Properties of chain programs over difference constraints

John E. Argentieri
West Virginia University

Properties of Chain Programs Over Difference Constraints

John E. Argentieri

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2005

Keywords: Difference Constraint Logic, Linear Programming

Properties of Chain Programs Over Difference Constraints

John E. Argentieri

ABSTRACT

Chain Programming is a restricted form of linear programming in which there exists a total ordering over the program variables. In other words, the constraints $x_1 \leq x_2 \leq \ldots \leq x_n$ are either implicitly or explicitly part of the constraint system. At the present juncture, it is not clear whether an arbitrary Chain Program is easier to solve than a linear program, either asymptotically or computationally. This thesis presents a variety of interesting properties pertaining to the case in which a Chain Program is constituted entirely of difference constraints. Deciding the feasibility of a system of difference constraints is a well-studied problem. When a system of difference constraints includes the constraints $x_1 \leq x_2 \leq \ldots \leq x_n$, it is an instance of Chain Programming over Difference Constraints. Existing methods for deciding the feasibility of systems of difference constraints may be used for Chain Programs over Difference Constraints, but the properties of this problem strongly suggest that a more efficient method may exist.

Contents

1 Introduction **1**
1.1 Difference Constraint Logic 3
1.2 Linear Program Formulation 4
1.3 Chain Programming 5

2 Statement of Problem **7**

3 Applications **14**
3.1 Job Shop Scheduling 15
3.2 RDBMS software 17
3.3 Embedded systems software 18

4 Related Work **20**

5 Properties of Chain Programs over Difference Constraints **22**

6 Conclusion **28**

List of Figures

2.1 Edges corresponding to chain constraints are *black* edges, and always zero-valued. 9
2.2 LTR edges are represented in *blue* and RTL edges are represented in *red*. 10
2.3 Simple constraint network containing a negative cycle. 11
2.4 RTL edges with positive weights are redundant. 12
2.5 An LTR edge with negative weight implies existance of a negative cycle. 12
2.6 Constraint network representation of Chain Program corresponding to System 2.2 . 13

3.1 Bounded buffer flow shop with relative timing constraints 15
3.2 Sequential order of executing blocks in a large program 18

5.1 Completely dense constraint network of a CPD with 12 variables. 27

Chapter 1

Introduction

The feat of automating a process is usually directly involved with a synchronization mechanism such as a global clock or message passing interface. In scheduling problems, the global clock can be referenced in order to assert that certain conditions have been met. Similarly, people create schedules for themselves which reference the time of day. The schedules people create for their daily lives are designed under the assumption that they will arise in the morning at a particular time of day. For other applications, it is desirable to have schedules which do not rely on a particular starting time. In order to design such a schedule, no assumptions about the time at which it begins can be made. One benefit of having such a schedule is the ability to monitor or influence the behavior of processes for which the starting time is unknown. Another benefit is the ability to monitor or influence the behavior of processes which happen repeatedly, but for which the overall execution times are permitted to vary to a certain degree.

To design such a schedule, the first action that must be taken is to define a set of variables that will represent points of reference in time. First, let the variable x_1 represent the starting time of the schedule, as opposed to a discrete clock time. Normally, the rest of the times in a person's daily schedule are also discrete clock times, offset from one another by the time taken to complete various tasks, perhaps with some allowance for wasted time. To capture this quality in the schedule being proposed, simply create the necessary number of variables, x_2 through x_n, which can be used as reference points in time at which different events are scheduled to occur.

In a person's schedule, the times at which actions are planned throughout the day can be seen as having one-to-one relationships. For instance, everything a person has scheduled throughout the day is planned to happen a certain amount of time after the person wakes up in the morning. This is not the most expressive or powerful representation of such one-to-one relationships, but it serves the purpose of justifying that a schedule can be defined in such a way. Alternatively, perhaps the action of returning home occurs only after having worked for eight hours. To capture this quality in the new schedule, define one-to-one relationships between the variables that were created. For instance, if x_1 is the time a person wakes up, x_2 is the time a person arrives at work, and it typically takes the person 45 minutes between waking up and arriving at work, define the relationship $x_2 - x_1 = 45$. In reality, this may take the person any amount of time in the range of 40 to 50 minutes from day to day. To create a relationship that more clearly depicts this reality, replace the previously defined relationship with the two relationships $x_2 - x_1 \leq 50$ and $x_2 - x_1 \geq 40$. In this way, a more realistic schedule can be designed which can permit a degree of variation.

An infeasible schedule is one that defines such timing requirements in a way that it is impossible to satisfy them all collectively. Adherence to a schedule that is infeasible is not possible, so the ability to recognize an infeasible schedule is oftentimes crucial to the overall success of a project. In other words, including an infeasible schedule into a project can be entirely crippling to the overall success of that project. *The primary focus of the research documented in this thesis is finding a more efficient method of deciding the feasibility of a special class of this very problem.*

Given the daily schedule of a person, it is typically the case that the person has some sequence of obligations they have planned to fulfill *in order* throughout the day. Likewise, given the schedule of an automated process, it is also typically the case that the schedule demands the system to execute an ordered sequence of tasks under a set of timing requirements.

The problem of interest in this thesis is called Chain Programming over Difference Constraints, a problem which has been introduced by our current research. Chain Programming over Difference Constraints is primarily concerned with determining the feasibility of a difference constraint system over an ordered set of variables. Chain Programming over Difference Constraints will be referred to as 'CPD' throughout this thesis. In order to clearly define exactly what Chain Programming over Difference Constraints entails, the components of the problem and the problem formulation must first be introduced and explained.

1.1 Difference Constraint Logic

When designing a system that must meet various requirements, it may be the case that these requirements may be specified as a minimum or maximum quantity that is permissible between two points of reference. Requirements like these can be specified using difference constraints. A difference constraint is an inequality of the form $x_i - x_j \le b_{ij}$, where b_{ij} is a constant and x_i,x_j are two points of reference. As such, difference constraints are well suited to express temporal requirements. Consider the case in which it is required that at least 20 seconds elapse between x_1 and x_2, but it is also required that no more than 50 seconds elapse. This kind of relationship can be expressed by the conjunction of the two difference constraints $x_2 - x_1 \ge 20$ and $x_2 - x_1 \le 50$.

For reasons that will be discussed throughout this thesis, it is desirable to express every difference constraint in its less-than-or-equal-to form, unlike the previous example. Note that the following types of constraints may be translated into this form [13]:

$$\begin{aligned} x_i - x_j &< b_{ij} \\ x_i - x_j &\ge b_{ij} \\ x_i - x_j &> b_{ij} \\ x_i - x_j &= b_{ij} \end{aligned} \tag{1.1}$$

A collection of m difference constraints defined over a set of n variables is referred to as a difference constraint system (DCS). A solution to a DCS is any n-vector $\vec{\mathbf{x}}$ which satisfies the conjunction of all the constraints in the DCS. It is possible that, given a particular DCS, no such vector exists. Clearly, many problems may present themselves if a collection of requirements is designed for which no satisfactory conditions exist. For instance, it may be essential to the success of a critical system to satisfy the supplied set of requirements at least once. However, this is impossible in the event of an infeasible set of requirements. The design of a set of requirements like these is prone to human error, as the interdependencies are allowed to become arbitrarily complex. It would therefore be beneficial to find a more efficient method of recognizing infeasible difference constraint systems, even for a particular subclass of this problem.

1.2 Linear Program Formulation

Linear Programming refers to the problem of checking whether a conjunction of *linear* constraints is feasible. A linear constraint is an expression of the form: $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n \sim b_i$, where $\sim$ is one of $\{\leq, <, >, \geq, =\}$. Difference constraints are therefore the subset of linear constraints for which the vector $\vec{\mathbf{a}}$ contains only two non-zero elements, one of which must be 1 while the other must be -1. A linear program with m constraints over a set of n variables is expressed as a matrix-vector inequality $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$. A depiction of this linear program formulation is given in System 1.2.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \tag{1.2}$$

In this form, the m rows of the matrix $\mathbf{A}$ are each an n-vector $\vec{\mathbf{a}}$ of coefficients corresponding to one of the m constraints. The n-vector $\vec{\mathbf{x}}$ contains the n variables x_1 through x_n. As such, a solution to a linear program is an n-vector of values for which all of the m constraints are satisfied. The m-vector $\vec{\mathbf{b}}$ contains the right-hand-side constants in each of the m constraints.

Notice that the set of solutions to a single linear constraint is a half-space. A conjunction of several linear constraints is used to represent the solution space of a linear program, which can also be thought of as the intersection of the half-spaces represented by each of the linear constraints. The solution space of a linear program is referred to as its feasible region. It is possible that a subset of these half-spaces do not share even a single point in their intersection. When this is the case, the feasible region given by the system of linear constraints is said to be empty, and the system of linear constraints is said to be infeasible. As additional requirements are added, the solution space can only shrink. Notice that the intersection of the empty set and the universal set is equivalent to the empty set.

Since difference constraints are a subset of linear constraints, the matrix-vector inequality $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$ of a linear program is sufficient to accurately express a DCS. This is one reason that the difference constraints are translated, if needed, into an equivalent less-than-or-equal-to form. When a DCS is expressed in this manner, the rows of the matrix $\mathbf{A}$ retain the property that only two entries are non-zero, of which one must be 1 and the other -1. This formulation is sufficient because a solution $\vec{\mathbf{x}}$ is one which satisfies all of the m difference constraints. Not unlike a system of linear constraints, a system of difference constraints can also have an empty feasible region in the case that the system of difference constraints is infeasible.

1.3 Chain Programming

While the research documented in this thesis is concerned with Chain Programming over Difference Constraints, Chain Programming itself is a broader concept introduced by this research. Chain

Programming is a restricted form of linear programming in that the n variables of a Chain Program have a total ordering imposed on them. In other words, a Chain Program is a linear program with the additional constraints $x_1 \leq x_2 \leq \ldots \leq x_n$ included, either implicitly or explicitly, as part of the constraint system. These additional constraints are referred to as a 'chain'. The chain can be broken into pieces which become equivalent to the chain as a whole when the pieces are treated as a conjunction. For instance, it is obvious that $x_1 \leq x_2 \wedge x_2 \leq x_3$ is equivalent to $x_1 \leq x_2 \leq x_3$. Additionally, $x_1 \leq x_2 \wedge x_2 \leq x_3$ can be restated as $x_1 - x_2 \leq 0 \wedge x_2 - x_3 \leq 0$. Notice that these two constraints are in fact difference constraints of the desired form.

Chain Programs over Difference Constraints are the subset of Chain Programs which are comprised entirely of difference constraints. A Chain Program over Difference Constraints is a difference constraint system which includes or is augmented by the chain constraints in their piecewise form mentioned above. Chain Programs over Difference Constraints are concerned solely with the answer to one seemingly simple question. That question is, *"Is the feasible region given by this system of difference constraints empty or nonempty?"*. Of course, this question has already been asked and answered, but it has never been asked in the context which is about to be defined.

Inasmuch as CPDs are a special class of Difference Constraint Systems, any algorithm for the latter also serves as an algorithm for the former. In particular, the Bellman-Ford procedure [8, 11] which is widely used for solving conjunctions of difference constraints can also be used for solving CPDs. In this method, the DCS is converted to a directed, weighted graph called a constraint network, using a linear time procedure. An application of Farkas' lemma establishes that the DCS is feasible if and only if the corresponding constraint network does not contain a negative cost cycle; further, in the absence of negative cost cycles, the Single Source Shortest Paths in the network serve as a solution to the DCS. However, this algorithm runs in time $O(m \cdot n)$ on a DCS with m constraints on n variables. Since a constraint network may contain $m = n^2$ constraints, this strategy may require time that is cubic in the size of the input. The properties documented here strongly suggest that this time can be improved upon for CPDs.

Chapter 2

Statement of Problem

In this chapter, formal definitions of the various components of Chain Programming over Difference Constraints are provided. The problem of Chain Programming over Difference Constraints is formally defined. Additionally, the construction of a CPD constraint network is detailed and analyzed. The constraint network of a CPD is a graphical representation that can be used as a tool to determine the feasibility of a CPD.

Definition 2.0.1 *A difference constraint is a linear relationship of the form $x_i - x_j \leq b_{ij}$, where b_{ij} is a numerical constant.*

Note that constraints of the form $x_i - x_j \sim b_{ij}$, where $\sim$ is one of $\{<, \geq, >, =\}$, are also called difference constraints, since they can be easily transformed into the form demanded by Definition 2.0.1 in the following manner [13]:

(a) $x_i - x_j < b_{ij}$ is transformed by introducing small ϵ such that $x_i - x_j \leq b_{ij} - \epsilon$.

(b) $x_i - x_j \geq b_{ij}$ is transformed by multiplying by -1 to obtain $x_j - x_i \leq -b_{ij}$.

(c) $x_i - x_j > b_{ij}$ is transformed by multiplying by -1 to obtain $x_j - x_i < -b_{ij}$, which can be transformed as described above.

(d) $x_i - x_j = b_{ij}$ is transformed by splitting the constraint into the two constraints $x_i - x_j \leq b_{ij}$ and $x_i - x_j \geq b_{ij}$, the latter of which can be transformed as described above.

Definition 2.0.2 *A conjunction of difference constraints is called a Difference Constraint System (DCS).*

A DCS is usually expressed in the form of a linear program, as a matrix-vector inequality: $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$, where A is an $m \times n$ matrix, $\vec{\mathbf{b}}$ is an m-vector and $\vec{\mathbf{x}} = [x_1, x_2, \ldots, x_n]^T$ is an n-vector. Note that for a DCS in matrix form, the rows of the matrix $\mathbf{A}$ each have only two non-zero entries; one which is 1 and another which is -1. The product of row i of $\mathbf{A}$ with $\vec{\mathbf{x}}$ is the left-hand side of a difference constraint, and b_i is the corresponding right-hand side.

Definition 2.0.3 *A Chain Program over Difference Constraints(CPD) is a DCS augmented by the constraints $x_1 \leq x_2 \leq \ldots \leq x_n$.*

Observation 2.0.1 *The chain constraints can themselves be expressed as difference constraints; for instance, the constraint $x_1 \leq x_2$, can be expressed as $x_1 - x_2 \leq 0$. Accordingly, the chain constraints can be integrated into the original DCS. Therefore, when a DCS $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$ is referred to as a CPD, it is understood that the chain constraints are present in the constraint system.*

A chain can be broken into the conjunction of several difference constraints. System 2.1 provides the difference constraints whose conjunction is the equivalent of a chain. These constraints must be included in the DCS $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$ in order for the problem to be considered a Chain Program.

$$\begin{gathered} x_1 - x_2 \leq 0 \\ x_2 - x_3 \leq 0 \\ \vdots \\ x_{n-1} - x_n \leq 0 \end{gathered} \tag{2.1}$$

There exist several methods to decide the feasibility of a DCS in the general case. At present, the most efficient of these methods is to convert the DCS into a directed, weighted graph known as a constraint network. Then the Bellman-Ford procedure for the single-source shortest paths problem is run on the constraint network[8]. Chain Programming over Difference Constraints also makes use of a constraint network, whose construction is only slightly different.

Given the CPD $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$, construct the following constraint network ($\mathbf{G} =< \mathbf{V}, \mathbf{E}, \vec{\mathbf{b}} >$) in linear time:

(a) Corresponding to each variable x_i, there is a vertex v_i

(b) Corresponding to each constraint $x_i - x_j \leq b_{ij}$, there is an edge from v_j to v_i with weight b_{ij}.

This construction is different from the construction described in [8], since there is no v_0 vertex; the vertex v_n plays the role of this vertex since there is a path from v_n to every other vertex with zero cost.

The constraint network in Figure 2.1 shows only edges corresponding to the chain constraints. When representing CPDs as constraint networks, these edges are denoted as *black* edges. Because they represent chain constraints such as $x_1 - x_2 \leq 0$, these edges are always zero-valued.

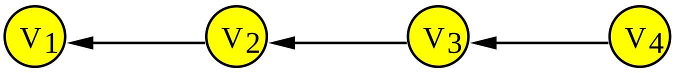

Figure 2.1: Edges corresponding to chain constraints are *black* edges, and always zero-valued.

It is convenient to think of the constraint network $\mathbf{G}$ as being laid out from left to right, with v_1 being the leftmost vertex and v_n being the rightmost vertex. In this representation, every edge can be classified as going from right-to-left (RTL) (from a vertex to a lower-numbered vertex) or from left-to-right (LTR) (from a vertex to a higher-numbered vertex). The adjacency-list representation for $\mathbf{G}$ is assumed, with the added provision that each vertex has a Right list and a Left list. The Right list of vertex v_i contains LTR edges originating from v_i, while its Left list contains RTL edges originating from v_i. When representing CPDs as constraint networks, LTR edges are denoted as *blue* edges and RTL edges as *red* edges. Figure 2.2 provides an example of a constraint network with both LTR and RTL edges.

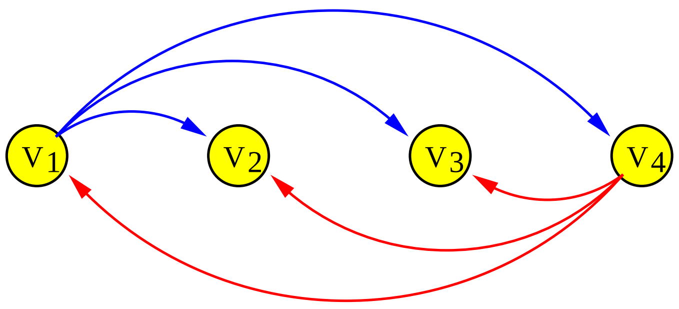

Figure 2.2: LTR edges are represented in *blue* and RTL edges are represented in *red*.

Thus the formal problem statement is as follows: *Given a Chain Program over Difference Constraints, is it feasible?*

Lemma 2.0.1 *If* $\mathbf{G}$ *has a negative cost cycle, then the corresponding CPD* $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$ *is infeasible.*

Proof: Let the negative weight cycle be $c = \langle v_1, v_2, \ldots, v_n, v_1 \rangle$. Now sum up the constraints corresponding to the edges which create the negative cycle:

$$\begin{aligned} x_2 - x_1 &\leq b_{1,2} \\ x_3 - x_2 &\leq b_{2,3} \\ &\vdots \\ x_n - x_{n-1} &\leq b_{n-1,n} \\ x_1 - x_n &\leq b_{n,1} \end{aligned}$$

The left-hand sides of the constraints corresponding to the negative cycle sum up to zero. However, the right-hand sides of the constraints corresponding to the negative cycle sum up to some negative value a, which is the cost of the cycle. This yields the inequality $0 \leq a$, where $a < 0$ must also hold. This contradiction implies that if there is a negative cycle in the constraint network of a CPD, then that CPD is infeasible. This proof is identical to the proof in [8]. □

Consider the example depicted in Figure 2.3. A negative cycle is present in the constraint network between vertices v_2 and v_3. The two edges creating the negative cycle correspond to the constraints $x_2 - x_3 \leq -10$ and $x_3 - x_2 \leq 5$. It can clearly be seen that a negative cycle is given by the corresponding edges, by traversing the edge from v_2 to v_3 of cost 5 then the edge from v_3 to v_2 of cost -10 for a cycle that costs -5. When these two constraints are summed together they yield the inequality $0 \leq -5$, which will always be untrue regardless of any setting for x_2 or x_3. Therefore, the CPD represented by the constraint network in Figure 2.3 must be infeasible.

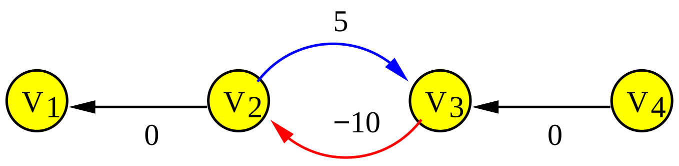

Figure 2.3: Simple constraint network containing a negative cycle.

Lemma 2.0.2 *An RTL edge of positive weight is redundant and can be discarded from* G.

Proof: Let (v_i, v_j) for $j < i$ denote the RTL edge with positive weight. As per the construction outlined above, this edge represents the constraint $x_j - x_i \leq c$, where $c > 0$ represents the weight of the edge. Observe that in a chain program, $x_j \leq x_i$, since $j < i$. Recall that $x_j \leq x_i$ can be restated as $x_j - x_i \leq 0$. Clearly, any setting for x_i and x_j which satisfies $x_j - x_i \leq 0$ also satisfies $x_j - x_i \leq c$ for $c > 0$. Therefore, an RTL edge with positive weight is redundant. $\square$

Consider the example in Figure 2.4 containing an RTL edge with a positive weight. The constraint $x_2 - x_4 \leq 10$ is redundant when compared to the two constraints $x_2 - x_3 \leq 0$ and $x_3 - x_4 \leq 0$, since the sum of these two is $x_2 - x_4 \leq 0$.

Lemma 2.0.3 *An LTR edge of negative weight implies that there exists a negative cost cycle in* G.

Proof: Let (v_i, v_j) for $j > i$ denote the LTR edge with negative weight. As per the construction outlined above, this edge represents the constraint $x_j - x_i \leq c$, where $c < 0$ represents the weight

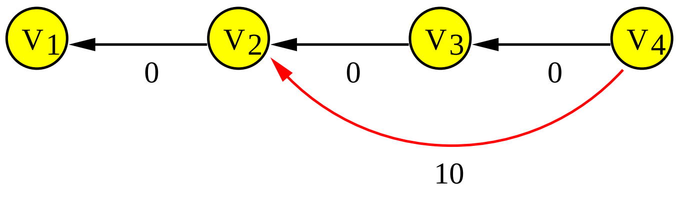

Figure 2.4: RTL edges with positive weights are redundant.

of the edge. Observe that in a chain program, $x_i \leq x_j$, since $i < j$. Recall that $x_i \leq x_j$ can be restated as $x_i - x_j \leq 0$. As per the construction of **G**, this constraint implies that there is a path from v_j to v_i with cost 0. By following the negative cost edge from v_i to v_j, then following the zero cost path from v_j to v_i a negative cycle is completed. The sum of the constraints $x_j - x_i \leq c$ for $c < 0$ and $x_i - x_j \leq 0$ yields the inequality $0 \leq c$ for $c < 0$. This is a contradiction indicating the existence of a negative cycle. Therefore, an LTR edge of negative weight implies the existence of a negative cycle in **G**. $\square$

Consider the example in Figure 2.5 containing an LTR edge with negative weight representing the constraint $x_4 - x_2 \leq -10$. Recall that in a CPD, $x_2 \leq x_4$. Therefore, edges (v_4, v_3) and (v_3, v_2) must each possess a weight of no more than zero. In other words, there will always be a path from right to left which costs at most zero. This is why introducing an LTR edge with negative weight will always create a negative cycle.

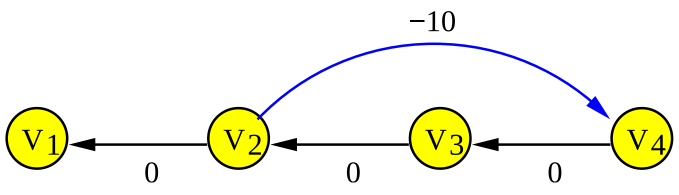

Figure 2.5: An LTR edge with negative weight implies existence of a negative cycle.

From Lemma 2.0.3 and Lemma 2.0.1, it is clear that if the constraint network corresponding to a chain program has a negative cost LTR edge, then the chain program is infeasible.

System 2.2 and Figure 2.6 represent an instance of a CPD in the form of a matrix-vector inequality ($\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$) and the corresponding constraint network. The first three constraints in the matrix form correspond to the chain constraints. It is important to note that redundant constraints are not represented in the constraint network; for instance the constraint $x_1 \leq x_2$ is made redundant by the constraint $x_1 - x_2 \leq -12$, since any assignment satisfying the latter will satisfy the former. Accordingly, only $x_1 - x_2 \leq -12$ is represented in the constraint network.

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 10 \\ -12 \\ 20 \\ 5 \\ 3 \\ -2 \end{pmatrix} \tag{2.2}$$

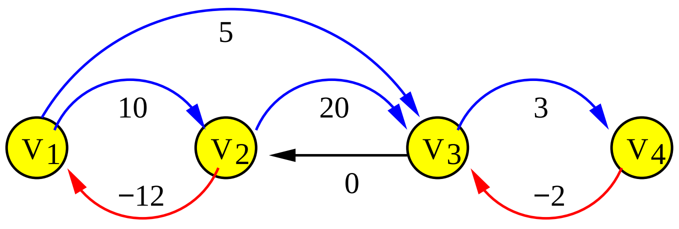

Figure 2.6: Constraint network representation of Chain Program corresponding to System 2.2

Chapter 3

Applications

This chapter provides examples of problems in real-world design which could be captured by Chain Programs over Difference Constraints (CPDs). For each of the examples, a description of the system and the desired outcomes are given. An explanation of how and why a CPD is sufficient for reaching the desired outcome follows each of the descriptions of the systems.

At this point, it is important to note that any problem that can be solved by a Chain Program over Difference Constraints can also be solved as a Difference Constraint System. Deciding the feasibility of a DCS is a problem that is well-studied for which the Bellman-Ford procedure mentioned in Chapter 1 is currently considered the optimal method with a running time of $O(m \cdot n)$. While the research documented in this thesis does not provide an algorithm for solving a CPD, it does present evidence that strongly suggests that solving a CPD may be easier than solving a DCS. If this is actually the case, then a more efficient algorithm which solves the following problems will likely exist.

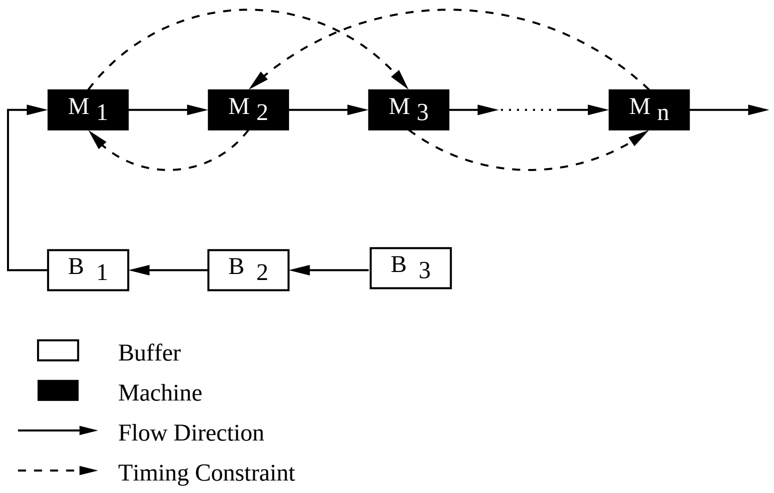

Figure 3.1: Bounded buffer flow shop with relative timing constraints

3.1 Job Shop Scheduling

Figure (3.1) 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 B_1, B_2 and B_3. Objects to be tooled also called *jobs* are placed in these buffers. The time line on which the flow shop operates is divided into equal length portions called *periods*. At the start of each period, the job in each buffer moves to the buffer ahead of it, while the job in the first buffer (Buffer B_1) 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. 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. As stated, the timing constraints are relevant to each job on an individual basis. Under these conditions, jobs may have a degree of variance, but there exists only one set of timing constraints which every job must respect.

Design Problem: Given the timing constraints between the machines and the process time of each machine, does there exist a valid schedule i.e., a schedule that respects the timing constraints? In other words, given a particular schedule which is supposed to automate this process, is the given schedule feasible?

Note that the operation of a flow-shop such as the one described here is an entirely sequential process. No job, once it has visited machine M_i will ever revisit machine M_i. Furthermore, any job which leaves machine M_i must subsequently visit machine M_{i+1} in particular. Even in the case that every job is different, and perhaps machine M_i has nothing to do on a particular job, the job may idle at that machine.

The flow-shop example in this section is easily modeled as a Chain Program over difference constraints, since we must have $s_1 \leq s_2 \leq \ldots \leq s_n$.

To model this flow-shop example, let s_1 through s_n be the set of variables x_1 through x_n in the CPD $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$. Given the m difference constraints that are specified as part of the flow-shop design, let each row of matrix $\mathbf{A}$ contain the coefficients of each of the m difference constraints. Note that for a difference constraint, the only non-zero coefficients are a positive 1 for the positive variable and a negative 1 for the negative variable. Finally, augment the matrix $\mathbf{A}$ with the $n - 1$ chain constraints $x_1 - x_2 \leq 0$ through $x_{n-1} - x_n \leq 0$.

Given the CPD $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$ that has been constructed as described, if the solution to the CPD is yes, then the system of difference constraints must be feasible. If this is the case, then it must be true that there is some set of times s_1 through s_n such that $s_1 \leq s_2 \leq \ldots \leq s_n$ that also respects all of the m timing requirements (expressed as difference constraints) that are given as part of the supposed schedule for the flow-shop. Therefore, a CPD is sufficient to determine whether or not the flow-shop is schedulable.

3.2 RDBMS software

Real-time database management software requires careful synchronization, especially in an environment where distributed access is permitted. For instance, trading accounts in financial markets represent one such application area of real-time databases [4]. In financial trading, certain conditions are likely to influence the behavior of human traders. For example, if the price of a stock begins to fall, there is a likelihood that traders will rush to sell their shares of that particular stock.

For trading account databases, each trade request is broken down into a number of sub-requests, viz., *permit()*, which checks whether the trader is allowed to execute the requested trade, *balance()*, which checks whether the trader has the required funds in his account, *execute()*, which actually executes the trade and reports the results, *consistency-check()*, which synchronizes the databases and so on.

Each of these operations may have individual timing requirements concerning any of the other operations. However, for any trade request, a certain ordered set of these operations is known. If there are n such operations, let the variables s_1 through s_n represent the start times of each of the n operations. Note that since the operations are a sequence, it must be the case that $s_1 \leq s_2 \leq \ldots \leq s_n$.

To model the trade request as the CPD $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$, let the variables x_1 through x_n represent the starting times of the operations that make up the request s_1 through s_n, respectively. Let the m rows of the matrix $\mathbf{A}$ take on the values of the coefficients in the m timing requirements specified over the operations. Note that since timing requirements are specified as difference constraints, each row of $\mathbf{A}$ will contain exactly two non-zero entries, of which one must be -1 and the other 1. Augment the matrix $\mathbf{A}$ with the additional $n-1$ rows representing the chain constraints, $x_1 - x_2 \leq 0$ through $x_{n-1} - x_n \leq 0$.

For the CPD constructed as described above, if the solution to the CPD is yes, then the set of difference constraints (timing constraints and chain constraints) must be feasible. If this is indeed the case, then some set of values for the start times of the operations s_1 through s_n must exist such that $s_1 \leq s_2 \leq \ldots \leq s_n$ which also respects all of the timing requirements. In other words, the trade request may be scheduled as proposed.

Likewise, any large computer program is composed of blocks that execute in strict sequential order and the sequencing order is fixed and known in advance. Timing constraints exist between blocks on account of real-time requirements. These requirements can be converted into difference constraints (See Figure 3.2). In a similar fashion, the feasibility of real-time requirements over an ordered set of blocks of code in a large computer program can be decided by modeling the problem as a Chain Program over Difference Constraints.

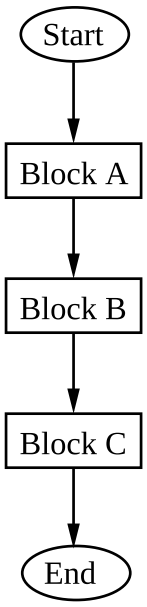

Figure 3.2: Sequential order of executing blocks in a large program

3.3 Embedded systems software

A discussion of the design of an embedded controller for a real-time coffee machine is given in [14]. In this machine, there is a continuous operation that begins with the user selecting the number of coffee cups that he wants. The process of delivering the coffee to the user is constituted of a number of sub-tasks. For instance, there is a task that decides how much coffee is to be released; a second task that controls the creamer amount and yet another task that controls the sugar levels. All these sub-tasks are performed in a strict sequential order. Further, there are relative timing constraints between these sub-tasks of the form:

(a) Release the creamer at least 8 seconds after the coffee has been poured;

(b) Move the coffee cup within 2 inches of the sugar orifice (Distance constraints can be converted into timing requirements.)

Notice that the actions the machine takes to prepare a cup of coffee are strictly sequential. Like the previous two problems, let the start times of those actions be represented by the variables s_1 through s_n. Since the actions are strictly sequential, it holds that $s_1 \leq s_2 \leq \ldots \leq s_n$. These are the perfect conditions for modeling the problem as a CPD.

The construction of the CPD is identical to the previous examples. Given the CPD construction of the supplied requirements, is the schedule for automating coffee dispensing feasible? If the solution to this CPD is yes, then there must exist a set of times s_1 through s_n such that $s_1 \leq s_2 \leq \ldots \leq s_n$ which respects all the timing requirements in the system.

As the above examples demonstrate, Chain Programs over difference constraints occur in a number of natural examples and an analysis of their properties is justified.

Chapter 4

Related Work

Difference Constraint Logic (DCL) has been part of the Artificial Intelligence (AI) and Model-checking communities for quite some time [9, 6] on account of its expressiveness and flexibility. From the perspective of AI, the Simple Temporal Problem (STP) is one of the fundamental problems in temporal reasoning. The Simple Temporal Problem is the problem of deciding the feasibility of a schedule expressed as a set of temporal requirements, which is identical to a system of difference constraints. Additionally, conjunctions of difference constraints have been used to express and solve a number of problems in real-time scheduling [10, 27].

In model-checking, DCL is used to express constraints on the transitions of Timed Automata [1] and also safety and liveness properties [15, 17]. By using difference constraints as conditions on the transitions of timed automata, a potentially large number of states is represented in one simple form. This is referred to as symbolic representation, where the subset of states that is represented symbolically contains only those states which satisfy the conditional transition.

Conjunctions of difference constraints are used to capture requirements in a number of application domains such as Symbolic Model checking [7], Verification of Timed Systems [29, 16], Timed Automata [1, 24] and so on. Difference Constraint feasibility has also been studied as the Single Source Shortest Paths problem within the Operations Research and Algorithms communities [11]. Additionally, separation relationships in a number of scheduling problems are captured through difference constraints [23, 5, 12]. In real-time software, temporal requirements are modeled through variants of difference constraints [18, 19].

It is well known that the problem of feasibility checking in arbitrary boolean combinations of Difference constraints is `NP-complete` [2]. SAT-based procedures have found reasonable success in solving practical instances of this class of problem [7, 25]. Many of these procedures have also been implemented as part of practical systems such as UPPAL [3].

From the perspective of pure conjunctions only of difference constraints, there has been a lot of work within the Operations Research and Theoretical Computer Science communities, inasmuch as DCS solving is closely connected to the The Single Source Shortest Path (SSSP) problem (in the presence of negative weights). Most of the approaches in the literature use a variant of the Bellman-Ford approach (Dynamic Programming) for this problem, although there exist other approaches like greedy algorithms [28].

Special-purpose approaches for the SSSP problem have also been designed and enjoyed reasonable success [20, 21, 22]. Each of these approaches is designed for problem instances occurring in a specific domain; for instance [20] performed very well on transportation networks. To the best of our knowledge Chain Programing over Difference constraints has not been addressed in the literature.

Chapter 5

Properties of Chain Programs over Difference Constraints

This chapter discusses the various properties of Chain Programs over Difference Constraints that were discovered after analyzing the problem as defined in Chapter 2. These properties were drawn from the structure of a CPD, after considering the range of possibilities for difference constraints in the system which are also edges in the constraint network. The properties discovered are clearly defined, proved and explained here.

Let $\delta(v_i, v_j)$ denote the shortest path cost between vertices v_i and v_j in the constraint network $G = \langle V, E, \vec{\mathbf{b}} \rangle$ corresponding to a CPD $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$.

Let

$$M = \begin{bmatrix} \delta(v_1, v_1) & \delta(v_1, v_2) & \dots & \delta(v_1, v_n) \\ \delta(v_2, v_1) & \delta(v_2, v_2) & \dots & \delta(v_2, v_n) \\ \vdots & \vdots & \dots & \vdots \\ \delta(v_n, v_1) & \delta(v_n, v_2) & \dots & \delta(v_n, v_n) \end{bmatrix} \tag{5.1}$$

denote the matrix of shortest path distances.

Theorem 5.0.1 *The CPD $\mathbf{A} \cdot \vec{\mathbf{x}} \leq \vec{\mathbf{b}}$ is infeasible if and only if $\exists\ v_i, v_j$ such that $j > i$ and $\delta(v_i, v_j) < 0$.*

Proof: If there exist vertices v_i, v_j such that $j > i$ and $\delta(v_i, v_j) < 0$, then there must exist a negative cycle between vertices v_i, v_j. To construct one such negative cycle, simply follow the path that gives $\delta(v_i, v_j) < 0$, then complete the cycle by following black zero-cost edges from v_j to v_i. There must be such a path since $x_i \leq x_j$ in a CPD. From Farkas' lemma, it follows that the CPD is infeasible.

Consider the case in which the constraint network contains a negative cycle, and is therefore infeasible. Let v_a be the left-most vertex in the cycle as per the description of a left-to-right layout in Chapter 2. The negative cycle must also have a right-most vertex v_b, which is not necessarily identical to v_{a+1}. However, there must exist a negative cost path from v_a to v_{a+1} which can be found by traversing the negative cycle as many times as necessary, then following black zero-cost edges from v_b to v_{a+1}. The same is true for any vertex v_k for $a < k \leq b - 1$. □

Corollary 5.0.1 *Let G be a network with a negative cycle. Let v_i be the left-most vertex in the cycle, and v_k be the right-most vertex in the cycle. There exists a negative cost path from v_i to v_j for $j = i + 1 \ldots k$ which can be obtained by traversing the negative cycle as many times as are necessary to cancel any positive cost from v_i to v_k, then following the 0-cost black edges.*

Theorem 5.0.2 *In the absence of negative cycles, for any vertex v_j,*

$$\delta(v_i, v_j) \geq \delta(v_{i+k}, v_j)\ k = 1, 2, \ldots n - i.$$

Proof: Clearly, v_i lies to the left of v_{i+k} since $x_i \leq x_{i+k}$ in a Chain Program. There are three possibilities for the relative location of v_j. The truth of Theorem 5.0.2 must be proved for each of these possibilities in order for correctness to be established.

The first possibility occurs when $j < i < i + k$. In other words, v_j lies to the left of both v_i and v_{i+k}. There must exist a path of cost 0 from v_{i+k} to v_i. Therefore, there must exist a path from v_{i+k} to v_j with cost $\delta(v_i, v_j)$, by following the zero-cost path from v_{i+k} to v_i then following the shortest path from v_i to v_j. Either this path is the shortest path from v_{i+k} to v_j or the true shortest path costs even less than $\delta(v_i, v_j)$. Therefore, Theorem 5.0.2 is true when $j < i < i + k$.

The second possibility occurs when $i < j < i + k$. In this case, any path from v_i to v_j must be non-negative. It has been shown that a negative path from left-to-right implies the existence of a negative cycle, and Theorem 5.0.2 applies to CPDs in the absence of negative cycles. While a path from v_{i+k} to v_j can be positive, such a path has been shown to be redundant. Further, there exists a path from v_{i+k} to v_j of cost 0. If this is not the shortest path, $\delta(v_{i+k}, v_j)$ must be less than 0. Therefore, Theorem 5.0.2 is true when $i < j < i + k$.

The third and final possibility occurs when $i < i + k < j$. If there exists a path from v_i to v_j, then there exists a path from v_{i+k} to v_j, which can be found by first following the zero-cost path from v_{i+k} to v_i, and then following the same path from v_i to v_j. Therefore, Theorem 5.0.2 is true when $i < i + k < j$. □

In other words, every column of Matrix 5.1 is linearly ordered, with the largest element being at the top and the smallest element being at the bottom.

Theorem 5.0.3 *In the absence of negative cycles, for any vertex v_j,*

$$\delta(v_j, v_i) \leq \delta(v_j, v_{i+k}) \ k = 1, 2, \ldots n - i.$$

Proof: Clearly, v_i lies to the left of v_{i+k} since $x_i \leq x_{i+k}$ in a Chain Program. There are three possibilities for the relative location of v_j. The truth of Theorem 5.0.3 must be proved for each of these possibilities in order for correctness to be established.

The first possibility occurs when $j < i < i + k$. If a path exists from v_j to v_{i+k}, then there must also exist a path of equal value from v_j to v_i, by following the path from v_j to v_{i+k} then following the zero-cost path from v_{i+k} to v_i. Given the shortest path from v_j to v_{i+k}, there must also exist a path from v_j to v_i with cost $\delta(v_j, v_{i+k})$. If this path is not the shortest path from v_j to v_i, then the true shortest path must be less than $\delta(v_j, v_{i+k})$. Therefore, Theorem 5.0.3 is true when $j < i < i + k$.

The second possibility occurs when $i < j < i + k$. In this case, any path from v_j to v_{i+k} must be non-negative. It has been shown that a negative path from left-to-right implies the existence of a negative cycle, and Theorem 5.0.3 applies to CPDs in the absence of negative cycles. While a path from v_j to v_i can be positive, such a path has been shown to be redundant. Further, there exists a path from v_j to v_i of cost 0. If this is not the shortest path, $\delta(v_{i+k}, v_j)$ must be less than 0. Therefore, Theorem 5.0.3 is true when $i < j < i + k$.

The third and final possibility occurs when $i < i + k < j$. If there exists a path from v_j to v_{i+k}, then there exists a path of equal cost from v_j to v_i, found by following the given path from v_j to v_{i+k} then following the zero-cost path from v_{i+k} to v_i. Likewise, given the shortest path from v_j to v_{i+k}, there exists a path from v_j to v_i with cost $\delta(v_j, v_{i+k})$. If this is not the true shortest path from v_j to v_i, then the true shortest path must be less than $\delta(v_j, v_i)$. Therefore, Theorem 5.0.3 is true when $i < i + k < j$. $\square$

In other words, each row is linearly ordered with the smallest element at the left and the largest element at the right.

Corollary 5.0.2 *In the absence of negative cost cycles, $\delta(v_n, v_1)$ is the smallest element of M and $\delta(v_1, v_n)$ is the largest element.*

Lemma 5.0.1 *In the absence of negative cost cycles, all the entries below the diagonal of Matrix 5.1 are non-positive, the diagonal entries are zero and the entries above the diagonal are non-negative.*

Proof: Clearly, in the absence of negative cycles, the shortest path from a vertex to itself has cost 0. Therefore, the diagonal entries of Matrix 5.1 must be 0. It follows from Theorem 5.0.2 that entries above the diagonal are non-negative and entries below the diagonal are non-negative. $\square$

Lemma 5.0.2 *Let v_i, v_j, v_k denote three vertices in G such that $k < j < i$. If $c_{ik}, c_{ij} \neq \infty$, $c_{ik} \leq c_{ij}$.*

Proof: Observe that edge (v_i, v_k) represents the constraint $x_k - x_i \leq c_{ik}$. Likewise, edge (v_i, v_j) represents the constraint $x_j - x_i \leq c_{ij}$. It is given that $x_k \leq x_j$, so it follows that $c_{ik} \leq c_{ij}$. $\square$

Lemma 5.0.3 *Let v_i, v_j, v_k denote three vertices in G such that $k < j < i$. If $c_{ki}, c_{ji} \neq \infty$, $c_{ji} \leq c_{ki}$.*

Proof: Observe that edge (v_k, v_i) represents the constraint $x_i - x_k \leq c_{ki}$. Likewise, edge (v_j, v_i) represents the constraint $x_i - x_j \leq c_{ji}$. It is given that $x_k \leq x_j$, so it follows that $c_{ji} \leq c_{ki}$. $\square$

Lemma 5.0.4 *Let v_i, v_j, v_k denote three vertices in G such that $i < k < j$. If $c_{ik}, c_{ij} \neq \infty$, $c_{ik} \leq c_{ij}$.*

Proof: Observe that edge (v_i, v_k) represents the constraint $x_k - x_i \leq c_{ik}$. Likewise, edge (v_i, v_j) represents the constraint $x_j - x_i \leq c_{ij}$. It is given that $x_k \leq x_j$, so it follows that $c_{ik} \leq c_{ij}$. $\square$

Lemma 5.0.5 *Let v_i, v_j, v_k denote three vertices in G such that $i < k < j$. If $c_{ki}, c_{ji} \neq \infty$, $c_{ji} \leq c_{ki}$.*

Proof: Observe that edge (v_k, v_i) represents the constraint $x_i - x_k \leq c_{ki}$. Likewise, edge (v_j, v_i) represents the constraint $x_i - x_j \leq c_{ji}$. It is given that $x_k \leq x_j$, so it follows that $c_{ji} \leq c_{ki}$. $\square$

Finally, Figure 5.1 provides an example of the form of a completely dense Chain Program over Difference Constraints. In this example, there are a total of 12 variables in the constraint system. At its greatest density, a constraint network corresponding to a CPD has nearly n^2 edges for a CPD with n variables.

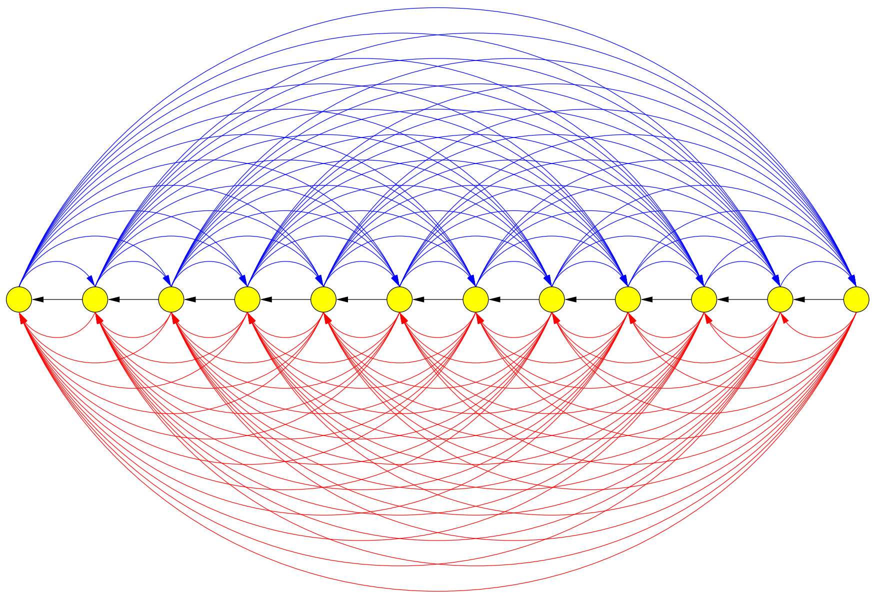

Figure 5.1: Completely dense constraint network of a CPD with 12 variables.

Chapter 6

Conclusion

This thesis introduces the concept of Chain Programming, a restricted form of linear programming in which a total ordering is imposed over the variables such that $x_1 \leq x_2 \leq \ldots \leq x_n$. However, the primary topic of this thesis is the problem of Chain Programming over Difference Constraints (CPD), a case of Chain Programming in which the system of constraints is constituted entirely of difference constraints. A detailed analysis of the properties and structure of CPD problems is also established by this thesis.

The conditions necessary for Chain Programming occur in a number of practical situations in real-time scheduling and verification problems. Furthermore, computer science techniques and the process of computation tends toward predetermined sequential instructions. It is therefore likely that there are many more applications for Chain Programming that have yet to be discovered. Difference constraints can be applied to sequences of instructions to achieve synchronization, while CPD can be utilized to decide the feasibility of the synchronized schedule more efficiently.

The imposition of a total ordering over the variables in a linear program is called the chain condition, i.e. $x_1 \leq x_2 \leq \ldots \leq x_n$. Given the vast changes in structure introduced by the chain condition, there may exist a method to exploit the various properties of CPDs and their structure to decide their feasibility more efficiently. At this juncture, no algorithm exists for CPD which runs in lesser time than the Bellman-Ford procedure, which has a worst-case running time of $O(m \cdot n)$. However, the properties of CPD documented in this thesis strongly suggest that a more efficient algorithm may exist. The Bellman-Ford procedure can be run on any system of difference constraints, but CPD problems exhibit far greater structure than systems of difference constraints in the general case. If there does exist some method of exploiting the additional properties seen in CPDs, then this knowledge would permit the design of an algorithm which is more efficient in the worst case at deciding CPDs than the Bellman-Ford procedure.

Any algorithm designed to decide the feasibility of CPDs can also use the existing data which the Bellman-Ford procedure is currently interfacing to. Thus, if conditions for deciding the feasibility of a system of difference constraints as a Chain Program exist in the problem, an algorithm designed specifically for solving CPD can be integrated into existing tools that deal with difference logic. Therefore, any existing projects which would benefit from an efficient algorithm for Chain Programming over Difference Constraints could adopt such a strategy with great ease.

The research documented in this thesis gives rise to several interesting open problems. This thesis introduces the concept of Chain Programming in general before analyzing Chain Programming over Difference Constraints. Chain Programming applies to all forms of linear and integer programming. It is unknown as of yet whether applying Chain Programming concepts to general linear and integer programs would yield similar properties which could be exploited to find a more efficient algorithm for feasibility deciding.

It is well-known that the problem of integer programming is `NP-complete`. Perhaps a strategy extending Chain Programming to integer programs would yield an improved algorithm for all integer programs. For instance, given a set of n variables, there exist only $n!$ or $n(n-1)/2$ orderings over those variables. If a polynomial-time algorithm for Chain Programming over Integer Programs is found, perhaps that algorithm could be run on each of the variable orderings to yield some interesting results.

It is unknown whether the concept of Chain Programming could be extended to other *restricted* forms of linear and integer programs. For instance, linear and integer programs with two variables per constraint have been called LP(2) and IP(2). The types of constraints that these problems include are similar to difference constraints in that they both have exactly two variables. However, LP(2) and IP(2) problems are permitted to use constants other than 1 and -1 as coefficients of variables. Additionally, LP(2) and IP(2) permit the use of the addition and subtraction operators.

Every linear program has a dual problem, found by swapping the coefficients of the objective function with the constants in the vector $\vec{b}$, and then changing the less-than-or-equal-to to greater-than-or-equal-to in each of the constraints. Linear programming problems and their dual problems are interestingly related in that the optimal answer in a maximization linear program is the same as the optimal answer of its minimization dual problem and vice versa. Perhaps the concepts of Chain Programming can be applied to the duals of these problems to determine their feasibility more efficiently.

Current trends in hardware and software are approaching processor cores with multiple threads of execution, distributed computing, and supercomputers. Perhaps the properties of Chain Programs will more easily permit the design of a distributed algorithm when compared to the Bellman-Ford procedure. In other words, the Bellman-Ford procedure is not readily parallelized, but perhaps the properties of Chain Programs over Difference Constraints will permit a parallelized approach to deciding the feasibility of CPDs. Perhaps the findings of such an experiment will yield more efficient ways of solving other kinds of linear and integer problems in a parallelized environment.

Bibliography

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 25 April 1994. Fundamental Study.

[2] Alessandro Armando, Claudio Castellini, and Enrico Giunchiglia. Sat-based procedures for temporal reasoning. In *ECP*, pages 97–108, 1999.

[3] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on uppaal. In *SFM*, pages 200–236, 2004.

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

[5] P. Brucker. *Scheduling Algorithms*. Springer, 1998. 2^{nd} edition.

[6] Edmund M. Clarke. Automatic verification of sequential circuit designs. In David Agnew, Luc Claesen, and Raul Camposano, editors, *Proceedings of the 11th International Conference on Computer Hardware Description Languages and their Applications (CHDL'93)*, volume 32 of *IFIP Transactions A: Computer Science and Technology*, pages 163–166, Amsterdam, The Netherlands, April 1993. North-Holland.

[7] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill Book Company, Boston, Massachusetts, $2nd$ edition, 1992.

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

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

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

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

[13] James P. Ignizio and Tom P. Cavalier. *Linear Programming*. Prentice Hall, 1993.

[14] Kim G. Larsen, B. Steffen, and C. Weise. Continuous modelling of real time and hybrid systems. Technical report, Aalborg Universitet, 2003. BRICS Technical Report.

[15] J. Møller, J. Lichternberg, H. R. Andersen, and H. Hulgaard. On the symbolic verification of timed systems. Technical report, Technical University of Denmark, 2003.

[16] Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. Difference decision diagrams. In *CSL*, pages 111–125, 1999.

[17] Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. *Electr. Notes Theor. Comput. Sci.*, 23(2), 1999.

[18] N. Muscettola, B. Smith, S. Chien, C. Fry, G. Rabideau, K. Rajan, and D. Yan. In-board planning for autonomous spacecraft. In *The Fourth International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*, July 1997.

[19] Nicola Muscettola, Paul Morris, Barney Pell, and Ben Smith. Issues in temporal reasoning for autonomous control systems. In *The Second International Conference on Autonomous Agents*, Minneapolis, MI, 1998.

[20] Pallottino and Scutella. Dual algorithms for the shortest path tree problem. *NETWORKS: Networks: An International Journal*, 29, 1997.

[21] S. Pallottino. Shortest path methods: Complexity, interrelations and new propositions. *NETWORKS: Networks: An International Journal*, 14:257–267, 1984.

[22] U. Pape. Implementation and efficiency of moore algorithms for the shortest root problem. *Mathematical Programming*, 7:212–222, 1974.

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

[24] J. I. Rasmussen, Kim Gulstand Larsen, and K. Subramani. Resource-optimal scheduling using priced timed automata. In Kurt Jensen and Andreas Podelski, editors, *Proceedings of the* 10^{th} *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2988 of *Lecture Notes in Computer Science*, pages 220–235. Springer-Verlag, April 2004.

[25] Ofer Strichman, Sanjit A. Seshia, and Randal E. Bryant. Deciding separation formulas with sat. In *CAV*, pages 209–222, 2002.

[26] K. Subramani. An analysis of zero-clairvoyant scheduling. In Joost-Pieter Katoen and Perdita Stevens, editors, *Proceedings of the* 8^{th} *International Conference on Tools and Algorithms for*

the construction of Systems (TACAS), volume 2280 of *Lecture Notes in Computer Science*, pages 98–112. Springer-Verlag, April 2002.

[27] K. Subramani. An analysis of totally clairvoyant scheduling. *Journal of Scheduling*, 8(2):113–133, 2005.

[28] K. Subramani and L. Kovalchick. A greedy strategy for detecting negative cost cycles in networks. *Future Generation Computer Systems*, 21(4):607–623, 2005.

[29] Anders Wall, Kristian Sandström, Jukka Mäki-Turja, Christer Norström, and Wang Yi. Verifying temporal constraints on data in multi-rate transactions using timed automata. In *RTCSA*, pages 263–270, 2000.