Empirical validation of polyhedral projection schemes for selected problems in Network Optimization, Real-Time Scheduling and Logic

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

David Owen
LDCSEE,
West Virginia University,
Morgantown, WV
{dowen@csee.wvu.edu}

Abstract

In this paper, we study the implementation of polyhedral projection schemes for problems in the domains of Network Optimization, Real-Time Scheduling and Logic. We use the Fourier-Motzkin elimination procedure or its dual to effect polyhedral projection. In the case of the Network Optimization and Real-Time scheduling problems, polyhedral projection schemes performed an order of magnitude better than existing schemes on randomly generated inputs, thereby conclusively demonstraing their superiority. To the best of our knowledge, our implementational study of Quantifed 2SAT (Logic) is the first of its kind.

1 Introduction

This paper is concerned with the experimental validation of polyhedral projection (or its dual) as a general-purpose technique for attacking problems in diverse domains, such as Network Optimization, Real-Time Scheduling and Logic. We use the Fourier-Motzkin elimination technique (see §B) (or its dual) as the backbone of our polyhedral projection algorithms. Although, other polyhedral projection schemes exist in the literature [Wei97, LW93], the Fourier-Motzkin procedure is our method of choice, on account of its simplicity and wide applicability.

In Network Optimization, we are concerned with the problem of checking whether there exists a negative cost cycle in an arbitrary, directed graph with positive and negative costs on the edges. This problem has numerous applications in VLSI design and scheduling (§2.2). In real-time scheduling, we are interested in ascertaining the parametric feasibility of a constraint system on jobs with variable execution times. Checking for parametric feasibility is in some sense, a natural generalization of the negative cost cycle problem. In this case, polyhedral projection (although not necessarily the Fourier-Motzkin procedure), is necessitated by the description of the problem (See §3. In Logic, we work on the problem of checking whether an arbitrarily specified Quantified 2SAT formula is feasible. Our approach to this problem too is based on polyhedral projection and is completely different from the approach in [APT79].

Our experiments indicate that polyhedral projection is an effective alternative to "standard" algorithms in the literature; our work also has some interesting theoretical consequences, which are delineated in the appropriate sections.

The rest of this paper is organized as follows: Section §2 discusses the problem of identifying a negative-cost cycle in a directed, weighted graph. The contraction algorithm developed in this section is used as the basis for designing an algorithm for Parametric Scheduling in Real-Time Systems; implementations of the primal and dual versions of a quantifier elimination algorithm are discussed in Section §3. Section §4 is concerned with the implementation of algorithms for the Quantified 2SAT problem. We conclude in Section §5, by summarizing our contributions and discussing problems of theoretical interest.

2 The Negative Cost Cycle Problem

This section is concerned with the problem of identifying the existence of negative cost cycles in directed graphs.

2.1 Statement of Problem

Given a directed graph $\mathbf{G} =< \mathbf{V}, \mathbf{E} >$, *where* $|\mathbf{V}| = n$ *and* $|\mathbf{E}| = m$, *and a cost function* $c : \mathbf{E} \rightarrow \Re$, *is there a negative cost cycle in* $\mathbf{G}$*?*

2.2 Motivation and Related Work

Constraint graphs are often used to represent systems of difference constraints; an application of Farkas' Lemma shows that a system of difference constraints is feasible if and only if there does not exist a negative cost cycle in the corresponding constraint graph [CLR92, DMP91]. In the design of VLSI circuits, it is required to isolate negative feedback loops. These negative feedback loops correspond to negative cost cycles in the amplifier-gain graph corresponding to the circuit [WE94]. The problem of checking whether a zero-clairvoyant scheduling system has a valid schedule can also be reduced to the problem of identifying negative cost cycles in the appropriate graph [Sub00].

One of the earliest and to date the fastest (asymptotically) algorithm for the negative cost cycle problem is the Bellman-Ford algorithm [CLR92]. When all the arc-weights are integral, the scaling algorithm in [Gol95] has been shown to be better empirically. The principal problem with both these algorithms is that they are both functional in nature, i.e. they are designed to calculate the actual shortest paths in the graph (*optimization*). As a result, they always execute a pre-specified number of steps: $O(mn)$ in the case of Bellman-Ford and $O(\sqrt{n}.m \log N)$ in the case of Goldberg's algorithm (N is absolute value of the most negative edge in the graph). Our problem is much simpler in the following sense; we are content with merely identifying the existence of a negative cost cycle (*feasibility*). It is therefore at least reasonable to expect that our problem has a faster strategy (even, if only from an empirical perspective) than the problem of finding the shortest paths in a directed graph with positive and negative edge weights.

2.3 The Vertex-Contraction Algorithm

The *vertex contraction* procedure consists of eliminating a vertex from the input graph, by merging all its incoming and outgoing edges. Consider a vertex v_i with incoming edge e_{bi} and outgoing edge e_{ia}. When v_i, is contracted, e_{bi} and e_{ia} are deleted and a single edge e_{ba} is added with cost $c_{bi} + c_{ia}$. This process is repeated for each pair of incoming and outgoing edges. Consider the edge e_{ba} that is created by the contraction; it falls into one of the following categories:

1. It is the first edge between vertex v_a and v_b. In this case, nothing more is to be done.
2. An edge e'_{ba} already existed between v_a and v_b, prior to the contraction of v_i. In this case, if $c_{ba} \leq c'_{ba}$, keep the new edge and delete the previously existing edge (since it is redundant); otherwise delete the new edge (since it is redundant).

Algorithm (2.1) is a formal description of our technique.

The function Prune-Graph() checks for the presence of multiple edges and loops. Multiple edges are replaced by a single non-redundant edge, while loops are discarded (if they have non-negative cost). If a negative cost loop is detected in Prune-Graph(), the algorithm terminates.

2.4 Correctness and Analysis

The correctness of Algorithm (2.1) follows from the trivial observation that vertex contraction is a path-cost preserving operation, i.e. if there is a path of cost c_1 from vertex v_a to vertex v_b, before the contraction of a vertex (other than v_a or v_b), then there exists a path of cost at most c_1 from v_a to v_b, after the contraction.

```
Function NEGATIVE-COST-CYCLE($\mathbf{G}, n$)
1: if ($n = 1$) then
2:    if ($c_1 < 0$) then
3:       return(true)
4:    else
5:       return(false)
6:    end if
7: else
8:    $\mathbf{G}'$ =VERTEX-CONTRACT($\mathbf{G}, n$)
9:    if (PRUNE-GRAPH)($\mathbf{G}', n-1$)) then
10:      return(true)
11:   end if
12:   NEGATIVE-COST-CYCLE ($\mathbf{G}', n-1$)
13: end if
```

Algorithm 2.1: Negative cost cycle detection

```
Function VERTEX-CONTRACTION($\mathbf{G}, n$)
1: for ($i = 1$ to $n$) do
2:    for ($j = 1$ to $n$) do
3:       if ($e_{in}$ and $e_{nj}$ exist) then
4:          Delete edges $e_{ij}$ and $e_{jn}$
5:          Add edge $e_{ij}$ with cost $c_{ij} = c_{in} + c_{nj}$
6:       end if
7:    end for
8: end for
```

Algorithm 2.2: Vertex Contraction

Every cycle (including negative cost cycles) is a path from some vertex to itself and hence negative cost cycles will eventually be detected as negative cost loops about some vertex.

Each vertex-contraction and pruning operation takes at most $O(n^2)$ time; since there are n vertices in all, NEGATIVE-COST-CYCLE() takes $O(n^3)$ time in the worst case. Thus, for dense graphs, Algorithm (2.1) is competitive with Bellman-Ford; however for sparse graphs, the situation is not so sanguine. For instance, an adversary could provide the graph in Figure (1) as input.

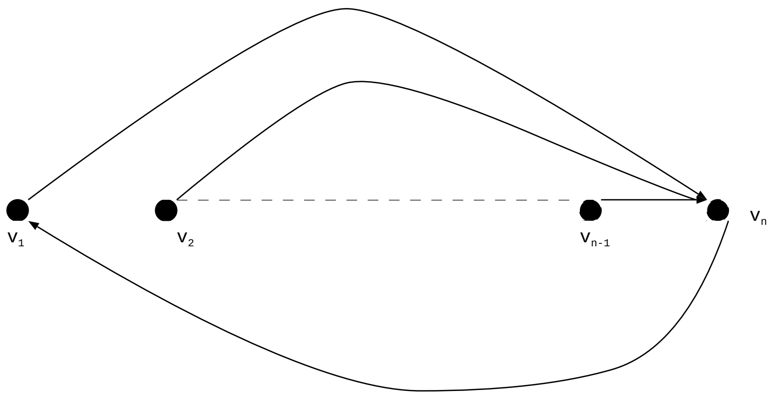

Figure 1: Sparse graph that becomes dense after vertex contraction

The graph is sparse and has exactly $2.(n-1)$ edges. However, if vertex v_n is contracted first, the resultant graph is the complete graph on $n-1$ vertices and therefore dense. It follows that $O(n^3)$ is a lower bound on the time required by the vertex-contraction algorithm. Note that instead of picking vertices in the defined order, we could choose the vertex to be contracted at random, without affecting the correctness of the algorithm. We have implemented Algorithm (2.1) in two different ways; in one implementation, the vertex to be contracted is chosen in a well-defined order, whereas in the second implementation, it is chosen at random.

Remark: 2.1 *The vertex contraction algorithm is precisely the Fourier-Motzkin elimination technique applied to the dual graph! (see §B)*

2.5 Implementation

```
Function GENERATE-GRAPH($\mathbf{G}, n, k, p_1, p_2$)
 1: for ($i = 1$ to $n$) do
 2:   for ($j = 1$ to $n$) do
 3:     if ($i > j$) then
 4:       With probability $p_1$ create edge $e_{ij}$.
 5:       Assign random weight from $[1 \cdot \cdot k]$.
 6:     else
 7:       if ($i < j$) then
 8:         With probability $p_2$ create edge $e_{ij}$.
 9:         Assign random weight from $[-k \cdot \cdot -1]$.
10:       end if
11:     end if
12:   end for
13: end for
```

Algorithm 2.3: Procedure for generating a graph **G**, with n vertices, maximum edge weight absolute value k, positive weight edge probability p_1, and negative weight edge probability p_2, for the Negative Cost Cycle Problem.

	Sparse			Medium			Dense		
n	BF	VC	RVC	BF	VC	RVC	BF	VC	RVC
25	1	0	0	1	0	0	0	0	0
50	2	0	0	3	0	0	4	0	0
75	7	0	0	10	0	0	12	0	0
100	15	0	0	23	0	0	29	0	0
125	30	0	0	44	0	0	55	1	0
150	51	1	1	77	1	0	99	0	1
175	83	0	1	121	1	0	158	1	1
200	124	1	0	183	1	1	236	1	1
225	174	1	1	258	1	1	332	1	1
250	236	2	1	352	2	1	458	2	1
275	314	2	2	468	2	2	610	2	2
300	404	2	3	612	2	2	800	2	2
325	522	2	3	775	2	2	1,022	2	3
350	644	4	3	982	3	3	1,254	3	3
375	803	3	4	1,202	3	3	1,578	3	3
400	973	3	3	1,461	3	4	1,910	4	4
425	1,141	4	5	1,767	4	5	2,291	4	4
450	1,400	5	5	2,080	5	5	2,716	6	4
475	1,636	5	5	2,432	6	5	3,182	6	5
500	1,924	6	6	2,867	6	6	3,677	7	5
Total:	10,484	41	43	15,718	42	40	20,423	45	40

Table 1: Comparison of Bellman-Ford (BF), Vertex Contraction (VC) and Randomized Vertex Contraction(RVC) execution times required to solve the Negative Cost Cycle problem on graphs of varying size and edge density.

Graphs for the Negative Cost Cycle problem were generated according to the procedure shown in Algorithm (2.3). The graphs generated are vertex ordered in that edge e_{ij} will have cost ≤ 0, if $i < j$ and cost ≥ 0, if $i > j$ [1]. Table (1) shows a comparison of Bellman-Ford, Vertex Contraction, and Randomized Vertex Contraction execution times required to solve the Negative Cost Cycle problem for graphs of varying size and edge density. Execution times for all implementation tables are given in (truncated) hundredths of a second; a running time of zero, merely indicates that the time taken was less than $\frac{1}{100}$ of a second. "Sparse" graphs are those in which the probability that a particular edge exists $p_1 = p_2 = 0.1$; in "Medium" graphs $p_1 = p_2 = 0.5$, and in "Dense" graphs $p_1 = p_2 = 0.9$. Edge weights are generated from a uniform distribution over the appropriate interval.

Table (1) compares the performance of the 3 algorithms: Bellman-Ford (BF), Vertex Constration (VC) and Randomized Vertex-Contraction (RVC) on vertex ordered graphs of varying densities.

Table (2) compares Bellman-Ford, Vertex Contraction, and Randomized Vertex Contraction execution times for graphs in which the probability that a backward (positive weight) edge exists differs from the probability a forward (negative weight) edge exists. For graphs with "many Negative Edge Weights," the probability that a backward edge exists $p_1 = 0.1$, and the probability a forward edge exists $p_2 = 0.9$. For graphs with "Few Negative Edge Weights," the probabilities are reversed, with $p_1 = 0.9$ for backward edges and $p_2 = 0.1$ for forward edges. Essentially, we are taking into account graphs which do not have a high probability of containing negative cost cycles.

2.6 Summary

From Table(1) and Table (2), it is easy to see that the Vertex Contraction technique (with or without randomization) is vastly superior to the Bellman-Ford algorithm in all instances. The superiority is demonstrated not only in the total suite time, but also in each individual instance. We note that the effect of randomization has been minimal at best; the reason is that we are generating random graphs to begin with; consequently the probability

[1] It is not hard to see that the class of vertex ordered graphs encompasses the set of all directed, weighted graphs.

	Many Negative Edge Weights			Few Negative Edge Weights		
n	BF	VC	RVC	BF	VC	RVC
25	0	0	0	0	0	0
50	2	0	0	3	0	0
75	9	0	0	9	0	0
100	22	0	0	21	0	1
125	43	1	0	41	0	1
150	75	0	1	72	0	1
175	120	0	1	116	1	1
200	180	1	1	173	1	0
225	252	1	2	247	1	1
250	348	2	2	344	2	1
275	470	2	2	451	2	1
300	606	2	2	592	2	2
325	779	3	2	760	3	2
350	970	3	3	942	3	2
375	1,191	3	4	1,164	3	4
400	1,450	3	4	1,411	4	4
425	1,727	5	4	1,694	4	5
450	2,043	5	5	2,005	5	5
475	2,420	4	6	2,363	5	5
500	2,824	6	7	2,764	6	6
Totals:	15,531	41	47	15,172	42	42

Table 2: Comparison of Bellman-Ford (BF), Vertex Contraction (VC) and Randomized Vertex Contraction(RVC) execution times required to solve the Negative Cost Cycle problem on graphs of varying size and number of negative edge weights.

that in any given instance, we will make a sequence of bad choices for vertices to be contracted is small. One interesting direction in which our research could be extended is to provide an analysis of the expected convergence time, in case of the Randomized Vertex Contraction algorithm.

3 Real-Time Scheduling

An important feature in Real-Time systems is *parameter impreciseness*, i.e. the inability to accurately determine certain parameter values. The most common such parameter is *task execution time.* A second feature is the presence of relative timing constraints between jobs such as: *Start Job J_2 5 units after J_1 finishes.* Here, we focus on *parametric real-time systems* wherein the only information available before a job is to be dispatched is the *actual execution time of every job sequenced before it.* The primary scheduling goal is to provide an offline guarantee that the input constraints will be met at run-time. In a Parametric System, the dispatch time of a job will, in general, be a parameterized function of the start and execution times of jobs sequenced before it.

3.1 Statement of Problem

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

3.2 Constraint Model

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

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

where,

- $\mathbf{A}$ is an $m \times 2.n$ rational matrix,
- $\mathbf{E}$ is an axis-parallel rectangle aph represented by:

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

  The aph Υ 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.

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

Since, the jobs are non-preemptive, the finish time f_i of job J_i can be expressed as $f_i = s_i + e_i$; hence there is no gain in introducing separate variables to represent finish times. In this paper, we only care about "standard" constraints, i.e. strict difference constraints between start or finish times of jobs. Accordingly, the following types of constraints are permitted between J_i and J_j, $(i < j)$:

1. Relative timing between s_i and s_j, e.g. $s_i + 4 \le s_j$
2. Relative timing between s_i and $(s_j + e_j)$, e.g. $s_i + 4 \ge s_j + e_j + 8$
3. Relative timing between $(s_i + e_i)$ and s_j, e.g. $(s_i + e_i) + 4 \le s_j$
4. Relative timing between $(s_i + e_i)$ and $(s_j + e_j)$, e.g. $s_i + e_i + 8 \ge s_j + e_j$

For a detailed description of standard constraints, see [GPS95].

3.3 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: 3.1 *A parametric schedule of an ordered set of jobs, in a scheduling window, is a vector* $\vec{\mathbf{s}} = [s_1, s_2, \ldots, s_n]$, *where* s_1 *is a rational number and each* $s_i, i \neq 1$, *is a function of the start time and execution time variables of jobs sequenced prior to job* J_i, *i.e.* $\{s_1, e_1, s_2, e_2, \ldots, s_{i-1}, e_{i-1}\}$. *Further, this vector should satisfy the constraint system (1) for all execution time vectors* $\vec{\mathbf{e}} \in \Upsilon$.

Accordingly, the parametric scheduling problem is concerned with deciding the following query:

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

Remark: 3.1 *Observe that the query is asking whether a totally unimodular Quantified Linear Program [Sub01c] is feasible.*

3.4 Motivation and Related Work

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

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

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

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

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

This assignment satisfies the input set of constraints and is hence a valid dispatch schedule. The key feature of the schedule provided by Equation (4) is that the start time of job J_2 is no longer an absolute time, but a (parameterized) function of the start and execution times of job J_1.

Parametric Scheduling was introduced in [Sak94] as a technique to address the inflexibility of Static Scheduling. A number of special cases have been analyzed since in [GPS95, Cho00, SA00, Sub01b]. To the best of our knowledge, our effort in this paper marks the first implementational study of algorithms for parametric scheduling.

3.5 Primal and Dual Algorithms

Algorithm (3.1) [GPS95, Cho97] represents a primal strategy to solve the problem, while Algorithm (3.3) [Sub00, Sub01a] represents a dual strategy for the same.

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

Algorithm 3.1: The Primal Algorithm

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

Algorithm 3.2: Eliminating Universally Quantified variable $e_i \in [l_i, u_i]$

A detailed analysis of Algorithm (3.1), including the proof of correctness, is available in [GPS95]; essentially the algorithm works by unrolling the quantifier string, one quantifier at a time, until only the first variable remains. Each elimination step is solution-preserving; i.e. there is a solution to the original system, if and only if there is a solution to the system that results after the elimination of quantified (existential or universal) variable. It can be shown that the above algorithm converges in time $O(n^3)$. Eliminating universally quantified variables is done through Algorithm (3.2), while eliminating existentially quantified variables is accomplished through the Fourier-Motzkin elimination.

Algorithm (3.1) requires an array-based implementation for the Fourier-Motzkin elimination procedure; this implementation significantly slows down the task of checking the parametric feasibility of a Job system. Algorithm (3.3) represents a dual strategy for the same problem. It is based on the following observations:

Let $\mathbf{G} =< \mathbf{V}, \mathbf{Q} >$ represent the dual graph of a (standard) constraint system $\mathbf{A}.[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}$. $\mathbf{V}$ is the set of vertices and $\mathbf{Q}$ is the set of edges. Consider any two vertices s_i and s_j, $i \leq j$.

Definition: 3.2 *A valid path between s_i and s_j is a sequence $s_i = s'_0, q_1, s'_1, q_2, \ldots, q_k, s_j = s'_k$, where each q_l is an edge in $\mathbf{Q}$ from s'_{l-1} to s'_l and therefore represents a difference constraint of the form $s'_{l-1} - s'_l \leq ()$*

Remark: 3.2 *This definition is different from the standard definition of a path in a directed graph [CCPS98], in that a valid path can exist only from a vertex s_i to a vertex s_j, $i \leq j$, i.e. the index number of the start vertex must be less than or equal to the index number of the final vertex.*

Remark: 3.3 *An edge in a valid path can exist from a vertex with a higher numbered vertex to a lower numbered vertex. Thus in Figure (2), q_1 is not a valid path; however it is a valid edge on the path s_2, q_1, s_1, q_3, s_2.*

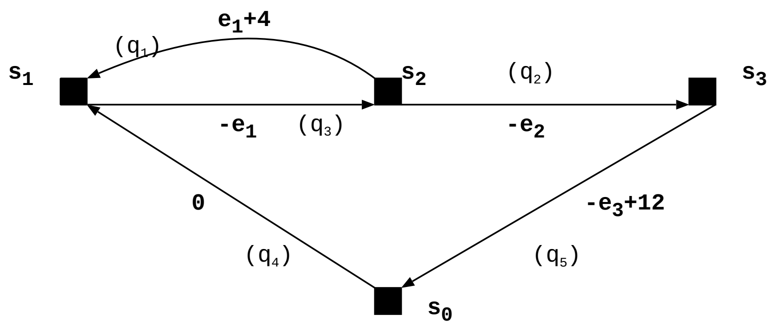

Figure 2: Constraint graph representation for a system of difference constraints

Let P denote the valid path $< s_i = s'_0, q_1, s'_1, q_2, \ldots, s_j = s'_k >$, $i \leq j$. The cost of this path $c(P)$ is calculated as follows: Substitute $e_k = l_k$ on all edges of the path, where e_k occurs with a positive sign and $e_k = u_k$ on all edges of the path, where e_k occurs with a negative sign if $k \geq i$ [2]. Now perform symbolic addition over the path P to get an affine function $\vec{\mathbf{r'}}.\vec{\mathbf{e'}} + k_1$, for some $\vec{\mathbf{r'}}, k_1$, where $\vec{\mathbf{e'}} = [e_1, e_2, \ldots, e_{i-1}]$. Then, $l = \min_{\mathbf{E}} \vec{\mathbf{r'}}.\vec{\mathbf{e'}} + k_1$ is called the *parametric cost* of the path, e.g. the parametric cost of path $< s_2, q_1, s_1, q_3, s_2, q_2, s_3 >$ is $\min_{\mathbf{E}} e_1 + 4 - e_1 - 4 = 0$.

Theorem: 3.1 *A constraint system $\mathbf{A}[\vec{\mathbf{s}}, \vec{\mathbf{e}}] \leq \vec{\mathbf{b}}$ is parametrically infeasible if and only if there is a valid loop in the corresponding graph $\mathbf{G} =< \mathbf{V}, \mathbf{Q} >$ having negative parametric cost.*

Proof: *See [Sub01a], [Sub00].* □

The above theorem is exploited by Algorithm (3.3), which essentially identifies negative parametric cost loops in the constraint graph. The VERTEX-CONTRACT() procedure use in Algorithm (3.3) is similar to the VERTEX-CONTRACT() procedure in Algorithm (2.1); the only difference between the two is that in case of Parametric Scheduling, the edge costs are added symbolically. Since at most two execution time variables need to be added during any contraction, the addition can still be implemented in $O(1)$ time.

3.6 Implementation

Parametric Scheduling constraint graphs were generated according to the procedure shown in Algorithm (3.4). Table (3) compares execution times for the primal algorithm and the dual algorithm running on Parametric

[2]Recall that $e_i \in [l_i, u_i]$

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

Algorithm 3.3: The Dual Algorithm

Scheduling constraint graphs with varying numbers of jobs and constraints. As with Negative Cost Cycle graphs, in "Sparse" constraint graphs a particular edge exists with probability $p_1 = 0.1$, in "Medium" graphs with $p_1 = 0.5$, and in "Dense" graphs with $p_1 = 0.9$. Table (4) compares the algorithms on graphs with varying probability that an edge, if it exists, is a forward (negative weight) edge. For graphs with "Many Forward Edges," $p_2 = 0.1$; for graphs with "Few Forward Edges," $p_2 = 0.9$.

3.7 Summary

To the best of our knowledge, our implementational study of the Parametric Scheduling algorithms is the first of its kind in the literature. It is clear that the performance of the dual algoritm, i.e. Algorithm (3.3) is definitely superior to that of the primal algorithm. We handcoded the Fourier-Motzkin procedure, as opposed to using freely available code; it is perhaps possible to speed up the Primal algorithm, by using *optimized* implementations of the Fourier-Motzkin procedure. An interesting open problem is to devise an $O(mn)$ algorithm for the Parametric Scheduling problem.

4 Quantified 2SAT

Quantified 2SAT refers to an instance of the Satisfiability problem (assumed to be in CNF) in which:

1. Each clause has at most 2 literals,
2. Every variable is quantified with either a $\exists$ or a $\forall$ sign.

[APT79] has shown that this problem admits a linear-time solution; they exploit the correspondence between 2SAT formulae and directed graphs to derive their criteria for checking satisfiability.

In this section, we show that a 2-SAT formula can be represented as an integer program which admits polyhedral elimination techniques; in particular we show that the Fourier-Motzkin elimination procedure can be applied to these formulae to test for satisfiability. The advantage of using elimination procedures are twofold: (a) they are inherently incremental, and (b) a number of symbolic computation programs have such procedures built-in [CH99].

```
Function GENERATE-CONSTRAINT-GRAPH($\mathbf{G}, n, x, L, w, p_1, p_2$)
for ($i = 1$ to $n$) do
  Choose $a$ and $b$ at random from $[0, x]$.
  Assign job $i$ execution time lower bound $l_i = \min(a, b)$, upper bound $u_i = \max(a, b)$.
end for
for ($i = 1$ to $n$) do
  for ($j = 0$ to $i - l$) do
    if ($i = 1$, $j = 0$) then
      Create type 1 edge from $v_0$ to $v_1$ with weight 0.
    else
      if ($i = n$, $j = 0$) then
        Create type 1 edge from $v_n$ to $v_0$ with weight $L$.
      end if
    else
      for (each of four possible edge types) do
        if (true with probability $p_1$) then
          if (true with probabiltiy $p_2$) then
            Create edge from $v_i$ to $v_j$ with constant portion of weight taken at random from $[0, w]$
            (positive weight edge).
          else
            Create edge from $v_j$ to $v_i$ with constant portion of weight taken at random from $[-w, 0]$
            (negative weight edge).
          end if
        end if
      end for
    end if
  end for
end for
for ($i = 1$ to $n - 1$) do
  if (No type 2 or type 4 edge between $v_i$ and $v_{i+1}$) then
    Create (type 2) edge from $v_i$ to $v_{i+1}$ with weight $e_i$.
  end if
end for
```

Algorithm 3.4: Procedure for generating a Parametric Scheduling constraint graph $\mathbf{G}$ with n jobs, maximum job execution time bound x, scheduling window L, maximum time between jobs w, edge existence probability p_1, and edge direction probability p_2.

	Sparse		Medium		Dense	
Jobs	Primal	Dual	Primal	Dual	Primal	Dual
2	1	0	0	0	0	0
4	0	0	0	0	0	0
6	0	0	1	0	1	0
8	0	0	1	0	1	0
10	0	0	1	0	3	0
12	2	0	2	0	6	0
14	1	0	6	0	10	0
16	4	0	7	0	22	0
18	4	0	13	0	37	0
20	3	0	20	0	67	0
22	10	1	29	0	97	0
24	22	0	50	0	147	0
26	29	0	71	0	209	0
28	23	0	95	0	304	0
30	35	0	127	0	432	0
32	13	0	188	0	595	0
34	10	0	235	0	847	0
36	187	0	360	1	973	0
38	244	1	452	0	4,891	0
40	138	0	554	0	10,755	0
Total:	726	2	2,212	1	19,397	0

Table 3: Comparison of Primal and Dual algorithm execution times required to verify the existence of a valid Parametric Schedule, for varying constraint graph density and number of jobs.

4.1 Statement of Problem

Let

$$F = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; C, \tag{5}$$

where C is a conjunction of m clauses in which each clause has at most 2 literals. The quantified satisfiability problem is concerned with answering the question: *Is F true?*

We first note that F has the following Integer Programming formulation,

$$F = Q_1 x_1 \in \{0,1\} Q_2 x_2 \in \{0,1\} \ldots Q_n x_n \in \{0,1\} \;\; \mathbf{A}.\vec{\mathbf{x}} \leq \vec{\mathbf{b}}, \tag{6}$$

where

1. $\mathbf{A}$ has n columns corresponding to the n variables and m rows corresponding to the m constraints.
2. A clause (x_i, x_j) is replaced by the integer constraint $x_i + x_j \geq 1$; a clause $(\bar{x_i}, x_j)$ is replaced by $1 - x_i + x_j \geq 1 \Rightarrow -x_i + x_j \geq 0$; a clause of the form $(x_i, \bar{x_j})$ is replaced by $x_i + 1 - x_j \geq 1 \Rightarrow x_i - x_j \geq 0$, and a clause of the form $(\bar{x_i}, \bar{x_j})$ is replaced by $1 - x_i + 1 - x_j \geq 1 \Rightarrow -x_i - x_j \geq 0$.
3. Each Q_i is one of $\exists$ or $\forall$.

The equivalence of of the clausal system and the integer programming formulation have been argued in [CLR92] and [Pap94].

4.2 Related Work

[Sch78] argued that Q2SAT could be solved in polynomial time, but provided no algorithm for the same; [APT79] gave the first algorithm for this problem. The parallel complexity of this problem has been explored in [Che92], [CM88] and [Gav93] and it has been demonstrated that Q2SAT is in the parallel complexity class $\mathtt{NC}_2$.

	Many Forward (Negative Weight) Edges		Few Forward (Negative Weight) Edges	
Jobs	Primal	Dual	Primal	Dual
2	0	0	0	0
4	1	0	0	0
6	0	0	1	0
8	1	0	1	0
10	3	0	1	0
12	2	0	5	0
14	4	0	4	0
16	34	0	7	0
18	12	0	26	0
20	23	0	22	1
22	30	0	36	0
24	139	0	53	0
26	71	0	70	0
28	98	0	338	0
30	137	0	151	0
32	192	0	197	0
34	263	0	276	1
36	350	0	356	0
38	439	0	424	0
40	589	0	600	0
Total:	2,388	0	2,568	2

Table 4: Comparison of the primal and dual algorithm execution times required to verify the existence of a valid Parametric Schedule, for varying numbers of forward edges and jobs.

Fourier-Motzkin elimination, as a technique for resolving feasibility in linear systems was proposed in [Fou24] and elaborated in [DE73]. Extending the technique to resolving integer programs was the thrust of [Wil76] and [Wil83]. A direct application of Fourier's theorem to integer programs results in congruences and modulo arithmetic. We show that for the restricted case of 2SAT integer programs, we can avoid modulo arithmetic altogether.

4.3 The Quantifier Elimination Algorithm

Algorithm (4.1) represents our strategy for deciding a Quantified 2SAT query. Although this algorithm is similar to Algorithm (3.1), it is important to note that the constraint program in §3 is a *Quantified Linear Program* (QLP), whereas in this section we are dealing with a *Quantified Integer Program* (QIP). The main contribution of this section, is the observation that 2SAT QIPs can be relaxed to QLPs and solved using existing algorithms for the same [Sub00].

Lemma: 4.1 *Given a polyhedral system of the form* $\mathbf{A}.\vec{\mathbf{x}} \le \vec{\mathbf{b}}$ *and a variable* x_i*, which is universally quantified, i.e. of the form* $\forall x_i \in \{0,1\}$*, we can replace the set* $\{0,1\}$ *with the interval* $[0,1]$ *while maintaining the truth value of the query.*

Proof: *Let us focus on any one variable, say* x_i*, which is universally quantified as:* $\forall x_i \in \{0,1\}$*. Let the query be:*

$$Q_1 x_1 Q_2 x_2 \ldots \forall x_i \in \{0,1\} \ldots Q_n x_n \mathbf{A}.\vec{\mathbf{x}} \le \vec{\mathbf{b}} \tag{7}$$

Now consider the query

$$Q_1 x_1 Q_2 x_2 \ldots \forall x_i \in [0,1] \ldots Q_n x_n \mathbf{A}.\vec{\mathbf{x}} \le \vec{\mathbf{b}} \tag{8}$$

It is clear that System (8) $\Rightarrow$ System (7). We need to show that System (7) $\Rightarrow$ System (8). Let us say that the polyhedral system is infeasible for $x_i = a, 0 < a < 1$. We then increase a to 1. If the system stays infeasible then we are done. Let us consider the case, in which the system becomes feasible. Then decreasing a to 0 must cause the constraint violation to increase and hence the System to become infeasible. This is a consequence of the local convexity of the polyhedral system, i.e. if the system is feasible for $x_i = a$ and for $x_i = b$, then it must be feasible for all $x_i \in (\lambda.a + (1-\lambda).b), \lambda \in [0,1]$. We can argue similarly for every other universally quantified variable. □
The import of Lemma (4.1) is that it allows us to replace every universal quantification of the form $\{0,1\}$ with the interval $[0,1]$, thereby permitting us to use interval elimination techniques.

```
Function DECIDE-Q2SAT ($\mathbf{A}, \vec{\mathbf{b}}$)
 1: for ($i = n$ down to 1) do
 2:   if ($Q_i = \exists$) then
 3:     Eliminate $x_i$ using the Fourier-Motzkin procedure
 4:     PRUNE-CONSTRAINTS()
 5:     if (CHECK-INCONSISTENCY()) then
 6:       return (false)
 7:     end if
 8:   end if
 9:   if ($Q_i = \forall$) then
10:     ELIM-UNIV-VARIABLE($x_i$)
11:     PRUNE-CONSTRAINTS()
12:     if (CHECK-INCONSISTENCY()) then
13:       return (false)
14:     end if
15:   end if
16:   if (CHECK-INCONSISTENCY()) then
17:     return (false)
18:   end if
19: end for
20: return (true)
```

Algorithm 4.1: A Quantifier Elimination Algorithm for deciding Quantified 2SAT

```
Function ELIM-UNIV-VARIABLE ($\mathbf{A}, \vec{\mathbf{b}}, x_i$)
 1: Substitute $x_i = 0$ in each constraint that can be written in the form $x_i \geq ()$
 2: Substitute $x_i = 1$ in each constraint that can be written in the form $x_i \leq ()$
```

Algorithm 4.2: Eliminating Universally Quantified variable

The elimination of a variable (existential or universal) results in one or more of the following consequences:

1. Some redundant constraints result, which can be pruned out. This is the function of the subroutine PRUNE-CONSTRAINTS(). For instance, suppose that the variable x_2, which is universally quantified, is eliminated from the constraint set $\{(x_1)(x_2, x_1)\}$, we get $\{(x_1), (x_1)\}$ which can be pruned to $\{(x_1)\}$. Likewise, suppose that the existentially quantified variable x_5 is eliminated from the constraint set $\{(x_5, x_2), (\bar{x_5}, x_3)(x_2, x_3)\}$, we get $\{(x_2, x_3)(x_2, x_3)\}$, which can be replaced by $\{(x_2, x_3)\}$. It is also possible that a clause of the form $(x_1, \bar{x_1})$ results in which case we can set the formula to be **true** and return;

2. An inconsistency results and we return **false** through the function CHECK-INCONSISTENCY(). The only inconsistencies that we care about are those that cause a variable to be set simultaneously to **true** and **false**. For instance, suppose that variable x_i is being eliminated from the constraint set $\{(x_1)(x_i, \bar{x_1})(\bar{x_i})\}$, we get $\{(x_1)(\bar{x_1})\}$, which is an inconsistency.

3. We get a smaller subset of variables to work with. Observe that the total number of constraints is always bounded by 4 times the square of the number of variables.

4.4 Correctness

The correctness of the Algorithm (4.1) follows from the correctness of the Universally quantified variable elimination procedure and the correctness of the Fourier-Motzkin elimination procedure. The correctness of the universal variable elimination procedure for polyhedra has been argued in [Sch87] and [Sub00]. The substitution operation performs an intersection of the polyhedron with the hyper-planes $x_i = 0$ and $x_i = 1$ (assuming that we are at step $\ldots \forall x_i \in [0,1] \mathbf{A'}.\vec{\mathbf{x'}} \leq \vec{\mathbf{b'}}$). It follows that the original proposition is true if and only if the intersection is non-empty. We now show that the Fourier-Motzkin elimination procedure works correctly over 2SAT clauses. Observe that the existential variable x_i needs to be eliminated only if it appears both in complemented and uncomplemented forms over the clause set. Otherwise its value is obvious and fixed.

Let us say that we are eliminating x_i from the clauses (x_i, x_j) and $(\bar{x_i}, x_k)$. Using resolution, we know that the conjunction of these two clauses can be replaced by (x_j, x_k). The constraint equations are $x_i + x_j \geq 1$ and $-x_i + x_k \geq 0$. We first rewrite the two equations as: $x_1 \geq 1 - x_j$ and $x_i \leq x_k$. Applying the elimination procedure gives $1 - x_j \leq x_k \Rightarrow x_j + x_k \geq 1 \Rightarrow (x_j, x_k)$, which is exactly what is needed to be proved. During the elimination, it is possible to obtain constraints such as $2.x_1 \leq 1$ or $2.x_1 \geq 1$; in the former case x_1 is set to 1, while in the latter case, x_1 is set to 0, since we only care about quantified lattice points.

Since x_j and x_k are chosen arbitrarily, the claim follows.

4.5 Implementation

```
Function GENERATE-QUERY($\mathbf{C}, n, p_1, p_2$)
 1: for ($i = 1$ to $n$) do
 2:   if (TRUE with probability $p_1$) then
 3:     Assign universal quantifier ($\forall$) to variable $i$.
 4:   else
 5:     Assign existential quantifier ($\exists$) to variable $i$.
 6:   end if
 7: end for
 8: for ($i = 1$ to $n - 1$) do
 9:   for ($j = i + 1$ to $n$) do
10:     With probability $p_2$ create clause $(x_i, x_j)$.
11:     With probability $p_2$ create clause $(\bar{x_i}, x_j)$.
12:     With probability $p_2$ create clause $(x_i, \bar{x_j})$.
13:     With probability $p_2$ create clause $(\bar{x_i}, \bar{x_j})$.
14:   end for
15: end for
```

Algorithm 4.3: Procedure for generating a quantified 2-SAT query, which is a set of clauses $\mathbf{C}$ with n variables; p_1 is the probability a particular variable is universally quantified, and p_2 is the probability a particular clause exists.

Queries for Quantified 2-SAT were generated according to the procedure shown in Algorithm (4.3). Tables (5), (6), and (7) compare execution times for Fourier-Motzkin elimination and Tarjan's algorithm running on quantified 2-SAT queries, varying the number of variables and the likelihood that a particular clause is included in the query. Consider the graph on the $2 \cdot n$ literals, $\{x_1, \bar{x_1}, \ldots, x_n, \bar{x_n}\}$. For "Sparse" constraint sets, the probability a particular clause is included in the query $p_2 = 0.1$; for "Medium" constraint sets $p_2 = 0.5$, and for "Dense" constraint sets $p_2 = 0.9$. The quantifier for each variable is either $\forall$ or $\exists$ with probability $\frac{1}{2}$. In Table 5, the probability a universal quantifier is chosen for a particular variable $p_1 = 0.1$; for Table (6), $p_1 = 0.5$, and for Table (7), $p_1 = 0.9$, i.e. we vary the number of alternations in the quantifier string.

	Sparse		Medium		Dense	
Vars.	T	FM	T	FM	T	FM
2	0	0	0	0	0	0
4	0	0	0	0	0	0
6	0	0	0	0	0	0
8	0	0	0	0	0	0
10	0	0	0	0	0	0
12	0	0	0	0	0	2
14	0	1	0	2	0	5
16	0	0	0	5	0	8
18	0	1	0	3	0	15
20	0	1	0	10	0	26
22	0	1	0	13	0	42
24	0	3	0	21	0	65
26	0	2	0	32	1	94
28	1	2	0	43	0	138
30	0	3	0	65	1	191
32	1	5	0	98	0	276
34	0	9	1	113	0	363
36	0	22	0	148	0	500
38	1	7	1	203	1	628
40	1	42	0	259	1	810
Total:	4	99	2	1,015	4	3,163

Table 5: Comparison of Tarjan's algorithm (T) and Fourier-Motzkin Elimination (FM) execution times required to solve Quantified 2-SAT problem for implication graphs of varying size and edge density, given the probability p_1 that a universal quantifier is chosen for a particular variable = 0.1.

4.6 Summary

From Table (6) and Table (7), it is clear that Tarjan's algorithm is superior to Algorithm (4.1) in all instances. However, in many constraint databses [BFW97], the Fourier-Motzkin procedure is already available and our work establishes that it can be used, at least in principle, for the purpose of resolving Q2SAT formulae.

Some of the interesting open problems are:

1. In [CR92] it was shown that Existential 2SAT formulae, are probably satisfiable if $m = c \cdot n, c < 1$ and probably unsatisfiable, if $c > 1$. In [Ala], we are working on similar bounds for the Q2SAT problem;

2. Can our rudimentary $O(n^3)$ analysis be improved?

3. Are there classes of 2SAT formulae for which the performance of Algorithm (4.1 compares favourably with Tarjan's algorithm?

5 Conclusion

In this paper, we analyzed polyhedral projection from an implementational perspective, by studying its performance on problems from 3 different domains. In the case of Network Optimization, polyhedral projection is clearly superior to the Bellman-Ford algorithm. In Real-Time parametric scheduling, some form of polyhedral projection is unavoidable, on account of the alternations in the quantifier string representing the schedulability query. Our dual algorithm is empirically superior to the primal algorithm, although both require $O(n^3)$ convergence time in the worst-case. In the case of the Quantified 2SAT problem, our result has some interesting theoretical offshoots. We know that IP(2), i.e. Integer Programming with at most 2 variables per constraint is `NP-complete` [HN94];

	Sparse		Medium		Dense	
Vars.	T	FM	T	FM	T	FM
2	0	1	0	0	0	0
4	0	0	0	0	0	0
6	0	0	0	0	0	0
8	0	0	0	0	0	0
10	0	0	0	0	0	1
12	0	0	0	1	0	2
14	0	0	0	1	0	4
16	0	0	0	5	0	9
18	0	1	0	6	0	15
20	0	1	0	8	0	25
22	0	1	0	13	1	41
24	0	2	0	21	0	66
26	0	2	1	30	0	94
28	0	6	0	49	0	139
30	0	7	1	64	0	189
32	0	12	1	81	0	267
34	0	15	0	119	0	356
36	0	8	0	285	0	466
38	0	9	1	189	0	622
40	0	34	0	249	0	828
Total:	0	99	4	1,121	1	3,124

Table 6: Comparison of Tarjan's algorithm (T) and Fourier-Motzkin Elimination (FM) execution times required to solve Quantified 2-SAT problem, given the probability p_1 that a universal quantifier is chosen for a particular variable $= 0.5$.

Q2SAT is a special subclass of IP(2). The question then is: *Are there other interesting subclasses of IP(2), that admit polynomial time solutions?*

Remark: 5.1 *The code for our implementations can be downloaded from :*
`http://www.csee.wvu.edu/~dowen/alenex-02`

A Implementation System

Table (8) describes the computer system used for all experiments presented in this paper.

B Fourier-Motzkin Elimination

The Fourier-Motzkin elimination procedure is an elegant, syntactic, variable elimination scheme to solve constraint systems that are comprised of linear inequalities. It was discovered initially by Fourier [Fou24] and later by Motzkin [DE73], who used it to solve general purpose Linear programs.

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

Algorithm (B.1) is a formal description of the above procedure.

Though elegant, this syntactic procedure suffers from an exponential growth in the constraint set, as it progresses. This growth has been observed both in theory [Sch87] and in practice [HLL90, LM91]. By appropriately

	Sparse		Medium		Dense	
Vars.	T	FM	T	FM	T	FM
2	0	0	0	0	0	0
4	0	0	0	0	0	0
6	0	0	0	0	0	0
8	0	0	0	0	0	1
10	0	0	0	0	0	1
12	0	0	0	1	0	2
14	0	0	0	1	0	5
16	0	0	0	3	0	9
18	0	1	0	5	0	16
20	0	1	0	8	0	25
22	0	1	0	13	0	42
24	0	1	0	18	0	62
26	0	4	1	28	0	96
28	0	8	0	42	1	133
30	0	5	0	60	0	188
32	0	11	0	88	0	266
34	0	17	0	114	0	360
36	0	21	0	151	0	487
38	0	6	0	211	0	623
40	1	10	1	264	0	793
Total:	1	86	2	1,007	1	3,109

Table 7: Comparison of Tarjan's algorithm (T) and Fourier-Motzkin Elimination (FM) execution times required to solve Quantified 2-SAT problem, given the probability p_1 that a universal quantifier is chosen for a particular variable $= 0.5$.

choosing the constraint matrix **A**, it can be shown that eliminating k variables causes the size of the constraint set to increase from m to $O(m^{2^k})$ [Sch87]. Algorithm (B.1) remains useful though as a tool for proving theorems on polyhedral spaces [VR99]. [Sch87] gives a detailed exposition of this procedure.

References

[Ala] A probabilistic analysis of quantified 2sat.

[APT79] Bengt Aspvall, Michael F. Plass, and Robert Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, (3), 1979.

[BFW97] Azer Bestavros and Victor Fay-Wolfe, editors. *Real-Time Database and Information Systems, Research Advances.* Kluwer Academic Publishers, 1997.

[CCPS98] William Cook, William H. Cunningham, William Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization.* John Wiley & Sons, 1998.

[CH99] V. Chandru and J. N. Hooker. *Optimization Methods for Logical Inference.* Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., 1999.

[Che92] Z. Chen. A fast and efficient parallel algorithm for finding a satisfying truth assignment to a 2-cnf formula. *Information Processing Letters*, pages 191–193, 1992.

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

Sun Enterprise 450

Processors	4 UltraSPARC-II 400 Mhz
Cache	4 MB / Processor
Memory	1 GB
Operating System	Solaris 7
Language	C
Software	gcc 2.95.2

Table 8: Implementation System.

```
Function FOURIER-MOTZKIN ELIMINATION ($\mathbf{A}, \vec{\mathbf{b}}$)
1:  for ($i = n$ down to 2) do
2:    Let $\mathbf{I}^+ = \{$ set of constraints that can be written in the form $x_i \geq ()\}$
3:    Let $\mathbf{I}^- = \{$ set of constraints that can be written in the form $x_i \leq ()\}$
4:    for (each constraint $k \in \mathbf{I}^+$) do
5:      for (each constraint $l \in \mathbf{I}^-$) do
6:        Add $k \leq l$ to the original constraints
7:      end for
8:    end for
9:    Delete all constraints containing $x_i$
10: end for
11: if ($a \leq x_1 \leq b, \quad a, b \geq 0$) then
12:   Linear program is consistent
13:   return
14: else
15:   Linear program is inconsistent
16:   return
17: end if
```
Algorithm B.1: The Fourier-Motzkin Elimination Procedure

[Cho00] Seonho Choi. Dynamic time-based scheduling for hard real-time systems. *Journal of Real-Time Systems*, 2000.

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

[CM88] S.A. Cook and M.Luby. A simple parallel algorithm for finding a satisfying truth assignment to a 2-cnf formula. *Information Processing Letters*, pages 141–145, 1988.

[CR92] V. Chvatal and B. Reed. Mick gets some (the odds are on his side) (satisfiability). In IEEE, editor, *33rd Annual Symposium on Foundations of Computer Science: October 24–27, 1992, Pittsburgh, Pennsylvania: proceedings [papers]*, pages 620–627, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1992. IEEE Computer Society Press.

[DE73] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin Elimination and its Dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

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

[Fou24] J. B. J. Fourier. *Reported in: Analyse de travaux de l'Academie Royale des Sciences, pendant l'annee 1824, Partie Mathematique, Historyde l'Academie Royale de Sciences de l'Institue de France 7 (1827)* *xlvii-lv. (Partial English translation in: D.A. Kohler, Translation of a Report by Fourier on his work on Linear Inequalities. Opsearch 10 (1973) 38-42.).* Academic Press, 1824.

[Gav93] F. Gavril. An efficiently solvable graph partition, problem to which many problems are reducible. *Information Processing Letters*, pages 285–290, 1993.

[Gol95] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, June 1995.

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

[HLL90] Tien Huynh, Catherine Lassez, and Jean-Louis Lassez. Fourier Algorithm Revisited. In Hélène Kirchner and W. Wechler, editors, *Proceedings Second International Conference on Algebraic and Logic Programming*, volume 463 of *Lecture Notes in Computer Science*, pages 117–131, Nancy, France, October 1990. Springer-Verlag.

[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.

[LM91] Jean-Louis Lassez and Michael Maher. On fourier's algorithm for linear constraints. *Journal of Automated Reasoning, to appear*, 1991. also IBM tech report (???).

[LW93] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.

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

[SA00] K. Subramani and A. Agrawala. A dual interpretation of "standard constraints" in parametric scheduling. In *Proceedings of the* 6^{th} *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, September 2000.

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

[Sch78] T.J. Schaefer. The complexity of satisfiability problems. In Alfred Aho, editor, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, New York City, NY, 1978. ACM Press.

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

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

[Sub01a] K. Subramani. Parametric scheduling - algorithms & complexity. In *Proceedings of the* 8^{th} *International Conference on High-Performance Computing*, December 2001.

[Sub01b] K. Subramani. Parametric scheduling for network constraints. In *Proceedings of the* 8^{th} *Annual International Computing and Combinatorics Conference*, August 2001.

[Sub01c] K. Subramani. A polynomial time decision procedure for a class of quantified linear programs. Technical Report 2001-0008, Department of Computer Science and Electrical Engineering, West Virginia University, 2001.

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

[WE94] Neil H. Weste and Kamran Eshragian. *Principles of CMOS VLSI Design.* Addison Wesley, 1994.

[Wei97] Volker Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *Applicable Algebra in Engineering, Communication and Computing*, 8(2):85–101, 1997.

[Wil76] H.P. Williams. Fourier-motzkin elimination extension to integer programming. *J. Combinatorial Theory*, (21):118–123, 1976.

[Wil83] H.P. Williams. A characterisation of all feasible solutions to an integer program. *Discrete Appl. Math.*, (5):147–155, 1983.