mathbf{G} \cdot \vec{\mathbf{s}} + \mathbf{H} \cdot \vec{\mathbf{e}} \leq \vec{\mathbf{b}},\ \vec{\mathbf{s}} \geq \vec{\mathbf{0}}? \tag{4}$$

The complement of Query (4) is:

$$\exists \vec{\mathbf{e}} \in \mathbf{E}\ \ \forall \vec{\mathbf{s}}\ \ \mathbf{G} \cdot \vec{\mathbf{s}} + \mathbf{H} \cdot \vec{\mathbf{e}} \not\leq \vec{\mathbf{b}},\ \vec{\mathbf{s}} \geq \vec{\mathbf{0}}? \tag{5}$$

where the notation $\mathbf{A} \cdot \vec{\mathbf{x}} \not\leq \vec{\mathbf{b}}$ means that at least one of the constraints is violated. By applying Farkas' Lemma [Sch87], we know that Query (5) is true if and only if:

$$\exists \vec{\mathbf{y}}\ \exists \vec{\mathbf{e}} \in \mathbf{E}\ \ \vec{\mathbf{y}} \cdot \mathbf{G} \geq \vec{\mathbf{0}},\ \vec{\mathbf{y}} \cdot (\vec{\mathbf{b}} - \mathbf{H} \cdot \vec{\mathbf{e}}) < 0,\ \vec{\mathbf{y}} \geq \vec{\mathbf{0}}\ ? \tag{6}$$

is true.

Construct the weighted directed graph $\mathcal{G} = < \mathbf{V}, \mathbf{F}, \mathbf{c} >$ as follows:

1. Corresponding to each start time variable s_i add the vertex v_i to $\mathbf{V}$
2. Corresponding to each constraint of the form $l_p : s_i(+e_i) \leq s_j(+e_j) + k$, add a directed edge of the form $v_i \rightsquigarrow v_j$ having cost $c_{l_p} = (e_j) - (e_i) - k$.

$\mathcal{G}$ is called the constraint graph corresponding to the constraint system $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}$. For an example, see Section §C.

Remark 3.1 *In the above construction, it is possible that there exist more than one edge between the same pair of vertices; hence, technically, $\mathcal{G}$ is a constraint hyper-graph.*

Definition 3.1 *Let $p = v_i \rightsquigarrow v_j \rightsquigarrow \ldots v_q$ denote a* simple *path in $\mathcal{G}$; the* co-static cost *of p is calculated as follows:*

1. *Symbolically add up the costs on all the edges that make up the path p to get an affine function $f(p) = \vec{\mathbf{r}} \cdot \vec{\mathbf{e}} - k$, ($\vec{\mathbf{r}} = [r_1\ r_2\ \ldots\ r_n]^T, \vec{\mathbf{e}} = [e_1\ e_2\ \ldots\ e_n]^T$) for suitably chosen $\vec{\mathbf{r}}$ and k. Note that each r_i belongs to the set $\{0, 1, -1\}$.*
2. *Compute a numerical value for $f(p)$ by substituting $e_i = l_i$, if $r_i \geq 0$ and $e_i = u_i$ otherwise. This computed value is called the co-static cost of p. In other words, the co-static cost of path p is $\min_{\mathbf{E}} f(p) = \min_{\mathbf{E}}(\vec{\mathbf{r}} \cdot \vec{\mathbf{e}} - k)$.*

The co-static cost of a cycle is calculated similarly; if the cost of a cycle C in $\mathcal{G}$ is negative, then C is called a negative co-static cycle.

Theorem 3.1 *A Totally Clairvoyant Scheduling Constraint System over a system of relative constraints has a solution if and only if its constraint graph does not have a simple negative co-static cycle.*

Proof: Assume that the constraint graph has a co-static negative cycle C_1 defined by $\{v_1 \rightsquigarrow v_2 \rightsquigarrow \ldots \rightsquigarrow v_k \rightsquigarrow v_1\}$; the corresponding set of constraints in the constraint set are:

$$\begin{array}{rcl} s_1 - s_2 & \leq & f_1(e_1, e_2) \\ s_2 - s_3 & \leq & f_2(e_2, e_3) \\ \vdots & \vdots & \vdots \\ s_k - s_1 & \leq & f_k(e_k, e_1) \end{array}$$

Now, assume that there exists a solution $\vec{\mathbf{s}}$ to the constraint system. Adding up the inequalities in the above system, we get $\forall \vec{\mathbf{e}} \in \mathbf{E}\ \ 0 \leq \sum_{i=1}^{k} f_i(e_i, e_{i+1})$, where the indexes are *modulo* k. But we know that C_1 is a negative co-static cycle; it follows that $\min_{\vec{\mathbf{e}} \in \mathbf{E}} \sum_{i=1}^{k} f_i(e_i, e_{i+1}) < 0$; thus, we cannot have $\forall \vec{\mathbf{e}} \in \mathbf{E}\ \ \sum_{i=1}^{k} f_i(e_i, e_{i+1}) \geq 0$, contradicting the hypothesis.

Now consider the case, where there does not exist a negative co-static cycle. Let $\mathcal{G}_{\vec{\mathbf{e}}} = < \mathbf{V}, \mathbf{F}, \mathbf{c}_{\vec{\mathbf{e}}} >$ denote the constraint graph that results from substituting $\vec{\mathbf{e}} \in \mathbf{E}$ into the constraint system defined by System (1). It follows that for all $\vec{\mathbf{e}} \in \mathbf{E}$, $\mathcal{G}_{\vec{\mathbf{e}}}$ does not have any negative cost cycles. Hence for each $\vec{\mathbf{e}} \in \mathbf{E}$, the corresponding constraint system in the start-time variables has a solution (the vector of shortest path distances, see [CLR92]). □

Our efforts in the next section, will be directed towards detecting the existence of negative co-static cycles in the constraint graph corresponding to a Totally Clairvoyant Scheduling Constraint system; this problem is henceforth called $\mathbf{P_1}$.

4 The Shortest Paths Algorithm

In this section, we propose an algorithm for $\mathbf{P_1}$ based on the Floyd-Warshall algorithm for the All-Pairs Shortest Path problem. The key idea is to find the path of least co-static cost (shortest path) from each vertex $v_i \in \mathbf{V}$ to itself. By Theorem (3.1), we know that the constraint system is infeasible if and only if at least one of these paths has negative co-static cost.

We motivate the development of our algorithm by classifying the edges that exist between vertices in the initial constraint graph. An edge $v_i \rightsquigarrow v_j$ representing a constraint between jobs J_i and J_j must be one of the following types:

1. Type I: The weight of the edge does not depend upon either e_i or e_j, i.e. the corresponding constraint is expressed using only the start times of J_i and J_j. For instance, the edge corresponding to the constraint $s_i + 4 \leq s_j$ is a Type I edge.

2. Type II constraint: The weight of the edge depends upon both e_i and e_j, i.e. the corresponding constraint is expressed using only the finish times of J_i and J_j. For instance, the edge corresponding to the constraint $s_i + e_i + 8 \leq s_j + e_j$ is a Type II edge.

3. Type III constraint: The weight of the edge depends upon e_i, but not on e_j, i.e., the corresponding constraint is expressed using the finish time of J_i and the start time of J_j. For instance, the edge corresponding to the constraint $s_i + e_i + 25 \leq s_j$ is a Type III edge.

4. Type IV constraint: The weight of the edge depends upon e_j, but not on e_i, i.e. the corresponding constraint is expressed using the start time of J_i and the finish time of J_j. For instance, the edge corresponding to the constraint $s_i + 13 \leq s_j + e_j$ is a Type IV edge.

Lemma 4.1 *There exists at most one non-redundant $v_i \rightsquigarrow v_j$ edge of Type II.*

Proof: Without loss of generality, we assume that $i < j$, i.e. job J_i occurs in the sequence before job J_j. For the sake of contradiction, let us suppose that there exist 2 non-redundant Type II $v_i \rightsquigarrow v_j$ edges; we denote the corresponding 2 constraints as $l_1 : s_i + e_i + k_1 \leq s_j + e_j$ and $l_2 : s_i + e_i + k_2 \leq s_j + e_j$; note that they can be written as: $l_1 : s_i - s_j \leq e_j - e_i - k_1$ and $l_2 : s_i - s_j \leq e_i - e_j - k_2$. Let us say that $k_1 \geq k_2$, so that $-k_1 \leq -k_2$. We now show that l_2 can be eliminated from the constraint set without affecting its feasibility. Note that for any fixed values of e_i and e_j, l_1 dominates l_2 in the following sense: If l_1 is satisfied, then l_2 is also satisfied and hence the conjunction $l_1 \wedge l_2$ is satisfied; however if l_1 is not satisfied, then the conjunction $l_1 \wedge l_2$ is not satisfied. In other words $l_1 \Leftrightarrow l_1 \wedge l_2$ and hence l_2 can be eliminated from the constraint set, without affecting its schedulability. The case in which $i > j$ can be argued in similar fashion. □

Corollary 4.1 *There exists at most one non-redundant $v_i \rightsquigarrow v_j$ edge each of Types I, III and IV.*

Proof: Identical to the proof of Lemma (4.1). □

Corollary 4.2 *The number of non-redundant constraints in the initial constraint matrix which is equal to the number of non-redundant edges in the initial constraint graph is at most $O(n^2)$.*

Proof: It follows from Corollary (4.1) that there can exist at most 4 $i \rightsquigarrow j$ constraints and hence at most 4 $v_i \rightsquigarrow v_j$ edges between every pair of vertices v_i, v_j, $i, j = 1, 2, \ldots, n, i \neq j$. Hence the total number of edges in the initial constraint graph cannot exceed $O(8 \cdot \frac{n \cdot (n-1)}{2}) = O(n^2)$. □

We extend the taxonomy of edges discussed above to classifying paths in a straightforward way; thus a Type I path from vertex v_a to vertec v_b is a path whose cost does not depend on either e_a or e_b. Paths of Types II, III and IV are defined similarly. Algorithm (4.1) returns the type of a path $v_a \rightsquigarrow v_k \rightsquigarrow v_b$, given the types of path $v_a \rightsquigarrow v_k$ and $v_k \rightsquigarrow v_b$.

Note that Algorithm (4.1) takes $O(1)$ time.

We restrict our attention to paths and cycles of Type I; we shall see that our arguments carry over to paths and cycles of other types. As discussed above, there are at most 4 edges $v_i \rightsquigarrow v_j$, for any vertex pair (v_i, v_j). We define

$$\begin{aligned} w_{ij}(I) &= \textit{symbolic cost of the Type I edge between } v_i \textit{ and } v_j, \textit{ if such an edge exists} \\ &= \infty, \textit{ otherwise.} \end{aligned}$$

$w_{ij}(II), w_{ij}(III)$ and $w_{ij}(IV)$ are similarly defined. Note that Lemma (4.1) and Corollary (4.1) ensure that $w_{ij}(R)$ is well-defined for $R = I, II, III, IV$. By convention, $w_{ii}(R) = 0$, $i = 1, 2, \ldots, n$; $R = I, III, IV$. Note that a path of Type II from a vertex v_i to itself, is actually a Type I path!

Initialize the $n \times n \times 4$ matrix $\mathbf{W}$ as follows: $\mathbf{W}[i][j][R] = w_{ij}(R), i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, n$ $R = I, II, III, IV$. Note that the entries of $\mathbf{W}$ are not necessarily numbers; for instance, if there exists a constraint of the form $s_i + e_i + 7 \leq s_j + e_j$, then $w_{ij}(II) = -e_i + e_j - 7$.

```
Function RETURN-TYPE(v_a ⇝ v_k, v_k ⇝ v_j)
 1: if (type((v_a ⇝ v_k)) = I) then
 2:   if (type((v_k ⇝ v_b)) = I) then
 3:     return(I)
 4:   else
 5:     if (type((v_k ⇝ v_b)) = II) then
 6:       return(IV)
 7:     else
 8:       if (type((v_k ⇝ v_b)) = III) then
 9:         return(I)
10:       else
11:         return(IV)
12:       end if
13:     end if
14:   end if
15: end if
16: if (type((v_a ⇝ v_k)) = II) then
17:   if (type((v_k ⇝ v_b)) = I) then
18:     return(III)
19:   else
20:     if (type((v_k ⇝ v_b)) = II) then
21:       if ( j = i ) then
22:         return(I)
23:       else
24:         return(II)
25:       end if
26:     else
27:       if (type((v_k ⇝ v_b)) = III) then
28:         return(III)
29:       else
30:         if ( j = i ) then
31:           return(I)
32:         else
33:           return(II)
34:         end if
35:       end if
36:     end if
37:   end if
38: end if
```

Algorithm 4.1: Algorithm for determining the type of a path, given the types of its sub-paths

```
Function RETURN-TYPE(v_a ⇝ v_k, v_k ⇝ v_j)
 1: if (type((v_a ⇝ v_k)) = III) then
 2:   if (type((v_k ⇝ v_b)) = I) then
 3:     return(III)
 4:   else
 5:     if (type((v_k ⇝ v_b)) = II) then
 6:       if ( i = j ) then
 7:         return(I)
 8:       else
 9:         return(II)
10:       end if
11:     else
12:       if (type((v_k ⇝ v_b)) = III) then
13:         return(III)
14:       else
15:         if ( i = j ) then
16:           return(I)
17:         else
18:           return(II)
19:         end if
20:       end if
21:     end if
22:   end if
23: end if
24: if (type((v_a ⇝ v_k)) = IV) then
25:   if (type((v_k ⇝ v_b)) = I) then
26:     return(I)
27:   else
28:     if (type((v_k ⇝ v_b)) = II) then
29:       return(IV)
30:     else
31:       if (type((v_k ⇝ v_b)) = III) then
32:         return(I)
33:       else
34:         return(IV)
35:       end if
36:     end if
37:   end if
38: end if
```

Algorithm 4.2: Algorithm for determining the type of a path, given the types of its sub-paths (contd.)

Let $p_{ij}^k(I)$ denote the path of Type I from vertex v_i to vertex v_j having the smallest co-static cost, with all intermediate vertices in the set $\{v_1, v_2, \ldots, v_k\}$, for some $k > 0$; note that $p_{ij}^0 = w_{ij}(I)$. We refer to p_{ij}^k as the *shortest Type I Path*, from v_i to v_j, with all intermediate vertices in the set $\{v_1, v_2, \ldots, v_k\}$. Further, let $c_{ij}^k(I)$ denote the co-static cost and $d_{ij}^k(I)$ denote the corresponding symbolic cost; observe that $c_{ij}^k(I) = \min_{\mathbf{E}} d_{ij}^k(I)$ and that given $d_{ij}^k(I)$, $c_{ij}^k(I)$ can be computed in $O(n)$ time, through substitution. The quantities, $p_{ij}^k(R)$, $d_{ij}^k(R)$ and $c_{ij}^k(R)$, $R = II, III, IV$ are defined similarly.

Let us study the structure of $p_{ij}^k(I)$. We consider the following two cases.

1. Vertex v_k is not on $p_{ij}^k(I)$ - In this case, the shortest Type I path from v_i to v_j with all the intermediate vertices in $\{v_1, v_2, \ldots, v_k\}$ is also the shortest Type I path from v_i to v_j with all the intermediate vertices in $\{v_1, v_2, \ldots, v_{k-1}\}$, i.e., $p_{ij}^k(I) = p_{ij}^{k-1}(I)$ and $d_{ij}^k(I) = d_{ij}^{k-1}(I)$.

2. Vertex v_k is on $p_{ij}^k(I)$ - Let us assume that $j \neq i$, i.e., the path p_{ij}^k is not a cycle. From Algorithm (4.1), we know that one of the following must hold (See Figure (1)):

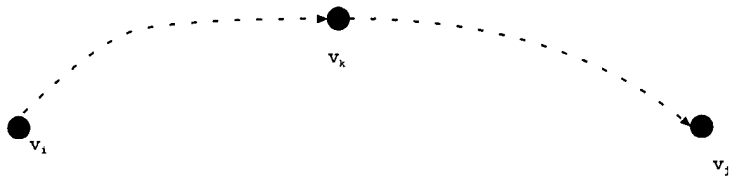

Figure 1: Shortest path of Type I from v_i to v_j through v_k

   (a) $v_i \rightsquigarrow v_k$ is of Type I and $v_k \rightsquigarrow v_j$ is of Type I - Let p_1 denote the sub-path of p_{ij}^k from v_i to v_k and let p_2 denote the sub-path of p_{ij}^k from v_k to v_j. We claim that p_1 must be the shortest Type I path from v_i to v_k with all the intermediate vertices in the set $\{v_1, v_2, \ldots, v_{k-1}\}$, i.e., $p_{ik}^{k-1}(I)$. To see this, let us assume that p_1 is not optimal and that there exists another Type I path of smaller co-static cost. Clearly this path can be combined with the existing Type I path from v_k to v_j to get a *shorter* Type I path from v_i to v_j, contradicting the optimality of $p_{ij}^k(I)$. The same argument holds for the optimality of the sub-path of p_2. This property is called the *Optimal Substructure property*. We thus have, $p_{ij}^k(I) = p_{ik}^{k-1} \bigoplus p_{kj}^{k-1}$ and $d_{ij}^k(I) = d_{ik}^{k-1} + d_{kj}^{k-1}$.

   (b) $v_i \rightsquigarrow v_k$ is of Type I and $v_k \rightsquigarrow v_j$ is of Type III - We argue in a fashion similar to the above case to derive: $p_{ij}^k(I) = p_{ik}^{k-1}(I) \bigoplus p_{kj}^{k-1}(III)$ and $d_{ij}^k(I) = d_{ik}^{k-1}(I) + d_{kj}^{k-1}(III)$.

   (c) $v_i \rightsquigarrow v_k$ is of Type IV and $v_k \rightsquigarrow v_j$ is of Type I - It follows that $p_{ij}^k(I) = p_{ik}^{k-1} \bigoplus p_{kj}^{k-1}$ and $d_{ij}^k(I) = d_{ik}^{k-1}(IV) + d_{kj}^{k-1}(I)$.

   (d) $v_i \rightsquigarrow v_k$ is of Type IV and $v_k \rightsquigarrow v_j$ is of Type III - It follows that $p_{ij}^k(I) = p_{ik}^{k-1}(IV) \bigoplus p_{kj}(III)$ and $d_{ij}^k(I) = d_{ik}^{k-1}(IV) + d_{kj}^{k-1}(III)$.

   Clearly, if v_k is on $p_{ij}^k(I)$, then

$$\begin{aligned} d_{ij}^k(I) \;=\; & \min_{\mathbf{E}}\{d_{ik}^{k-1}(I) + d_{kj}^{k-1}(I),\ d_{ik}^{k-1}(I) + d_{kj}^{k-1}(III), \\ & d_{ik}^{k-1}(IV) + d_{kj}^{k-1}(I),\ d_{ik}^{k-1}(IV) + d_{kj}^{k-1}(III)\} \end{aligned} \tag{7}$$

Remark 4.1 *$d_{ij}^k(I)$ represents the symbolic cost of the shortest Type I path from v_i to v_j, with all intermediate vertices in the set $\{v_1, v_2, \ldots, v_k\}$. Thus, the $\min_{\mathbf{E}}$ operator is used merely to select the appropriate path pairs. In particular, in the calculation of $d_{ij}^k(I)$, it does not reduce $d_{ij}^k(I)$ to a real number, although $c_{ij}^k(I)$ is a real number.*

Putting the 2 cases together, we have

$$\begin{aligned} d_{ij}^k(I) &= \min_{\mathbf{E}}\{d_{ij}^{k-1}(I),\ d_{ik}^{k-1}(I)+d_{kj}^{k-1}(I),\ d_{ik}^{k-1}(I)+d_{kj}^{k-1}(III), \\ &\quad d_{ik}^{k-1}(IV)+d_{kj}^{k-1}(I),\ d_{ik}^{k-1}(IV)+d_{kj}^{k-1}(III)\} \end{aligned} \tag{8}$$

Now consider the case that the path $p_{ij}^k(I)$ is a cycle, i.e., $j = i$. From Algorithm (4.1), we know that one of the following must hold:

1. $v_i \rightsquigarrow v_k$ is of Type I and $v_k \rightsquigarrow v_i$ is of Type I - This case has been handled above.
2. $v_i \rightsquigarrow v_k$ is of Type II and $v_k \rightsquigarrow v_i$ is of Type II, i.e., $d_{ii}^k(I) = d_{ik}^{k-1}(II) + d_{ki}^{k-1}(II)$.
3. $v_i \rightsquigarrow v_k$ is of Type II and $v_k \rightsquigarrow v_i$ is of Type IV, i.e., $d_{ii}^k(I) = d_{ik}^{k-1}(II) + d_{ki}^{k-1}(IV)$.
4. $v_i \rightsquigarrow v_k$ is of Type III and $v_k \rightsquigarrow v_i$ is of Type II, i.e., $d_{ii}^k(I) = d_{ik}^{k-1}(III) + d_{ki}^{k-1}(II)$.
5. $v_i \rightsquigarrow v_k$ is of Type III and $v_k \rightsquigarrow v_i$ is of Type IV, i.e., $d_{ii}^k(I) = d_{ik}^{k-1}(III) + d_{ki}^{k-1}(IV)$.

Note that the case $k = 0$, corresponds to the existence (or lack thereof) of a Type I edge from v_i to v_j. Thus, the final recurrence relation to calculate the cost of p_{ij}^k is:

$$\begin{aligned} d_{ij}^k(I) &= w_{ij}(I),\ if\ k = 0 \\ &= \min_{\mathbf{E}}\{d_{ik}^{k-1}(I)+d_{ki}^{k-1}(I),\ d_{ik}^{k-1}(II)+d_{ki}^{k-1}(II),\ d_{ik}^{k-1}(II)+d_{ki}^{k-1}(IV), \\ &\quad d_{ik}^{k-1}(III)+d_{ki}^{k-1}(II),\ d_{ik}^{k-1}(III)+d_{ki}^{k-1}(IV))\},\ if\ j = i \\ &= \min_{\mathbf{E}}\{d_{ij}^{k-1}(I),\ d_{ik}^{k-1}(I)+d_{kj}^{k-1}(I),\ d_{ik}^{k-1}(I)+d_{kj}^{k-1}(III), \\ &\quad d_{ik}^{k-1}(IV)+d_{kj}^{k-1}(I),\ d_{ik}^{k-1}(IV)+d_{kj}^{k-1}(III)\},\ otherwise \end{aligned} \tag{9}$$

Using similar analyses, we derive recurrence relations for $d_{ij}^k(R),\ R = II, III, IV$ as follows:

$$\begin{aligned} d_{ij}^k(II) &= w_{ij}(II),\ if\ k = 0 \\ &= \min\{d_{ij}^{k-1}(II),\ d_{ik}^{k-1}(II)+d_{kj}^{k-1}(II),\ d_{ik}^{k-1}(II)+d_{kj}^{k-1}(IV), \\ &\quad d_{ik}^{k-1}(III)+d_{kj}^{k-1}(II),\ d_{ik}^{k-1}(III)+d_{kj}^{k-1}(IV)\},\ otherwise \end{aligned} \tag{10}$$

$$\begin{aligned} d_{ij}^k(III) &= w_{ij}(III),\ if\ k = 0 \\ &= \min\{d_{ij}^{k-1}(III),\ d_{ik}^{k-1}(II)+d_{kj}^{k-1}(I),\ d_{ik}^{k-1}(II)+d_{kj}^{k-1}(III) \\ &\quad d_{ik}^{k-1}(III)+d_{kj}^{k-1}(I),\ d_{ik}^{k-1}(III)+d_{kj}^{k-1}(III)\},\ otherwise \end{aligned} \tag{11}$$

$$\begin{aligned} d_{ij}^k(IV) &= w_{ij}(IV),\ if\ k = 0 \\ &= \min\{d_{ij}^{k-1}(IV),\ d_{ik}^{k-1}(I)+d_{kj}^{k-1}(II),\ d_{ik}^{k-1}(I)+d_{kj}^{k-1}(IV), \\ &\quad d_{ik}^{k-1}(IV)+d_{kj}^{k-1}(II),\ d_{ik}^{k-1}(IV)+d_{kj}^{k-1}(IV)\},\ otherwise \end{aligned} \tag{12}$$

Algorithm (4.3) summarizes the above discussion on the identification of a negative co-static cycle in the constraint graph $\mathcal{G}$. We note that $\mathbf{D}_{ij}^n(I)$ represents the shortest Type I $v_i \rightsquigarrow v_j$ path with all the intermediate vertices in the set $\{v_1, v_2, \ldots, v_n\}$, i.e., it is the shortest Type I $v_i \rightsquigarrow v_j$ path. EVAL-LOOP() evaluates the co-static cost of each of the diagonal entries and declares the $\mathcal{G}$ to be co-static negative cycle free, if all entries have non-negative cost. Further, we need not consider the case $j = i$ separately, in the computations of $d_{ij}^k(R),\ R = II, III, IV$.

Function DETECT-COSTATIC-NEGATIVE-CYCLE($\mathcal{G}$)

1: Initialize $\mathbf{W}$.
2: Set $\mathbf{D}^0 = \mathbf{W}$.
3: **for** ($k = 1$ **to** n) **do**
4: {We are determining $\mathbf{D}^k$}
5: **for** ($i = 1$ **to** n) **do**
6: **for** ($j = 1$ **to** n) **do**
7: Compute $\mathbf{D}^k_{ij}(I)$, $\mathbf{D}^k_{ij}(II)$, $\mathbf{D}^k_{ij}(III)$, $\mathbf{D}^k_{ij}(IV)$ using the relations (9), (10), (11), (12).
8: **end for**
9: **end for**
10: **end for**
11: **for** ($i = 1$ **to** n) **do**
12: **for** ($R = I$ **to** IV) **do**
13: **if** (EVAL-LOOP($\mathbf{D}^n_{ii}(R)$) < 0) **then**
14: **return**(**true**)
15: **end if**
16: **end for**
17: **end for**
18: **return**(**false**)

Algorithm 4.3: Algorithm for identifying negative co-static cycles in the constraint graph

4.1 Complexity

The complexity of Algorithm (4.3) is determined by Step (7 :) within the $O(n^3)$ triple loop. It is easy to see that if the symbolic costs are stored in arrays, Step (7 :) can be implemented in $O(n)$ time; it follows that Steps ($1 : -10$:) can be implemented in time at most $O(n^4)$. Step (13 :) takes time at most $O(n)$ and hence Steps ($11 : -18$:) take time at most $O(n^2)$.

4.2 Dispatching

Having decided the schedulability query affirmatively, it is the task of the dispatcher to compute the start-time vector for each window, given the execution time vector for that window. Let $\vec{\mathbf{e_p}}$ denote the execution time vector in the current scheduling window. Substituting the components of $\vec{\mathbf{e_p}}$ in Query (3), we get a linear system of difference constraints $\mathbf{G} \cdot \vec{\mathbf{s}} \leq \vec{\mathbf{b'}}$, in the $\vec{\mathbf{s}}$ variables only. Using the techniques from [CLR92], we know that a feasible solution to such a system can be computed in $O(m \cdot n)$ time.

Theorem 4.1 *The schedulability query for an instance of a co-static scheduling problem with n jobs and m strict relative (standard) constraints can be decided in $O(n^4)$ time, while dispatching can be effected in $O(m \cdot n)$ time.*

5 Conclusions

In this paper, we demonstrated the expressiveness of `E-T-C` scheduling framework by using it to model a real-time flow shop scheduling problem. The study of clairvoyant scheduling in the literature thus far, had been restricted to ready-time and deadline constraints; to the best of our knowledge our work is the first of its kind to consider more general relationships in the form of relative timing constraints. Our work establishes that even in the presence of these non-trivial constraints, the scheduling and dispatching problems can be decided in polynomial time ($O(n^4)$).

We also studied the problem of optimizing an error-minimizing metric, viz., *Violation Degree* (See Section §D); our analysis demonstrated that this optimization problem is `NP-Hard`. From an implementation perspective, the algorithms we present are straightforward and use very simple data structures; we are currently engaged in the task of studying the performance of the same.

Some of the interesting open theoretical questions concern:

1. The complexity of Totally Clairvoyant Scheduling under partial orders,
2. The complexity of Totally Clairvoyant Scheduling in the presence of more general constraints such as *Network Constraints* [Sub01],
3. The exact complexity of Violation Degree,
4. The complexity of finding a schedule minimizing metrics such as *Sum of Start Times* and *Sum of Completion Times.*

6 Acknowledgements

We thank Vijaya Ramachandran for helpful discussions.

A Motivation

In this section, we show that a practical real-time scheduling problem can be captured as an instance of <aph|stan|co-stat>.

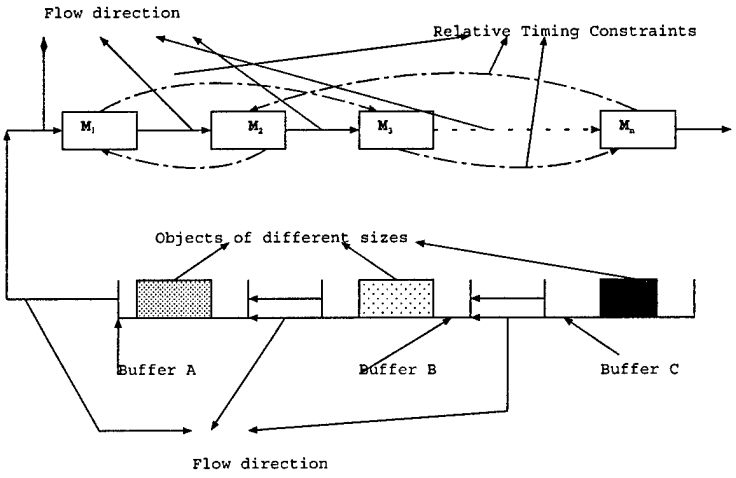

Figure 2: Bounded-buffer Flow Shop

Figure (2) represents a bounded buffer flow shop. The flow shop consists of n machines M_1 through M_n and one or more feed-buffers (or feeders). In our example these buffers are A, B and C. Objects to be tooled also called *jobs* are placed in these buffers. The timeline on which the flow shop operates is divided into equal length portions called *periods.* At the start of each period, the job in the each buffer moves to the buffer ahead of it, while the job in the first buffer (Buffer A) enters machine M_1. Within the period, the job moves sequentially from machine M_i to machine M_{i+1}, respecting the relative timing constraints (represented by the curved, broken arrows) and finally exits at machine M_n before the end of the period. This process is repeated in every period with new objects continuously entering the flow at the last buffer [1]. Let s_i denote the time at which the machine M_i begins operating on the current job and let e_i denote the time it takes to complete its operation on the job. Relative timing constraints are used to capture relationships such as heating and cooling requirements; for instance the requirement that the object should wait 5 units of time after exiting machine M_1, before it enters machine M_2 is represented as: $s_2 \geq s_1 + e_1 + 5$, where s_2 is the time at which the object enters M_2 and $s_1 + e_1$ is the time at which it exits M_1.

[1] This example is taken from [Pin95b].

Observation A.1 *Each machine in the flow shop has well-defined lower and upper limits on the sizes of the jobs it can handle. Let sz_{li} and sz_{ui} denote the lower and upper bounds on job-size insofar as machine M_i is concerned; given these bounds it is straightforward to compute the bounds l_i and u_i on e_i, i.e. the operation time of machine M_i must lie in the interval $[l_i, u_i]$ for all jobs. We can similarly calculate the execution time range for each machine in the flow shop.*

Observation A.2 *When a job is loaded into the buffer (Buffer C), its size is known; consequently the operation time of each machine on that job, i.e. $e_i, i = 1, 2, \ldots, n$ can be calculated, even before it enters machine M_1. In other words, the look-ahead factor k is 3.*

DESIGN PROBLEM: Given

1. Timing constraints between the flow shop machines,

2. Lower and Upper bounds on the operation time of by each machine,

Does there exist a valid schedule i.e., a schedule that respects the timing constraints, for any job with size sz, where $sz_{li} \leq sz \leq sz_{ui}, i = 1, 2, \ldots, n$?

The flow-shop example in this section is easily modeled as a Totally Clairvoyant scheduling problem, with the machines acting as the jobs with variable execution times.

B Related Work

In this section, we contrast the Totally Clairvoyant model with two other models in the real-time scheduling literature that attempt to model impreciseness, viz. the *imprecise computation model* and the *error-task model.*

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

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

Orthogonal approaches to the issues of clairvoyance and speed have been discussed extensively in [KP00] and [KP95]. A number of online scheduling models with and without clairvoyance are discussed in [Sga98] and [FW98]; however their primary concern is optimizing performance metrics in the presence of multiple processors, whereas we are concerned with checking feasibility in a uniprocessor scheduling problem with relative timing constraints.

C Constraint System to Constraint Graph Example

Example (1): Consider a job-set $\mathcal{J}$ with 4 jobs $\{J_1, J_2, J_3, J_4\}$ having execution times $e_1 \in [4, 8], e_2 \in [6, 11], e_3 \in [10, 13], e_4 \in [3, 9]$ and the following constraints:

1. *J_4 finishes with 12 units of J_3 finishing:* $s_4 + e_4 \leq s_3 + e_3 + 12$

2. *J_4 starts no earlier than 18 units of J_2 finishing:* $s_2 + e_2 + 18 \leq s_4$

3. *J_3 finishes within* 31 *units of J_1 finishing:* $s_3 + e_3 \leq s_1 + e_1 + 31$

4. *J_1 finishes before J_2 starts:* $s_1 + e_1 \leq s_2$

5. *J_2 finishes before J_3 starts:* $s_2 + e_2 \leq s_3$

6. *J_3 finishes before J_4 starts:* $s_3 + e_3 \leq s_4$

Putting the constraints in matrix form, we get

$$(\mathbf{A}) \begin{bmatrix} 0 & 0 & -1 & 1 & 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot [\vec{s}\ \vec{e}]^T \leq \begin{bmatrix} 12 \\ -18 \\ 31 \\ 0 \\ 0 \\ 0 \end{bmatrix} (\vec{b}) \tag{13}$$

The constraint graph $\mathcal{G}$ corresponding to the constraint system $\mathbf{A} \cdot [\vec{s}\ \vec{e}]^{\mathbf{T}} \leq \vec{b}$ is given in Figure (3).

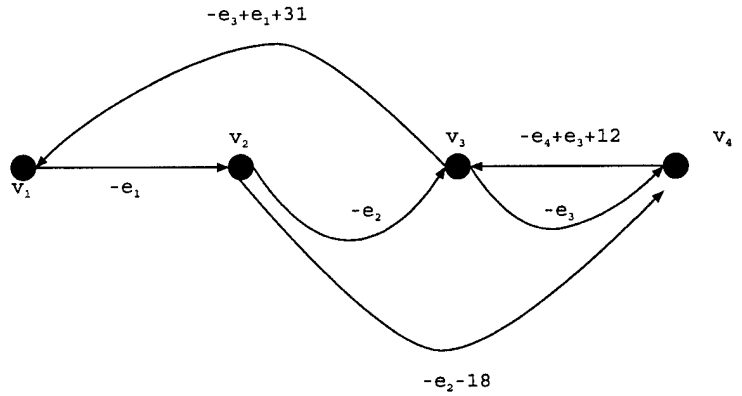

Figure 3: Conversion from Constraint System to Constraint Graph

D Performance Metric Optimization

Thus far, our focus has been on finding a feasible solution to a Totally Clairvoyant Scheduling system involving standard constraints. However, it is often the case, that real-time designers require solutions optimizing certain Performance Metrics. Typical Performance Metrics include *Makespan*, *Sum of Start times* and *Sum of Completion times*. Definitions of these and other metrics are provided in [Pin95a].

The performance metrics mentioned above assume that the constraint system has a feasible schedule and the goal is to find a schedule that minimizes a criterion. In traditional scheduling, designers are also interested in performance metrics such as *Lateness*, *Tardiness* and *Unit Penalty*, wherein the assumption of feasibility is relaxed [Pin95a]. These metrics are concerned with minimizing the *degree of infeasibility* of the constraint system and are called an *error-minimizing* metrics. If the constraint system is feasible, the minimum value of an error-minimizing metric is clearly 0. Error-minimizing performance metrics are also of interest in real-time scheduling; in Section §D.1 we define a metric called *Violation Degree* and analyze its complexity.

D.1 Violation Degree

Consider a Totally Clairvoyant scheduling system with the imposed constraints described by $\mathbf{A} \cdot [\vec{s}\ \vec{e}]^{\mathbf{T}} \leq \vec{b}$, as discussed in Section §2. We assume that the constraint set is infeasible, from a Totally Clairvoyant perspective.

It follows that there exists a set of execution time vectors $\mathbf{V_E} \subseteq \mathbf{E}$, which violate the constraint system; this set is called the *Violation Set* of the input constraint system.

Rewrite the input constraint set as: $\mathbf{G} \cdot \vec{\mathbf{s}} + \mathbf{H} \cdot \vec{\mathbf{e}} \leq \vec{\mathbf{b}}$. Let $\vec{\mathbf{e}'}$ be any vector in the set $\mathbf{V_E}$. It follows that the constraint system $\mathbf{G} \cdot \vec{\mathbf{s}} \leq \vec{\mathbf{b}} - \mathbf{H} \cdot \vec{\mathbf{e}'}$ is infeasible. Let $\vec{\mathbf{b}_{\vec{\mathbf{e}'}}} = \vec{\mathbf{b}} - \mathbf{H} \cdot \vec{\mathbf{e}'}$; note that the infeasibility of the linear system $\mathbf{G} \cdot \vec{\mathbf{s}} \leq \vec{\mathbf{b}_{\mathbf{e}'}}$ implies that at every start-time vector $\vec{\mathbf{s}}$, one or more constraints are violated. Let $\vec{\mathbf{s}_{\vec{\mathbf{e}'}}}$ denote the start-time vector at which the least number of constraints are violated and let $n_{\vec{\mathbf{e}'}}$ denote the number of constraints violated by $\vec{\mathbf{s}_{\vec{\mathbf{e}'}}}$; $\vec{\mathbf{s}_{\vec{\mathbf{e}'}}}$ and $n_{\vec{\mathbf{e}'}}$ are respectively called the violation vector and the violation degree at the execution time vector $\vec{\mathbf{e}'}$. The Violation Degree (V_d) of the Totally Clairvoyant system $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}},\ \vec{\mathbf{e}} \in \mathbf{E}$ is defined as the maximum violation degree over all execution time vectors $\vec{\mathbf{e}} \in \mathbf{V_E}$. We note that for all execution time vectors $\vec{\mathbf{e}} \in \mathbf{E} - \mathbf{V_E}$, $n_{\vec{\mathbf{e}}} = 0$, since the constraint system is satisfied at those execution time vectors. We can thus write:

$$V_d(\mathbf{A}, \vec{\mathbf{b}}, \mathbf{E}) = \max_{\vec{\mathbf{e}} \in \mathbf{V_E}} n_{\vec{\mathbf{e}}}, \tag{14}$$

where,

$$n_{\vec{\mathbf{e}}} = \min_{\vec{\mathbf{s}}} (\#\ of\ constraints\ violated\ in\ \mathbf{G} \cdot \vec{\mathbf{s}} \leq \vec{\mathbf{b}_{\vec{\mathbf{e}}}})$$

From an error-minimizing perspective, we wish to know if the Violation Degree of the Constraint System is bounded. Accordingly, the query that we are interested in is:

$$V_d(\mathbf{A}, \vec{\mathbf{b}}, \mathbf{E}) \leq k? \tag{15}$$

Note that the defining equation (14) assumes that the set $\mathbf{V_E}$ is provided; this may not always be a valid assumption, since there could be exponentially many violation vectors and it is not clear in such cases, $\mathbf{V_{\vec{e}}}$ can be calculated in polynomial time. To account for this nuance, we define Violation Degree as:

$$V_d(\mathbf{A}, \vec{\mathbf{b}}, \mathbf{E}) = \max_{\vec{\mathbf{e}} \in \mathbf{E}} n_{\vec{\mathbf{e}}}. \tag{16}$$

Further, given an execution time vector $\vec{\mathbf{e}'} \in \mathbf{V_E}$, $n_{\vec{\mathbf{e}'}}$ corresponds to the minimum number of constraints, whose removal makes the constraint system feasible.

The rest of this section is devoted to analyzing the complexity of deciding the bound query representing the Violation Degree metric.

Theorem D.1 *Query (15) is* `NP-Hard`*, even when all jobs have the same execution time, i.e.,* $[0, 0]$ *and* $|\mathbf{E}| = 1$.

The proof of Theorem (D.1) requires the development of a few concepts in Network Design.

Let $\mathbf{G} =< V, E, c >$ denote a weighted, directed graph, with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, edge set E, with $e_{ij} \in E$ representing the edge $v_i \rightsquigarrow v_j$ and cost function $c : E \rightarrow \mathcal{Z}$. The negative feedback arc set problem is defined as follows: Is there a set of edges, $E_1 \subseteq E$, $|E_1| \leq k$, such that $\mathbf{G_1} =< V, E - E_1, c >$ does not have a negative cost cycle?

Lemma D.1 *The negative feedback arc set problem [NFAS] is* `NP-complete`.

Proof: [NFAS] is clearly in `NP`, since an NDTM can guess the set E_1 and verify in polynomial time (using the Bellman-Ford algorithm, for instance) that the subgraph obtained by the elimination of E_1 does not have a negative cost cycle. To show `NP-Hardness`, we reduce the `NP-complete` Feedback arc set problem [FAS] to [NFAS]. An instance of [FAS] is $(\mathbf{G} =< V, E >, k)$, where $\mathbf{G}$ is an unweighted, directed graph and $k \leq n \in Z^+$. The query is: Is there a subset $E_1 \subseteq E$, $|E_1| \leq k$, such that the subgraph $\mathbf{G_1} =< V, E - E_1 >$ does not have a cycle? The corresponding instance of [NFAS] is $(\mathbf{G'} =< V, E, c' >, k)$, where c' assigns -1 to every edge in E. Observe that every cycle in $\mathbf{G}$ can be identified with a negative cost cycle in $\mathbf{G'}$ through the same set of vertices and vice versa. □

Lemma D.2 *The restriction of [NFAS] to complete graphs, i.e., [NFASC], is* `NP-complete`.

Proof: Let $(\mathbf{G} =< V, E, c >, k)$ be an instance of [NFAS]. The corresponding instance of [NFASC] is $(\mathbf{G}' =< V, E', c' >, k)$, where, $E' = \{e_{ij} : i, j = 1, 2, \ldots, n\}$ and c' is defined as

$$\begin{aligned} c'_{ij} &= c_{ij}, \; if \; e_{ij} \in E \\ &= M >> 0, \; otherwise, \end{aligned}$$

such that $M + c'_{ij} = M$, for all $e'_{ij} \in E'$. Note that an edge with weight M cannot be part of a negative cost cycle. Thus every negative cost cycle in $\mathbf{G}$ can be identified with a negative cost cycle in $\mathbf{G}'$ through the same set of vertices and vice versa. □

Let $\{v_1, v_2, \ldots, v_n\}$ be the vertex labels of a directed, weighted, complete graph $\mathbf{G} =< V, E, c >$; the graph is said to be *vertex-ordered*, if for all edges $e_{ij} = v_i \rightsquigarrow v_j$, $c_{ij} < 0$ if $j > i$ and $c_{ij} \geq 0$, if $j < i$, i.e. all edges from a vertex to vertices with label values higher than its own have negative cost, while edges to vertices with label values lower than its own have non-negative cost.

Lemma D.3 *The restriction of [NFASC] to vertex-ordered graphs, i.e. [NFASCVO], is* `NP-complete`.

Proof: Let $(\mathbf{G} =< V, E, c >, k)$ be an instance of [NFASC]. An instance of [NFASCVO] is created using the Algorithm (D.1).

Function INSTANCE-TRANSFORM($\mathbf{G} =< V, E, c >, k$)

1: **for** $(i = 1$ **to** $n)$ **do**
2: Let $S_i = \{e_{ij}, j > i : c_{ij} > 0\}$
3: **if** $(|S_i| > 0)$ **then**
4: Create a vertex v_{n+i}
5: **for** (each edge $e_{ij} \in S_i$) **do**
6: Add an edge $v_i \rightsquigarrow v_{n+i}$ of cost -1 and an edge $v_{n+i} \rightsquigarrow v_j$ of cost $(c_{ij} + 1)$
7: Delete edge e_{ij}
8: **end for**
9: **end if**
10: Let $D_i = \{e_{ij}, j < i : c_{ij} < 0\}$
11: **if** $(|D_i| > 0)$ **then**
12: Create a vertex v_{-i}
13: **for** (each edge $e_{ij} \in D_i$) **do**
14: Add an edge $v_i \rightsquigarrow v_{-i}$ of cost 1 and an edge $v_{-i} \rightsquigarrow v_j$ of cost $(c_{ij} - 1)$
15: Delete edge e_{ij}
16: **end for**
17: **end if**
18: **end for**
19: Let $V' = V \cup \{v_{-i} : i = 1, 2, \ldots, n\} \cup \{v_{n+i} : i = 1, 2, \ldots, n\}$
20: **if** (no edge exists between $v_i, v_j \in V'$) **then**
21: Add an edge e_{ij} of cost M, if $i > j$ and $-M$, if $i < j$ {We are making the new graph complete!}
22: **end if**
23: Let c' be the new cost function as defined above.
24: Let $E' = E \cup \{new \; edges \; are \; created\} - \{edges \; that \; are \; deleted\}$
25: **return**$(\mathbf{G}' =< V', E', c' >, k)$

Algorithm D.1: Algorithm for transforming an instance of [NFASC] to an instance of [NFASCVO]

We use the convention that $-M + a = -M$, for any $a \in \mathcal{Z} \cup \{-M, M\}$. The query associated with the [NFASCVO] problem is: Is there a subset of edges, of cardinality k whose deletion, removes all the cycles of cost a, for all $-M < a < 0$? Note that any negative cost cyle in $\mathbf{G}'$ having cost strictly greater than $-M$ is also a negative cost cycle in $\mathbf{G}$ and vice versa. □

We are now ready to prove Theorem (D.1).

Proof (Theorem (D.1)): Let $(\mathbf{G} =< V, E, c >, k)$ be an instance of [NFASCVO]. We construct the Totally Clairvoyant Scheduling System $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}, \vec{\mathbf{e}} \in \mathbf{E}$, where,

1. $\mathbf{E} = \Pi_{i=1}^{n}[0, 0]$,
2. Corresponding to edge $e_{ij} = v_i \rightsquigarrow v_j$ in $\mathbf{G}$, we create a constraint $s_i - s_j \leq c_{ij}$; these constraints are part of the constraint matrix $\mathbf{A} \cdot [\vec{\mathbf{s}}\ \ \vec{\mathbf{e}}]^{\mathbf{T}} \leq \vec{\mathbf{b}}$.

Since $\mathbf{G}$ is vertex-ordered, the jobs are ordered, as required by our model. It is straightforward to see that every negative cost cycle in $\mathbf{G}$ passing through a set of vertices $\{v_{i_1}, v_{i_2}, \ldots, v_{i_p}\}, 2 \leq p \leq n, \{i_1, i_2, \ldots, i_p\} \in \{1, 2, \ldots, n\}$, can be identified with a Minimum Unsatisfiable Subset (MUS) through the jobs $\{s_{i_1}, s_{i_2}, \ldots, s_{i_p}\}$. Thus finding the minimum number of constraints whose removal makes the system feasible is equivalent to finding the minimum number of edges in $\mathbf{G}$, whose removal gets rid of all the negative cost cycles. The theorem follows. □

Remark D.1 *From the perspective of Computational Complexity, the decision problem associated with violation degree belongs to the class Π_2 **P** in the polynomial hierarchy [GJ79]. We also note that if we had chosen to define Violation Degree as:*

$$V_d(\mathbf{A}, \vec{\mathbf{b}}, \mathbf{E}) = \min_{\vec{\mathbf{e}} \in \mathbf{E}} n_{\vec{\mathbf{e}}}$$

the same proof would have worked, since the problem is `NP-Hard`*, even when there is precisely one point in the execution time domain* **E***!*

References

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

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

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

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

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

[FW98] Amos Fiat and Gerhard Woeginger. *Online algorithms: the state of the art*, volume 1442 of *Lecture Notes in Computer Science.* Springer-Verlag Inc., New York, NY, USA, 1998.

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

[GLLK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Mathematics*, 5:287–326, 1979.

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

[HL92] C. C. Han and K. J. Lin. Scheduling real-time computations with separation constraints. *Information Processing Letters*, 12:61–66, May 1992.

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

[KP95] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 214–223, Los Alamitos, October 1995. IEEE Computer Society Press.

[KP00] Kalyanasundaram and Pruhs. Fault-tolerant real-time scheduling. *ALGRTHMICA: Algorithmica*, 28, 2000.

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

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

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

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

[Pin95a] M. Pinedo. *Scheduling: theory, algorithms, and systems.* Prentice-Hall, Englewood Cliffs, 1995.

[Pin95b] M. Pinedo. *Scheduling: theory, algorithms, and systems*, chapter 5. In [Pin95a], 1995.

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

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

[Sga98] Jiri Sgall. *Online Scheduling*, chapter 8, pages 196–203. Volume 1442 of *Lecture Notes in Computer Science* [FW98], 1998.

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

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

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

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

[Sub01] K. Subramani. Parametric scheduling for network constraints. In *Proceedings of the* 8^{th} *Annual International Computing and Combinatorics Conference*, Lecture Notes in Computer Science. Springer-Verlag, August 2001.

[Sub02] K. Subramani. A specification framework for real-time scheduling. In *Proceedings of the* 29^{th} *Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, Lecture Notes in Computer Science. Springer-Verlag, November 2002.

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

Optimal Length Tree-Like Resolution Refutations for 2SAT formulas

K. SUBRAMANI
LDCSEE
West Virginia University

In this paper, we exploit the graphical structure of 2SAT formulas to show that the shortest tree-like resolution refutation of an unsatisfiable 2SAT formula can be determined in polynomial time.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic**]: Computational Logic

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Resolution, tree-like proofs, 2SAT formulas

1. INTRODUCTION

The principal goal of research in proof complexity theory, is to obtain non-trivial lower bounds on the lengths of proofs of propositional tautologies (alternatively, lengths of refutations of contradictions), in non-trivial propositional proof systems. From a theoretical perspective, the study of proof complexity has fundamental implications for complexity theory; it was shown in [Cook and Reckhow 1973] that there exists a propositional proof system giving rise to short (polynomial-sized) proofs for all tautologies if and only if NP=coNP. Thus, obtaining super-polynomial lower bounds on the size of proofs of (propositional) tautologies in increasingly stronger proof systems leads us in the direction of separating NP from coNP. Note that the problem of finding the shortest proof of a tautology is polynomially equivalent to the problem of finding the shortest refutation of a contradiction and it is the latter problem that we shall be focussing on. From a practical perspective, Automated Reasoning is an integral part of Real-Time Databases [Bestavros and Fay-Wolfe 1997]. Queries to the database are converted into equivalent tautologies which in turn are converted to refutation problems; the search for efficient procedures to find the shortest length resolution refutation of an unsatisfiable formula is thus directly connected to practical concerns.

In this paper, we are concerned with finding the shortest resolution refutations of contradictions represented as 2SAT formulas.

[Haken 1985] describes one of the earliest results showing an exponential lower

Author's address: K. Subramani, LDCSEE, WVU, Morgantown, WV - 26508, ksmani@csee.wvu.edu

bound for a proof system; they proved that any refutation of the pigeonhole principle using resolution would generate exponentially many clauses. A similar lower bound using Bounded Arithmetic proof systems was proved in [Razborov 1996]. Stronger and simpler resolution lower bounds for the pigeonhole principle were derived in [Beame and Pitassi 1996]. In [Achlioptas et al. 2001], an example of a sharp threshold is given; they showed that randomly generated "dense" 3SAT formulas satisfying certain properties almost certainly require resolution proofs of exponential size. Note that under the assumption that NP $\neq$ co-NP, 3SAT formulas cannot have polynomial sized refutations. Exponential lower bounds for propositional formulas using cutting plane proof systems were first shown in [Pudlák 1997]; applications of cutting plane theory to propositional proof systems are also discussed in [Bockmayr and Eisnbrand]. Exhaustive surveys of Propositional proof complexity can be found in [Beame and Pitassi 1998] and [Urquhart 1995]. In [Clegg et al. 1996], a new propositional proof system called the Groebner proof system was introduced; their algorithm can be used to approximate the shortest length proof of a tautology and runs in time polynomial in the length of the shortest proof (which could be exponential in the size of the input formula). [Ben-Sasson and Wigderson 2001] relates the length of a resolution refutation to its width, where the width of a proof is defined as the maximum number of literals in any clause of the proof.

From an optimization perspective, [Alekhnovich et al. 1998] argues NP-Hardness and inapproximability results for a number of proof systems; the weakest proof system that they consider is Horn resolution, i.e. resolution as applied to a HornSAT formula. Their paper improves on the work in [Iwama 1997], which showed that finding the shortest resolution refutation to an arbitrary CNF formula is NP-Hard. [Iwama and Miyano 1995] considered a fragment of resolution called *Read-Once Resolution* and showed that the problem of checking whether a CNF formula has a Read-Once Resolution Refutation is NP-complete.

Note that the resolution refutation of an unsatisfiable formula can be either tree-like or dag-like (See Section §2). In this paper, we show that the shortest tree-like resolution refutation of an unsatisfiable 2SAT formula can be determined in polynomial time. Our work has the effect of classifying natural classes of Boolean formulas in the following resolution refutation complexity hierarchy:

(1) 3SAT formulas - There exist formulas, for instance, the formula encoding the pigeon-hole principle, that do not have short resolution refutations in either the tree-like or the dag-like proof systems [Haken 1985].
(2) HornSAT formulas - A short resolution refutation exists for both tree-like and dag-like proof systems; however, obtaining even a linear approximation to the length of the shortest refutation, in either case is NP-Hard [Alekhnovich et al. 1998].
(3) 2SAT formulas - A short resolution refutation (tree-like and dag-like) exists and the exact length of the shortest tree-like refutation can be determined in polynomial time.

2. ALGORITHMS AND COMPLEXITY

Let $C = C_1 \wedge C_2 \wedge \ldots C_m$ be an unsatisfiable 2SAT formula, defined on the variable set $X = \{x_1, x_2, \ldots, x_n\}$. Note that a formula C is unsatisfiable, if and only if the

empty clause or $\Box$ can be derived from C using resolution; further $\Box$ can be derived from C if and only if both $\{x_i\}$ and $\{\bar{x}_i\}$ can be derived from C, using resolution, for some variable x_i.

Definition 2.1. The size of a tree-like refutation of an unsatisfiable formula C is the number of clauses in the tree-like refutation of C; the shortest tree-like refutation of C is the tree-like refutation having the fewest number of clauses.

Observe that converting a dag-like refutation to a tree-like refutation may increase the size of the refutation.

For every 2SAT formula C, there exists an implication graph $\mathbf{G_C}$; see [Aspvall et al. 1979] for the details of constructing $\mathbf{G_C}$ from C.

THEOREM 2.2. *The 2SAT formula C is unsatisfiable if and only if there exists a directed path from the vertex corresponding to the literal x_i to the vertex corresponding to the complement literal $\bar{x}_i$ and vice versa, in $\mathbf{G_C}$.*

Proof: See [Aspvall et al. 1979].

Since the given formula C is unsatisfiable, as per Theorem (2.2), there must exist one or more variables x_i, such that there is a directed closed walk (note that both vertices and edges can repeat) containing x_i and $\bar{x}_i$. Each of these directed, closed walks provides evidence that C is unsatisfiable, in that it provides a derivation of $\Box$, i.e., a resolution refutation of C. A path from vertex x_i to vertex $\bar{x}_i$ in $\mathbf{G_C}$ corresponds to a resolution derivation of $\{\bar{x}_i\}$ from C; likewise a path from vertex $\bar{x}_i$ to vertex x_i in $\mathbf{G_C}$ corresponds to a resolution derivation of $\{x_i\}$. In fact, every candidate for a shortest resolution derivation of $\{\bar{x}_i\}$ corresponds to tracing out a path from vertex x_i to vertex $\bar{x}_i$; likewise for derivations of $\{x_i\}$.

LEMMA 2.3. *Let $\mathbf{T_C}$ denote the shortest tree-like resolution refutation of C and let $\Box$ be derived by resolving $\{x_i\}$ and $\{\bar{x}_i\}$ in $\mathbf{T_C}$. Then $\mathbf{T_C}$ also encodes the shortest length tree-like proof of $\{x_i\}$ and the shortest length tree-like proof of $\{\bar{x}_i\}$.*

Proof: Since $\mathbf{T_C}$ is a tree-like refutation, we can split it into two disjoint proofs $\mathbf{T_C}^{x_i}$, which derives $\{x_i\}$ and $\mathbf{T_C}^{\bar{x}_i}$, which derives $\{\bar{x}_i\}$. Note that the term disjoint does not mean that the two proofs do not have clauses in common; rather, multiple occurrences of the same clause are counted multiple times. Let $\mathbf{T_1}^{x_i}$ denote a shorter tree-like proof for $\{x_i\}$; we could combine this proof with $\mathbf{T_C}^{\bar{x}_i}$, to get a shorter tree-like proof for $\Box$, thereby contradicting the optimality of $\mathbf{T_C}$. The same argument holds if there exists a shorter tree-like proof for $\{\bar{x}_i\}$.

The above discussion leads to the following strategy to find the shortest tree-like refutation of C:

(1) For each literal pair $(x_i, \bar{x}_i)$ find the shortest path from x_i to $\bar{x}_i$ of length s_i and the shortest path from $\bar{x}_i$ to x_i of length s'_i, using BFS in time $O(m+n)$.
(2) If both s_i and s'_i are finite, then the shortest tree-like refutation of C, by resolving on $\{x_i\}$ and $\{\bar{x}_i\}$ has length $l_i = (s_i + (s_i - 1) + s'_i + (s'_i - 1))$.
(3) The shortest tree-like refutation of C has length $l = \min_{i=1}^{n} l_i$, accordingly we choose i so as to minimize $(s_i + s'_i)$.

It is not hard to see that this strategy can be implemented in time $O(m \cdot n + n^2)$.

We make the following observations:

(1) It is possible that the shortest path from x_i to $\bar{x_i}$ has one or more edges (clauses) in common with the shortest path from $\bar{x_i}$ to x_i; in case of tree-like refutations, an edge which is used twice must be counted twice, unlike the case for dag-like refutations.

(2) The shortest resolution refutation of a 2SAT formula need not be tree-like; the above strategy fails for dag-like refutations. In particular, the optimal dag-like refutation of C, need not be composed of *disjoint* proofs of $\{x_i\}$ and $\{\bar{x_i}\}$. Indeed the problem of finding the shortest dag-like refutation for an unsatisfiable 2SAT formula is open at this point.

ACKNOWLEDGMENTS

We are indebted to Samuel R. Buss for helpful discussions.

REFERENCES

ACHLIOPTAS, BEAME, AND MOLLOY. 2001. A sharp threshold in proof complexity. In *STOC: ACM Symposium on Theory of Computing (STOC)*.

ALEKHNOVICH, M., BUSS, S., MORAN, S., AND PITASSI, T. 1998. Minimum propositional proof length is np-hard to linearly approximate. In *Mathematical Foundations of Computer Science (MFCS)*. Springer-Verlag. Lecture Notes in Computer Science.

ASPVALL, B., PLASS, M. F., AND TARJAN, R. 1979. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters* 3.

BEAME, P. AND PITASSI, T. 1996. Simplified and improved resolution lower bounds. In *37th Annual Symposium on Foundations of Computer Science*. IEEE, Burlington, Vermont, 274–282.

BEAME, P. AND PITASSI, T. 1998. Propositional proof complexity: Past, present, future. *Bulletin of the EATCS 65*, 66–89.

BEN-SASSON AND WIGDERSON. 2001. Short proofs are narrow–resolution made simple. *JACM: Journal of the ACM 48*.

BESTAVROS, A. AND FAY-WOLFE, V., Eds. 1997. *Real-Time Database and Information Systems, Research Advances*. Kluwer Academic Publishers.

BOCKMAYR, A. AND EISNBRAND, F. Combining logic and optimization in cutting plane theory. In *Proceedings of Frontiers of Combining Systems (FROCOS) 2000*. Springer-Verlag, 1–17. Lecure Notes in Artificial Intelligence.

CLEGG, M., EDMONDS, J., AND IMPAGLIAZZO, R. 1996. Using the Groebner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*. Philadelphia, Pennsylvania, 174–183.

COOK, S. A. AND RECKHOW, R. A. 1973. Time bounded random access machines. *Journal of Computer and System Sciences 7*, 4 (Aug.), 354–375.

HAKEN, A. 1985. The intractability of resolution. *Theoretical Computer Science 39*, 2-3 (Aug.), 297–308.

IWAMA, K. 1997. Complexity of finding short resolution proofs. *Lecture Notes in Computer Science 1295*, 309–319.

IWAMA, K. AND MIYANO, E. 1995. Intractability of read-once resolution. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory (SCTC '95)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 29–36.

PUDLÁK, P. 1997. Lower bounds for resolution and cutting plane proofs and monotone computations. *The Journal of Symbolic Logic 62*, 3 (Sept.), 981–998.

RAZBOROV, A. A. 1996. Lower bounds for propositional proofs and independence results in bounded arithmetic. In *Automata, Languages and Programming, 23rd International Colloquium*, F. M. auf der Heide and B. Monien, Eds. Lecture Notes in Computer Science, vol. 1099. Springer-Verlag, Paderborn, Germany, 48–62.

URQUHART, A. 1995. The complexity of propositional proofs. *The Bulletin of Symbolic Logic 1*, 4 (Dec.), 425–467.

Received April 2002; revised July 2002; accepted November 2002