2-2006

Culprit Detection on Incremental Tractable Qualitative Temporal Reasoning Problems

Florent Marie Launay

Debasis Mitra

Technical report CS-2006-02

Culprit Detection on Incremental Tractable Qualitative Temporal Reasoning Problems

by

Florent Marie Launay and Dr. Debasis Mitra

Bachelor of Science
In Computer Science
Florida Institute of Technology
2002

Melbourne, Florida
Spring, 2006

Abstract

Title:

"Culprit Detection on Incremental Tractable

Qualitative Temporal Reasoning Problems"

Author:

Florent Marie Launay

Major Advisor:

Debasis Mitra, Ph.D.

This thesis is a confluence of three problems in constraint reasoning: qualitative temporal reasoning (QTR), incremental reasoning, and explanation generation. We first address a set of algorithms to solve the QTR for point algebra (PA) with explanation. Next, we turn to the tractable form of Allen's Interval Algebra (IA). For both problems, an incremental version of the problem, where a new temporal object is added to a set of objects committed on the time-line, is being studied. We provide a model or solution space for the new object, if it exists, otherwise, for (PA) and a particular subalgebra of (IA) called ORD-Horn, we provide explanation for inconsistency. Both the problems of incremental reasoning and explanation generation are based on some recent investigation in constraint programming.

Table of Contents

List of Figures

Acknowledgement

I would like to thank the Computer Science faculty of Florida Tech, most particularly Dr Debasis Mitra for his support, his advices and his patience. I would also like to express my gratitude to my family that always supported me morally and financially and pushed me to give the better of myself while always respecting my choices. Finally, I would like to thank Dr. Gerard Ligozat (LIMSI university of Paris) for the discussions we had and who helped me to better understand some aspects of my work.

Dedication

This Thesis is dedicated to my little sister Laurie-Anne and my little niece Maya

Chapter 1 – Introduction

1.1 – Problem Overview

Along with the exponential grow of complexity in recent technologies comes a need to more powerful computational techniques capable of solving increasingly complex problems in reasonable time. A slight improvement in the complexity of an algorithm can have significant effects on the overall problem resolution. Temporal Representation and Reasoning in AI often provide ways to compute complex problems and sometimes find the 'best' imperfect solution. There exist a growing interest in the scientific community for modeling and reasoning on spatio-temporal problems.

1.2 – Thesis organization

This thesis is divided into 7 chapters. The first two chapter chapters will provide the reader with essential background information needed to understand the problem addressed in this thesis and the suggestions we brought to solve a particular cast of scheduling problems. Chapter 3 formally describes the problem and proposes a solution for the Online Qualitative Temporal Reasoning on Point algebra. This section also introduces our approach in defining the "causes" for inconsistency and the algorithms to find it. Subsequently we have enhanced the approach toward online reasoning with time-intervals which is divided into two distinct sub problems. Chapter 4 deals with one of these problems known as the ORD-Horn Interval Algebra and describes a series of algorithms to solve this problem instance. Then we turn to a more general approach of tractable temporal

reasoning in chapter 5. This chapter also proposes a set of algorithms to solve the more general tractable temporal reasoning case. Chapter 6 generalizes again the Interval algebra's approach with a discussion on non-tractable cases. Finally we conclude this thesis with a discussion on past and current interest in the scientific community for solving the type of problems addressed here.

1.3 – Contribution

In many cases, no follow up intervenes after inconsistency detection in spatio-temporal reasoning problems. This thesis proposes such 'incomplete' solutions for a particular set of temporal reasoning problems when no perfect solution can be addressed. This partial solution is defined in a systematic way for the point based reasoning, and subject to tractability limitation in case of interval reasoning. Whereas many study focused on finding the maximal set of consistent constraints, the problem of finding the minimal set of culprit relations is new to the community. Section 3.2.1 proposes an algorithm to solve such problem for the point based reasoning and provides a proof of correctness and minimality of the output set. Finally, section 5.3 gives a graphical representation of 17 out of the 18 maximal tractable algebras of the full interval algebra, providing an alternative way to reason on the algebras, as well as an easy way to count them.

1.4 – Publications

The work addressed in this thesis has partially been published in:

1. Debasis Mitra, and Florent Launay: On-line Qualitative Temporal Reasoning with Explanation. FLAIRS 2006
2. Florent Launay, and Debasis Mitra: Incrementally Scheduling with Qualitative Temporal Information. IEA/AIE 2005: 229-231
3. Florent Launay, and Debasis Mitra: An optimum greedy algorithm for choosing minimal set of conflicting constraints in the point sequencing problem. Technical Report on http://www.cs.fit.edu/~tr/cs-2002-16.doc.

Chapter 2 – Background

2.1 – Temporal Reasoning

Temporal Reasoning Problem can be viewed as finding an assignment for a set of constrained temporal events. The next two sections describe two particular instances of TR. Both algebras are the central topics under discussion of the research.

2.1.1 Point Reasoning

The point-based temporal reasoning constitutes the simplest form of spatio-temporal reasoning (Allen). The scheme has three basic relations {<, >, =}.

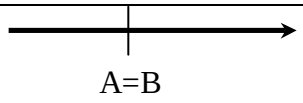
Binary operator	Relation type	Example	Representation
<	Before	A (<) B	
>	After	A (>) B	
=	Equal	A (=) B	

Figure 2.1: Basic relation on Point Reasoning

The corresponding relational algebra is comprised of the power set of the basic relations, thus containing 8 elements:

- {(<), (>), (=), (<=), (>=), (<>), (<=>), ()}

The three basic relations, the relation less-than-or-equal (<=), the relation greater-than-or-equal (>=), the relation different-from (<>), the relation any-relation (<=>), and the relation no-relation () stating that no relation should exist between two points. Thus, a constraint between two points A and B expressed as: (A <= B) has a valid assignment for A and B if either (A = B) *or* (A < B). The composition between basic relations is a disjunction of basic constraints.

The composition table of the point based relation is described in the table below:

.	<	=	>
<	<	<	<=>
=	<	=	>
>	<=>	>	>

Figure 2.2: Composition table of point relations

This set of relations forms an algebra since it is closed under the traditional reasoning operators; composition, converse, set union, and set intersection.

2.1.2 Interval Reasoning

With the interval reasoning problem, the objects are the time-intervals (Allen) rather than the time points. There are thirteen basic relations between a pair of intervals, B={before (b), meet (m), overlap (o), start (s), during (d), finish (f), equal (eq), finish-inverse (~f), during-inverse (~d), start-inverse (~s), overlap-inverse (~o), meet-inverse (~m), before-inverse (~b)}, as opposed to three basic

relations in the case of time-points. Every basic relation has a degree of freedom associated to it given by the freedom of end points of an interval for a given relation. Each of these basic relations and its associate dimension is defined in the tables below:

Among the 13 atomic relations;

- Six are of dimension 2:

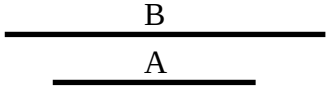
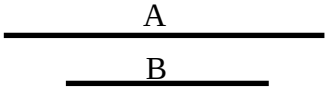
Binary operator	Relation type	Example	Representation
b	before	A(b)B	
~b	before inverse	A(~b)B	
o	overlap	A(o)B	
~o	overlap inverse	A(~o)B	
d	during	A(d)B	
~d	during inverse	A(~d)B	

Figure 2.3: Basic relation of higher-dimension Interval Relations

- Six are of dimension 1:

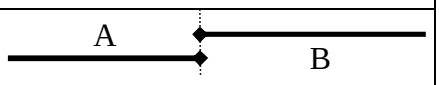
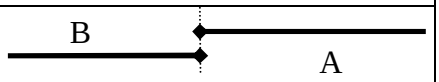
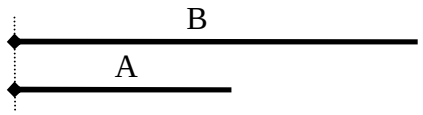
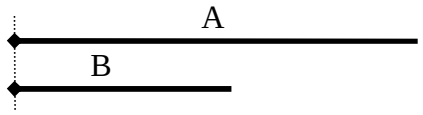
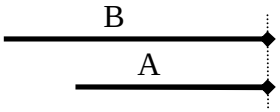
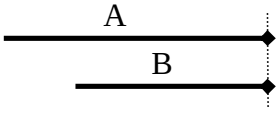
Binary operator	Relation type	Example	Representation
m	meet	A(m)B	
~m	meet inverse	A(~m)B	
s	start	A(s)B	
~s	start inverse	A(~s)B	
f	finish	A(f)B	
~f	finish inverse	A(~f)B	

Figure 2.4: Basic relation of one-dimension Interval Relations

- One is of dimension 0:

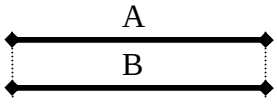
Binary operator	Relation type	Example	Representation
eq	equal	B(eq)A	

Figure 2.5: Basic relation of zero-dimension Interval Relations

A relation between two intervals is expressed as the disjunction between basic interval relations.

Example: A {b *or* d *or* ~d} B

Meaning that interval A must be placed before, *or* during, *or* after interval B. We will refer to such a set of constraints as A {b, d, ~d} B for the sake of simplified notations and clarity.

The composition table of the interval reasoning is not as trivial as for the point based reasoning. In this table, the rows indicate a relation between an interval A and an interval B. The columns indicate the relation between the interval B and an interval C. Finally, each entry represents the composed relation between interval A and interval C. The composition table for interval reasoning is given below:

	b	**m**	**o**	**s**	**d**	**f**	**eq**	**~f**	**~d**	**~s**	**~o**	**~m**	**~b**
b	b	b	b	b	Γ	Γ	b	b	b	b	Γ	Γ	U
m	b	b	b	m	Θ	Θ	m	b	b	m	Θ	F	~Γ
o	b	b	Ψ	o	Θ	Θ	o	Ψ	Ω	Σ	Φ	~Θ	~Γ
s	b	b	Ψ	s	d	d	s	Ψ	Ω	S	~Σ	~m	~b
d	b	b	Γ	d	d	d	d	Γ	U	~Ω	~Ω	~b	~b
f	b	m	Θ	d	d	f	f	F	~Γ	~Ψ	~Ψ	~b	~b
eq	b	m	o	s	d	f	=	~f	~d	~s	~o	~m	~b
~f	b	m	o	o	Θ	F	~f	~f	~d	~d	~Θ	~Θ	~Γ
~d	Ω	Σ	Σ	Σ	U	~Θ	~d	~d	~d	~d	~Θ	~Θ	~Γ
~s	Ω	Σ	Σ	S	~Σ	~o	~s	~d	~d	~s	~o	~m	~b
~o	Ω	Σ	Φ	~Σ	~Σ	~o	~o	~Θ	~Γ	~Ψ	~Ψ	~b	~b
~m	Ω	S	~Σ	~Σ	~Σ	~m	~m	~m	~b	~b	~b	~b	~b
~b	U	~Ω	~Ω	~Ω	~Ω	~b	~b	~b	~b	~b	~b	~b	~b

Figure 2.6: Composition table of Allen's Interval Algebra

With the following set of abbreviations:

- U = {b, m, o, s, d, f, eq, ~f, ~d, ~s, ~o, ~m, ~b}
- Φ = { ~o, ~f, ~d, ~s, eq, s, d, f, o}
- Γ= {b, m, o, s, d}
- ~Γ= {~b, ~m, ~o, ~s, ~d}
- Ω = {b, m, o, ~f, ~d}
- ~Ω = {~b, ~m, ~o, f, d}
- Ψ = {b, m, o}
- ~Ψ = {~b, ~m, ~o}
- Θ = {o, s, d}
- ~Θ = {~o, ~s, ~d}
- Σ = {o, ~d, ~f}
- ~Σ = {~o, d, f}
- F = {f, eq, ~f}
- S = {s, eq, ~s}

As it is the case for the point based reasoning, Allen's interval reasoning is closed under composition, converse, set union and set intersection and hence forms the Interval Algebra (IA). Our work uses the framework of Incremental Temporal Reasoning (Gerevini) also known as *Online Problem* that is applicable to some real life situations, where information is gradually added to a database. We will refer to the online qualitative temporal reasoning on point as OLQTR-P and on intervals as

OLQTR-I. In the next section we introduce the problem of online interactive reasoning scheme with qualitative constraints between temporal events.

2.2 – Online Qualitative Temporal Reasoning (OLQTR)

In our incremental framework a new temporal event is inserted into a set of events already committed on a time-line, i.e., the new temporal event is inserted within a sequence of old events, satisfying the binary constraints between the new event and the old ones. The problem could be viewed as a database entry, where a consistency checking needs to be first performed before committing the entry operation.

Incremental temporal reasoning is often really tied to real life problems. When an existing set of consistent constraints has to be redefined because of new or unexpected set of constraint, it is in most of the cases reasonable to treat them one by one until the whole set of new constraints is exhausted. This section will introduce the reader with the incremental problem in constraint reasoning.

Definition 1: Online Qualitative Temporal Reasoning (OLQTR).

Given:

- a total order $T = \{t_1, t_2, \ldots, t_m\}$ of temporal events located on a timeline,
- a new object n whose location is yet to be determined
- a set C of binary temporal constraints between n and some elements t_i of T, with $C = \{(n\ r_i\ t_i)$, where i is in [1, m]$\}$

Incremental temporal reasoning (ITR) problem answers whether n can be located on the time line satisfying the constraints in C or not. Moreover, for a satisfying problem it assigns n on the time line appropriately.

When an ITR problem is inconsistent there could be multiple subsets of C that has mutually conflicting constraints in each of them. We are interested in finding a minimal cardinality subset amongst these subsets. We presume that no weight or priority values are assigned apriori to any constraint in C.

An algorithm for solving the OLQTR-P problem is described in section 3.2.2. The next section introduces an algorithm that decides consistency for the Incremental point based reasoning. This algorithm is the core of our approach in defining the "causes" for inconsistency and is enhanced in the subsequent sections to form another set of algorithms solving our problem instance.

2.3 – Incremental Consistency issues

A system of constraint between temporal events is either consistent or inconsistent. In case of inconsistency, some constraints are not satisfied and conflict with each other. Algorithms solving temporal constraint reasoning traditionally do not inform the user about the causes of inconsistency. One of the reasons behind this is the ambiguity in identifying such a cause. For example, given a set of comparable objects a, b, and c, the information a<b, b<c, and c<a, is inconsistent (Figure 2.7a). There is no preferred constraint here that could be

identified as the cause for the inconsistency, and each one of them is equally responsible. However, if we have {a<b, b<c, b<d, d<e, e<a, c<a}, then the constraint a<b becomes a clear choice as the culprit, presuming that all constraints have equal priority (figure 2.7b).

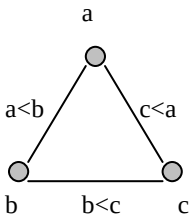


Figure 2.7a: An inconsistent temporal network

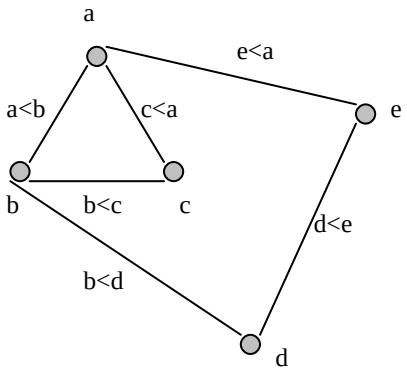


Figure 2.7b: An inconsistent temporal network

In this work, we have investigated the issues related to detecting the "reason" behind inconsistency. We have first used a simple domain of point-based temporal reasoning that is tractable and is well understood. Then, we apply this last to the wider area of incremental interval reasoning. The objective is to detect a minimal set of constraints that should be eliminated or modified for restoring consistency of an otherwise inconsistent problem instance. This work on identifying the responsible set of constraints causing inconsistency in a problem instance will be useful in the diagnosis area for quite obvious reasons. Adding such a capability would also improve the user-friendliness of the CSP-solver systems and thus, enhance their usage.

Assertion 1: Constraints in inconsistent point-based ITR conflict pairwise.

The following is the definition of the culprit detection (CD) problem.

Definition 2: Culprit Detection (CD)

Given an inconsistent ITR problem, culprit detection recalls constraints conflicting pairwise.

Definition 3: MinConflict

Given an inconsistent ITR problem, a MinConflict set is a minimal subset of the original constraint network removal of which will make the problem consistent.

For example, S= $\{s_1, s_2, s_3\}$ with $s_1<s_2<s_3$. C= $\{(n < s_1), (n >s_2), (n > s_3)\}$. This is inconsistent and the first constraint is in conflict with the other two. Hence the MinConflict= $\{(n < s_1)\}$. We define a related concept of degree of conflict of a constraint.

Mitra et al (1999) have obtained some interesting results in the point-based incremental reasoning problem, or point-sequencing problem. They have observed that a satisfiable region for a new point within a sequence of points or a contiguous region possibly excluding some old points within the interval (a preconvex interval as per Ligozat, 1996). They have utilized this property to devise an efficient

algorithm for preprocessing before attempting to find the actual valid regions for the new point that satisfies the binary constraints.

In this work we have attacked the same problem but with a different objective of finding the "cause" behind the inconsistency when the latter is detected. The following is an outline of (Mitra et al) et al's algorithm, adopted for the purpose of first detecting the inconsistency in an incremental problem.

Algorithm 1D (Mitra et al, 1999): Scan the sequence of existing points from left to right on the time-line and their relationship/constraints with respect to the "new" node that is to be inserted (within the sequence, satisfying those constraints). Keep a status variable that keeps track of whether the left end (of the list of valid regions for the new point, or as it is called, the "box") has been found, or an equality relation (new = x_i) has been found, or the right boundary has been found. The "box" is found when the constraint from the new point to the current point x_i changes from >, or ≥ to < or ≤, for i running over all the old points. The *inequality* (<>) and the *tautology* (< = >) are ignored in this scan. A singleton *equality* relation is a hard constraint, making the box converge to that old point. After a "box" is found, if any constraint demands the new point to be outside the "box," then an inconsistency would be detected, otherwise with a second scan over the "box" the algorithm would elicit the exact set of valid regions checking if the old points within the box themselves are valid regions or not. For example, a set of valid

region may be {(x_5, x_5), (x_5, x_6), (x_6, x_7), (x_7, x_7), (x_7, x_8)}, indicating that the new point may be assigned anywhere on the box [x_5, x_8) except on points x_6 and x_8.

On detection of inconsistency we run a second algorithm to find the conflict set between the constraints. This algorithm will be described in detail in section 3.2.2.

2.4 – Problem Complexity

2.4.1 Point reasoning complexity

The point reasoning is known to be solvable in polynomial time (Van Beek). However, the algorithm presented in section 3.2.2 not only decides whether the problem is consistent or not, it also suggests a minimal set of culprit relations to remove in order to 'roll back' to a consistent problem. Nevertheless, this enhancement doesn't affect the tractability of the point based reasoning problem. This will be further discussed in section 3.2.1.

2.4.2 Interval reasoning complexity

We will call ITR-I the incremental interval temporal reasoning. There exist tractable subsets of the full disjunctive set (2^{13} in number) of interval relations. Eighteen of these subsets have been identified as the only maximum tractable subsets of the full algebra (MTS). The most studied of these tractable subsets is the ORD-Horn subset (Nebel and Burckert). The problem resolution of the ORD-Horn algebra is the subject of chapter 4. Chapter 5 discusses the 17 other maximal tractable algebras.

Chapter 3 – Problem Establishment

3.1 – Disjunctive Normal Form

In an incremental temporal problem, each constraint between two relations is expressed in a disjunctive normal form. Thus, we can express the constraint "event E_1 is before *or* equal to event E_2" in a temporal point based reasoning as E_1 less-than-or-equal E_2; E_1 ($\leq$) E_2. In this simple problem, satisfying either one of the constraints is sufficient to find a valid assignment for the problem.

3.2 – Conjunctive Normal Form

Finding an assignment for a temporal event, satisfying a set of constraint is the same as satisfying at least one relation in every set of relation defining the new event. Such a problem can be expressed as a conjunction of disjunctive constraints. Consider the following problem:

- Events A, B, and C are committed to a time line
- A new event D is being added to the time line
- A set of relations is defined between the point D and each of the already existing points: D (R_a) A, D (R_b) B, D (R_c) C, where R_a, R_b, R_c are some disjunctive temporal constraints between D and the existing events.

In order to find a valid assignment for D, we must satisfy at least a relation in each of the R_a, R_b, R_c. Thus, the consistency of a problem P is given by: P = (D (R_a) A) $\wedge$ (D (R_b) B) $\wedge$ (D (R_d) C)

Example: let P be a problem on a point-based constrained network containing 2 ordered points a and b. We want to add a third point c to the network respecting the following constraints (Figure 3.1):

- c (>=) a ∧ c (<=>) b

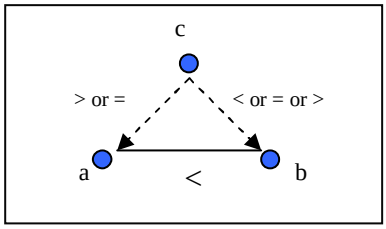


Figure 3.1: Conjunction of constraints

In other words, a valid assignment for c is a region greater-than *or* equal to a **and** less-than *or* equal *or* greater-than b. The satisfaction of an assignment on both sides of the **and** operator implies that the system is consistent and point c can be assigned to the time line.

The next section describes the algorithms we developed to solve the point incremental temporal problem and detects minimal culprits set when inconsistency occurs. We conclude this chapter by a discussion on detecting inconsistencies in the far more complex interval problem.

3.3 – Culprit detection in the point algebra

Many sets of constraints together could cause inconsistency, such that removal (or fixing) the constraints in this set would make the system consistent.

We call this problem the “consistency restoration” problem and such a set of constraints as the “responsible set.” It is quite inconceivable that all the provided constraints need to be removed/fixed to solve the “consistency restoration” problem. Actually this assertion could be easily checked with an example presented later in this section. A related question here is then which particular set of conflicting relations we chose as a solution to the problem, and subsequently report to the user. Our proposal is that we choose a *responsible set* that is of minimal cardinality out of all possible *responsible sets*. We call such a minimal cardinality-responsible set as the “minimal set” or *MinSet*. Of course, there could be more than one such minimal set with same cardinality values, but we would like to find any one of them.

Definition 4: The *degree of conflict* of a given constraint in a CSP is the number of other constraints, that, combined with this constraint, render the system inconsistent.

If a system contains *n* constraints, then the degree of conflict for any constraint will be at most *n-1*, since a relation cannot conflict with itself, and at least 0 which indicates that the relation does not conflict with any other relation .

Example: Let S be a set of constraints S = {c_1, c_2, c_3, c_4, c_5, c_6}. Note that the elements of S are not temporal objects, but binary constraints between the temporal objects.

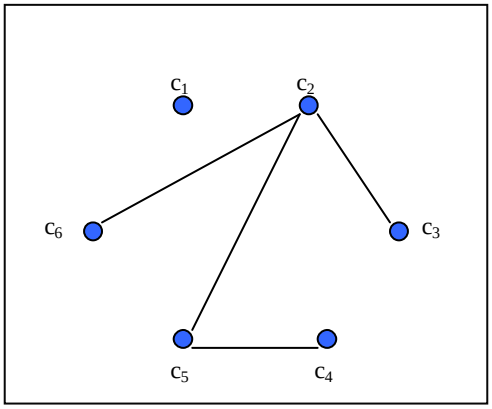


Figure 3.2: Binary constraints on temporal objects

The edges between binary constraints represent a conflict between the constraints. Here, the degree of conflict for c1 DC(c_1) is zero, since c1 does not conflict with any other constraint. The degree of conflict for c_2 is three, for c_3, c_4 and c_6 each it is one, and for c_5 it is two. The degree of conflict of S can be summarized as:

- DC(c_1) = 0
- DC(c_2) = 3
- DC(c_3) = 1
- DC(c_4) = 1
- DC(c_5) = 2
- DC(c_6) = 1

Removing relation c_2 and one of the relations between c_5 and c_4 would be enough to make the system consistent. Hence, $\{c_2, c_4\}$ and $\{c_2, c_5\}$ are both *MinSets* here, whereas $\{c_6, c_5, c_3\}$ is another *responsible set* that is not a *MinSet*.

We can now introduce the FindMinSet algorithm which purpose is to solve such problems and propose a minimal conflicting set of constraints to restore consistency.

3.2.1 – The FindMinSet Algorithm

The following algorithm is a preprocessing of the FindMinSet algorithm. It finds the conflict set for constraints and the corresponding degree of conflict for each constraint.

```
Input: A set of constraints between the new point and the set of committed intervals
Output: a set of conflicting constraints if the system is inconsistent, 'consistent'otherwise
Algorithm GenerateConflictSet
1 ConflictSet = null;
2 FOR i =1 to N do DegreeOfConflict[i] = 0;
3 FOR each constraint ci from i =1 to N do
4       FOR each constraint cj from j = i+1 to N DO
5           IF (ci is "<", or "≤", or "=") and (cj is ">", or "≥", or "=") THEN
6               ConflictSet = ConflictSet U {(ci, cj)};
7               DegreeOfConflict[i]++;
8               DegreeOfConflict[j]++;
9           end IF;
10          RETURN ConflictSet if ConflictSet ≠ null, consistent otherwise;
11      end FOR;
12 end FOR;
End Algorithm.
```

First, the conflict set and the degree of conflict of every relation are initialized (lines 1 and 2). Next, for each pair of ordered constraints (lines 3 and 4), if two relations conflict with each other, the conflicting pair of constraint is stored in the conflict set ConflicSet (line 6) and the respective degree of conflict of each relation is incremented (lines 7 and 8). When the outer FOR loop terminates, the algorithm returns the conflict set if different from null, consistent or 'consistent' if the conflict set is null (line 10). This is obviously an $O(N^2)$ algorithm. The problem of finding a *minimal set* of constraints removal/fixing of which would restore

consistency in a system (constraint network) is solved by the following greedy algorithm.

```
Input: A set of conflicting constraints with their respective degrees of conflict
Output: A minimal set of constraints to remove to restore consistency
Algorithm FindMinset
1 Minset = empty;                    // set of minimal nodes to be removed
2 AggregateDegreeOfConflict = Σ |ConflictSet [DegreeOfConflict]|;
3 WHILE AggregateDegreeOfConflict ≠ 0 DO                          // O(N)
4       Let c = a constraint with the maximum DegreeofConflict;   //O(N log N)
5       Minset = Minset U {c};
6       FOR each element (c, ci) in ConflictSet DO                // O(N)
7               ConflictSet = ConflictSet  -  (c,ci);
8               DegreeOfConflict[c] = DegreeOfConflict[c] -1;
9               DegreeOfConflict[ci] = DegreeOfConflict[ci] -1;
10              AggregateDegreeOfConflict = AggregateDegreeOfConflict -2;
11      end FOR;
12 end WHILE;
13 RETURN Minset;
End Algorithm.
```

This is a greedy algorithm that will populate the Minset set of relation with a minimal number of constraints responsible for inconsistency. Line 1 initializes Minset to null. AggregateDegreeOfConflict holds the summation of every degree of conflict within ConflictSet. While AggregateDegreeOfConflict is greater than 0, the system remains inconsistent, and some more relations must be removed from the original set (line 3). Line 4 picks a relation with highest degree of conflict as a 'best' candidate for elimination. Lines 8 through 10 update the set of variables,

omitting the relation c with highest degree of conflict. As soon as AggregateDegreeOfConflict reaches 0, *FindMinset* returns the Minset set containing the minimal set to remove (line 13).

The preprocessing '*GenerateConflictSet*' algorithm runs in $O(N^2)$. As shown with comments in the '*FindMinset*' algorithm, $O(N^2 \log N)$ is the asymptotic time complexity. Hence the overall complexity of the problem solver for point algebra is: $O(N^2) + O(N^2 \log N) = O(N^2 \log N)$

3.2.2 – A Tractable problem

The problem has a flavor of the well known "vertex cover" problem, and hence a polynomial algorithm is unlikely to be a complete algorithm in general (subject to $P \neq NP$). However, in the point-sequencing problem it could be easily shown that a graph generated over the conflict set (with nodes being the constraints and edges being the conflicting constraints) is a bipartite graph (see the set of properties below). The vertex cover problem is tractable over bipartite graph and the algorithm *FindMinset* is a complete algorithm in this situation.

We will now establish a set of theorems and properties that apply to our problem. These properties will be used to prove the correctness and minimality of the *FindMinset* algorithm.

Property 1: Considering a set $S = \{R_1, .., R_n\}$ where R_i is a relation making a system inconsistent ($1 \leq i \leq n$), and $m_i(S)$ is the degree of inconsistency of R_i in S. Then the number of conflicting pairs is $N = \sum_i m_i(S)/2$.

($\sum_i m_i(S)$ will always be an even number, since inconsistent relations exists pairwise)

Proof: Every time a relation appears in a pair of conflicting relation, both relations have their inconsistency degree increased by one, hence the sum of all inconsistency degree is twice the number of conflicting pairs.

Property 2: Let I be the set in which *FindMinset* adds relations to be removed form the original set to roll back to consistency. Let $m_i(I)$ be the inconsistency degree of a relation R_i contained in S. Finally, let S be the initial number of conflicting pairs in the original inconsistent set of relations S. Then, if $\sum m_i(I) \geq N$, the system is consistent.

Note that $\sum_i m_i(I) = N$ is a sufficient condition for the system to be consistent. As a matter of fact, in our algorithm, $\sum_i m_i(I)$ will never be greater than N, since the goal of our investigation is to find a minimal set of conflicting relations and hence to stop removing relations as soon as the system is consistent.

Proof: Removing a relation R_i will also decrement $DC(R_j)$ by one when R_i conflicted with R_j. So, the total inconsistency degree of S will be lowered by $2*m_i$ .Recall that when $\sum_i m_i(S)$ reaches 0 in line 3 of algorithm *FindMinset*, the system is consistent. Hence, if $2* \sum_i m_i(I) = \sum_i m_i(S)$, the system is consistent; $\sum m_i(I) = \sum m_i(S)/2 = N$

Lemma 1: From properties 1 and 2, we can conclude that if $\sum_i m_i(I) \geq \sum_i m_i(S)/2$, the system is consistent. Inversely, if $\sum_i m_i(I) < \sum_i m_i(S)/2$, then the system is not consistent.

Property 3: If no '=' relation is involved in a conflicting set, then a set of point-based conflicting constraints S is a bipartite graph with two partitions S_1 and S_2, S_1 being the "less-than partition", and S_2 being the "greater-than partition".

Proof: Whenever an element is added to S, it is conflicting with another element in S. let us call these two elements V_1 and V_2. Then, either:

V_1 (> *or* ≥) V_2 *or* V_1 (< *or* ≤) V_2. According to their relation with each other, V_1 and V_2 will be stored in their respective bipartite subgraph.

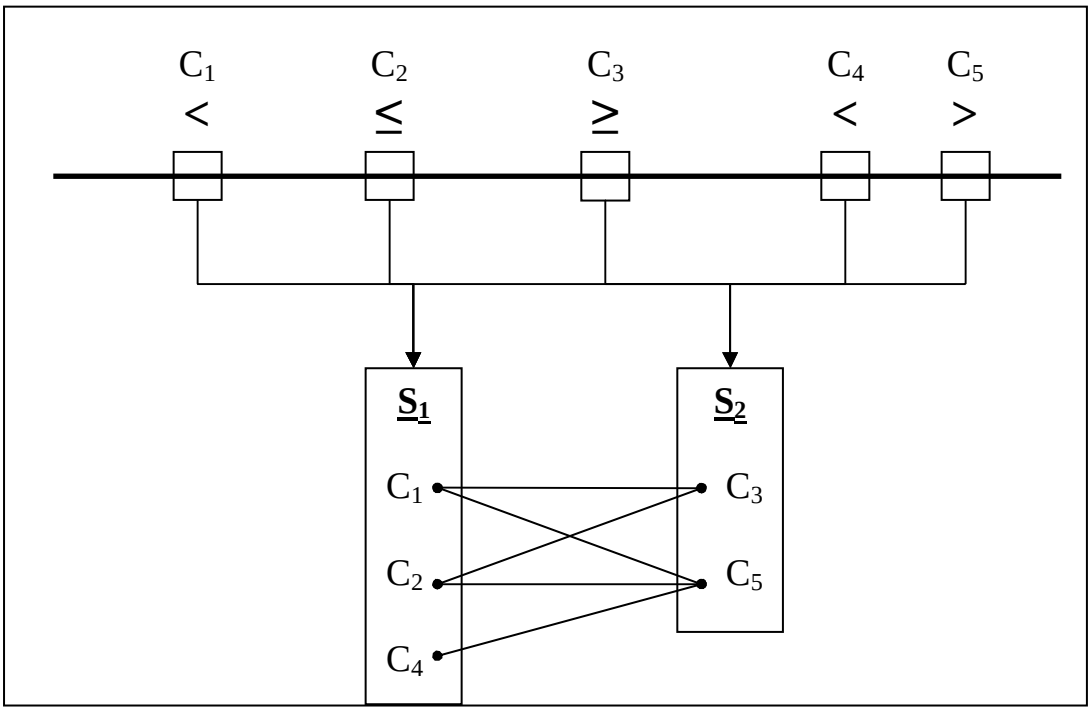


Figure 3.3: Bipartite graph on point-based reasoning

Property 4: If either $|S_1| = 0$ or $|S_2| = 0$, then both $|S_1| = 0$ and $|S_2| = 0$, therefore the system is consistent

Similarly, $|S_1|$ or $|S_2| = 0$ is a sufficient condition for $|S| = 0$.

Proof: If S_1 or S_2 is empty, then the graph S is totally disconnected and no conflicting constraint remains, hence $|S| = 0$ and the system is consistent.

Property 5: The element with the highest number of edges of each bipartite subgraph is connected with all elements of the other subgraph.

Proof: Consider a time line with events assigned on it. We want to assign a new point on the time line with inconsistent constraints. The element with highest inconsistency degree in each sub graph are the conflicting elements on the far most left side (e_1) for any '<', '≤', or '=' constraint and inversely on the far most right side (e_2) for any constraint of type '>', '≥', or '=' as illustrated in figure 3.4.

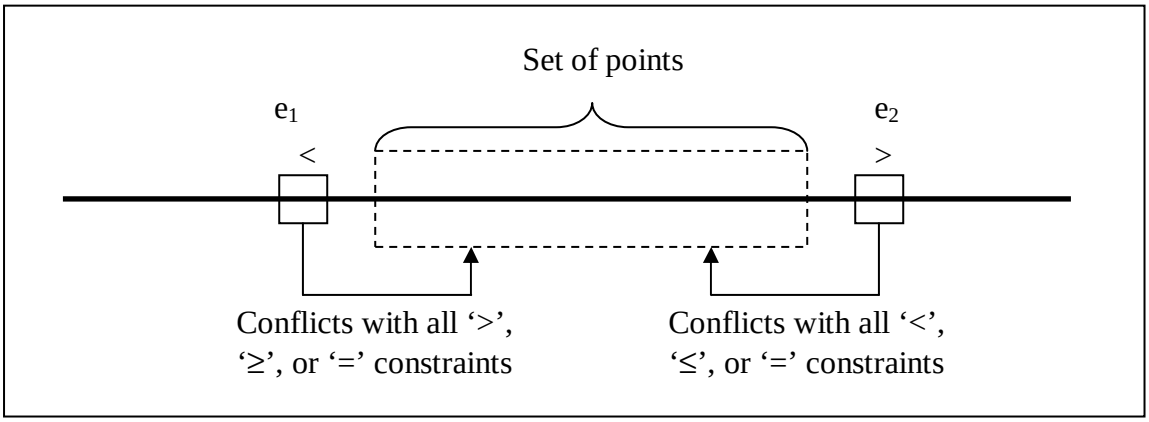


Figure 3.4: Conflict set on point-based reasoning

Hence, $e_1 \in$ partition 1 and conflicts with every '<', '≤', or '=' constraint from "Set of points" and $e_2 \in$ partition 2 and conflicts with every '>', '≥', or '=' constraints from "Set of points". Since no element is connected within a partition and elements (e_1) and (e_2) are connected with every element of their opposite partition, (e_1) and (e_2) have the maximum number of adjacent edges within their respective partitions. This completes the proof.

For the following theorem, we must define the procedure Edges(E) as being the set of adjacent edges of element E,

Theorem 1: Let two elements E_1 and E_2 belong to the same partition. Then, if $|Edges(E_2)| \leq |Edges(E_1)|$ then the set of adjacent edges of E_2 is a subset of the set of adjacent edges of E_1. In other words:

Let $E_1, E_2 \in S_1$. If $|Edges(E_2)| \leq |Edges(E_1)|$ then $Edges(E_2) \subseteq Edges(E_1)$.

Proof: From property 5, we know that the element E_1 with highest number of adjacent edges within a partition, say S_1 is connected to every element from the other partition. Hence, the set of adjacent edges of every other element in S_1 is a subset of the set of adjacent edges of E_1. Consider the *removing procedure* below:

Removing the element E_1 from S_1 will leave the partition with a new element with highest number of adjacent edges E_2. Recursively, every set of adjacent edges of every other element in S_1 is a subset of the set of adjacent edges of E_2. This

procedure can be applied until there remains no element in the set, proving Theorem 1.

Property 6: Each time a relation is chosen by Algorithm *FindMinset* on line 4, it has a number of adjacent edges of exactly: Max ($|S_1|$,$| S_2|$)

Proof: The element with the highest number of adjacent edges is the element in the smaller sub graph, and from property 5, is connected with all elements of the bigger sub graph.

Property 7: The equal relation can be added to the partition of the bipartite set by dividing it into two sub relations: less-than-or equal and greater-than-or-equal, each of which appear in its respective sub graph and do not change the nature of the bipartite graph. As a result:

- $\{n\ (=)\ p\} \equiv \{n\ (\geq)\ p\} \cap \{\ n\ (\leq)\ p\ \}$

This property is illustrated in figure 3.5.

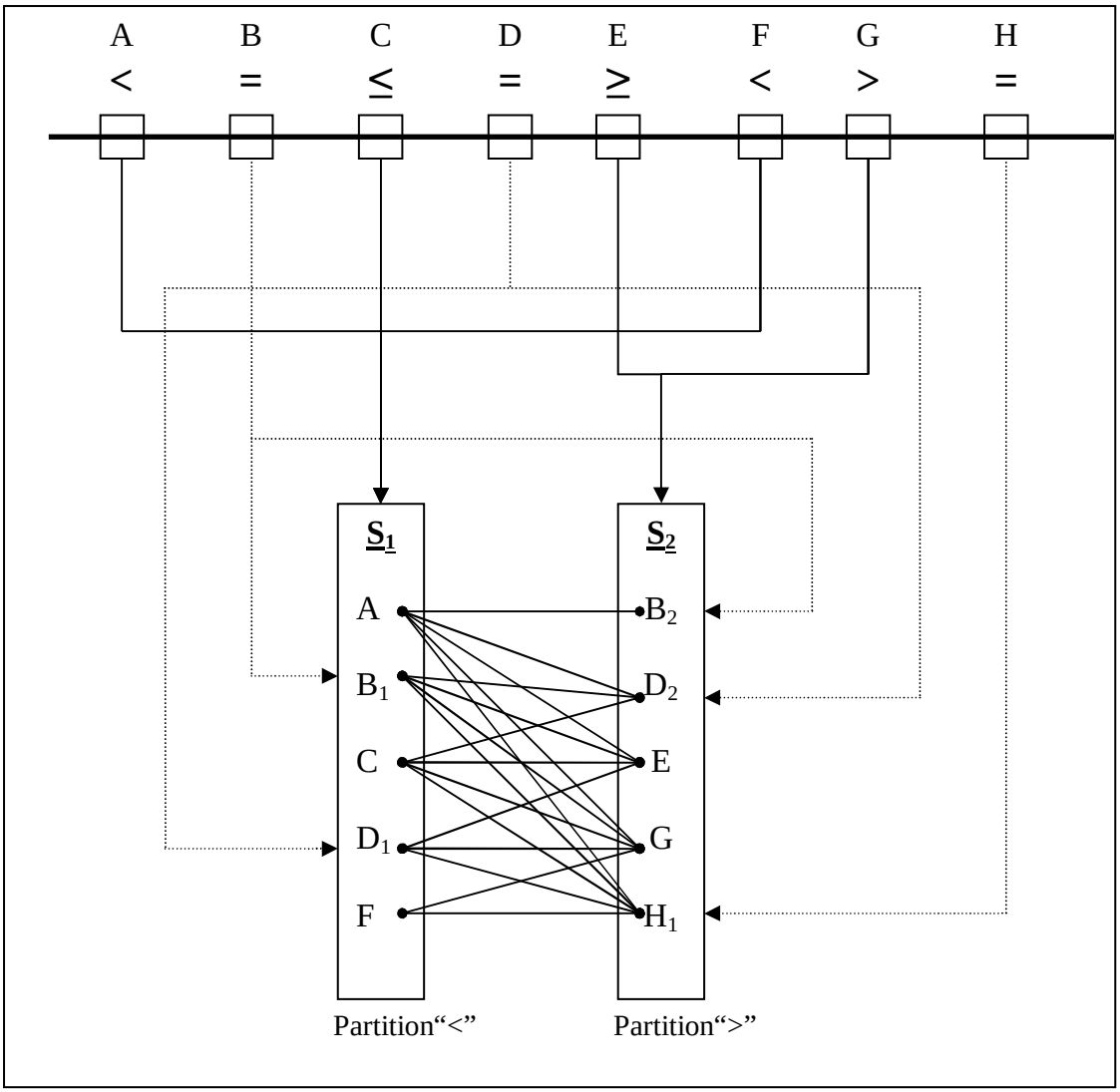


Figure 3.5: Bipartite constraints assignment

Proof: The ≤ part of an equality will not conflict with any relation in the '<' (S_1) partition, and will conflict with every relation from the partition '>' the = relation originally conflicted with. The proof for the partition (S_2) is equivalent.

For S_1: The "<"partition includes the following relations: {<, ≤, and the ≤ part of the equality}.

For S_2: The ">"partition includes the following relations: {>, ≥, and the ≥ part of the equality}.

Property 8: The maximum sufficient number of elements to remove to make the system consistent is ≤ Min ($|S_1|$, $|S_2|$)

Proof: At most, removing the entire smallest set will remove inconsistency (bipartite graph).

Lemma 8: Algorithm *FindMinset* finishes with a set $|I| \leq$ Min ($|S_1|$, $|S_2|$)

Proof: From property 6, we know that each time algorithm *FindMinset* chooses a relation, it is the one conflicting with Smax = argMax($|S_1|$, $|S_2|$). Then Algorithm *FindMinset* chooses a relation in the smallest partition. After each step, exactly one relation is removed from the current Smin set, and some relation(s) may be removed from Smax. Then, either Smax becomes less than Smin, in which case the next element to be picked up will be in Smax, or Smin still the minimal set, and the next element will be picked up in it. Eventually, Smin remains the minimal set during the entire procedure, and will be totally removed by successive choices of algorithm *FindMinset*.

Property 9: Two different sets I_{1a} and I_{1b} found with algorithm *FindMinset* have the same minimal cardinality for a given input:

Proof: The goal of the *FindMinset* algorithm is to obtain a minimal set of inconsistent relations to remove from the original inconsistent set of relations by removing (Max (S_1, S_2)) edges for the bipartite graph at each stage until the number of connecting edges reaches 0. Two sets found with algorithm *FindMinset* will not necessarily have the same set of elements. If at any stage, a tie occurs between two elements in the smallest partition, or if $|S_1| = |S_2|$, then the algorithm has the choice among several elements to remove.

Let us consider these two cases separately.

- First, if two elements are tied in the smallest partition: R_1 and R_2 (R_1, $R_2 \in S_i$), then both elements will eventually be picked up by the algorithm in arbitrary order.
- Second, if at any stage $|S_1| = |S_2|$, with R_1 being the element with maximal adjacent edges in S_1, and R_2 being the element with maximal adjacent edges in S_2. Removing R_1 will possibly make $|S_1| < |S_2|$, then at the next stage, R_2 will be picked up. Otherwise, it could make $|S_1| > |S_2|$, meaning that removing R_1 also removed at least 2 elements in S_2. Then the next element to be picked up will be in S_2.

Theorem 2: Algorithm *FindMinSet* finds a minimal set to remove from an inconsistent set of point-based relation to make the system consistent

Proof: The proof of theorem 2 is divided in two parts. The first part proves that when the algorithm terminates, the remaining set of relations is consistent. The second part proves that the set removed from the original set is minimal.

- *Consistent:*

When the algorithm finishes, n = 0 and S is empty. When S is empty, every inconsistent relation have been removed, consequently, the system becomes consistent.

- *Minimal:*

Let I_1 be a set of inconsistent relation found with *FindMinset*. Next, assume that there exists a minimal set of inconsistent relations to remove to regain consistency I_2 found with another algorithm than *FindMinset* which contains fewer elements than I_1. The set of constraints to remove from an inconsistent system found by Algorithm *FindMinset* is not unique. In fact, algorithm *FindMinset* proposes one of these sets if it exists. The choice of this set depends on the ordering of the variables as *FindMinset* picks the first occurrence of maximal conflict based cardinality within the set of all inconsistent relations. Hence, according to the constraint ordering, algorithm *FindMinset* may find different sets with same cardinality (minimal). Let us call the set of all different sets possibly found by algorithm *FindMinset*: GI = $\{I_{1a}, I_{1b}, I_{1c}, ...\}$

No proper subset of any set found by algorithm *FindMinset* can make the system consistent; hence, I_2 is not a proper subset of I_{1i} for any i ($1 \leq i \leq |GI|$).

- S_1 and S_2 are the sets held by the two subset of the bipartite set before the algorithms run (original bipartite sets).

- s_1, and s_2 are the sets held by the two subsets of the bipartite set while the algorithms are running (dynamic bipartite sets).

Contradiction

At one time, $|s_1| \neq |s_2|$ and *FindMinset* will pick the element with highest inconsistency degree, but I_2 will not in order to have a different set from I_1. Since the element picked by *FindMinset* is connected with all elements of the other set of the bipartite graph, the other algorithm MUST pick all elements of this last set MAX(s_1, s_2) (i.e., the set with the highest number of elements). But from property 8, removing MIN(s_1, s_2) is enough to have the bipartite graph totally disconnected. Therefore, the previous assumption is false, leading the proof to a contradiction.

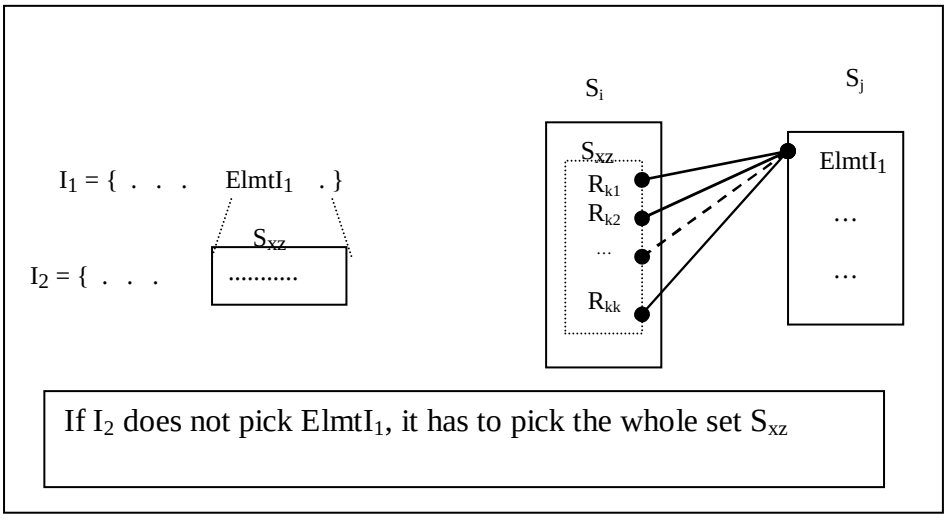


Figure 3.6: Relation conflicting with a maximum cardinality partition

3.2.3 – Implementation

The algorithms mentioned in the previous section have been implemented with a graphical user interface to the software. It displays the valid regions (if consistent) on the time-line with the existing point-sequence, and allows the user to commit the new point on one of the valid regions interactively. Then it goes to the next iteration for accepting the next new point. In case of inconsistency, the *GenerateConflictSet* algorithm provides a set of conflicting constraints to the *FindMinset* algorithm and dumps the *Minset* for possible corrective action by the user.

A graphical interface allows the user to add points incrementally, and at each new point, the system reacts giving the user the information about consistency of the system and the minimum conflict set if applicable.

The interface shows the time line basis and a set of options allowing the user to Add a new point, load a pre-computed example, or reset the current database.

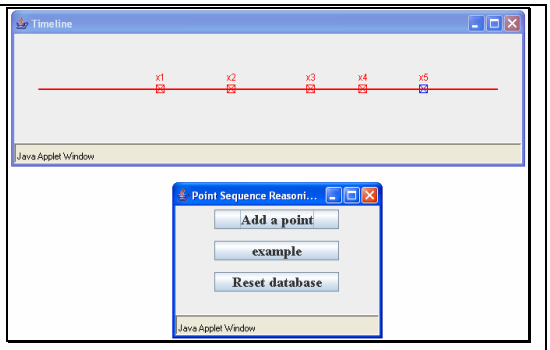
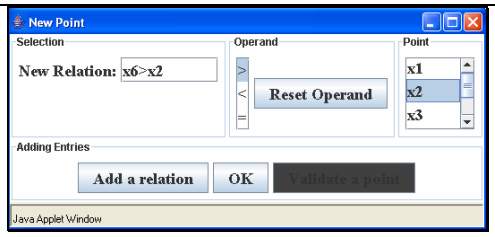
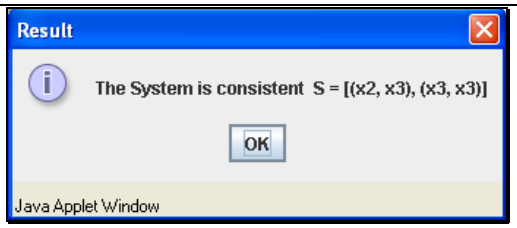
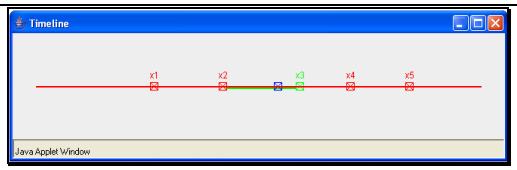
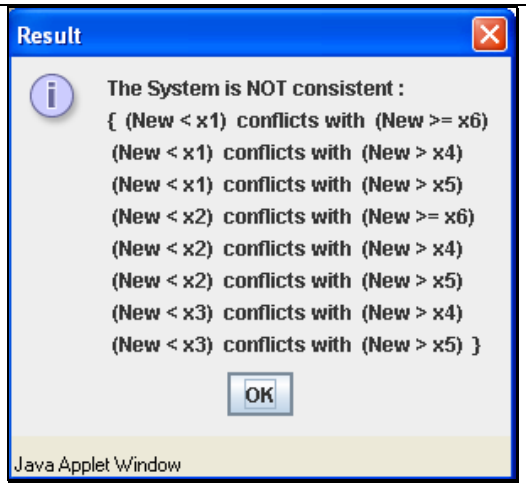
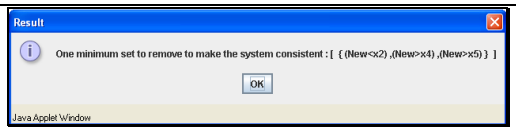


Figure 3.7a



When adding a new point, the user has the choice to set constraints between the old set of points and the new point to be

Figure 3.7b	added.
At each stage, if the system is consistent the application prompts the user to add the new point into a set of valid regions on the time line	 Figure 3.7c
 Figure 3.7d	The user then defines where the new point should be added. The valid regions are shown in green on the time line.
In case of inconsistency, the program display the conflict set as it is transmitted to the *FindMinSet* algorithm.	 Figure 3.7e
 Figure 3.7f	Then, the program suggests the user a minimal set of relations to remove from the previous query in order to make the system consistent.

This program has been implemented to test the correctness of the algorithms as well as to conduct some experience, giving us a better understanding of the OLQTR-P. The correctness of the algorithm is crucial as we use it to solve the OLQTR-I problem.

3.3 – Culprit detection in the Interval Problem

3.3.1 – Reasoning on the whole algebra is NP-Hard

The problem of finding consistency for interval algebra is obviously more complex than for point algebra since interval has 13 basic relations against 3 for point algebra. The extra set of relations comes from the fact that the constraints not only deals with a unique discrete point, but with a starting point and an ending point. In fact, any interval can be thought as a pair of points, where the following trivial property always holds:

Property 10: Any given interval A is defined by its two end points A^- and A^+ where $A^- < A^+$.

Hence, we will omit to specify this property for the formulas and examples presented in the rest of this thesis.

Since we can express an interval in terms of points, we can also express any relation between two intervals as a set of point interval problems. Hence, the

relation A (s) B can be represented as ($A^- = B^-$) & ($A^+ < B^+$). This is sufficient to describe the relation between both intervals and any other relation between any two points is redundant. Unfortunately, this dramatically increases the complexity of the overall Interval problem. Actually, reasoning on the whole interval algebra is known to be NP-complete. However, about 90% of the full algebra has been shown to belong to tractable subsets. Recent works have proven that there exist exactly 18 maximal tractable sub-algebras (MTS), and that only problems that do not belong to one of these subsets are untractable. The next chapter is entirely dedicated to one of these sub-algebras known as the ORD-Horn Algebra and identified as (H) here. It is an algebra with very different properties than the others. The remaining MTSs are explained in section 5.3.

Chapter 4 – ORD-Horn algebra

4.1 – Definition

Search for inconsistency and consistency restoration can be done in polynomial time on interval algebra when the set of constraints being involved belong to a tractable subclass of Allen's interval algebra. The ORD-Horn algebra is one of those, and it is also the only maximal tractable subset of Allen algebra containing all 13 basic relations making it one of the most interesting. Nebel and Burckert discovered that any relation within the ORD-Horn algebra can be expressed in a conjunctive normal form where each literal contains at most one relation of the type $\leq$ or =, and any number of relations of type $\neq$ (Nebel and Burckert).

Example: Consider an interval *Old* committed to a time line. We want to add a new interval *New* to the time line satisfying the following ORD-Horn interval relation: New{o, s, fi}Old. This relation can be expressed in terms of constraints between interval end points in following conjunctive normal form as:

$(New^- \leq New^+) \cap (New^- \neq New^+) \cap (Old^- \leq Old^+) \cap (Old^- \neq Old^+) \cap$

$(New^- \leq Old^-) \cap (New^- \leq Old^+) \cap (New^- \neq Old^+) \cap (Old^- \leq New^+) \cap$

$(New^+ \leq Old^+) \cap (New^+ \neq Old^-) \cap (New^- \neq Old^- \cup New^+ \neq Old^+)$.

With index '$^-$' standing for an interval's 'starting point' and '$^+$' standing for an interval's 'ending point'.

In Section 4.3, we introduce a systematic procedure to express any ORD-Horn relation into its equivalent point representation satisfying the ORD-Horn definition. We will call the OLQTR-I problem with ORD-horn relations only as OLQTR-I (O) problem. The following fact about the interval reasoning with tractable subsets like that with ORD-Horn relations is important in attacking the OLQTR-I problem.

Assertion 3: The path consistency (or triangle-consistency between every three intervals) in an interval-reasoning problem is necessary and sufficient for global consistency of the interval reasoning problem with tractable subsets. In the ITR-I framework all the old intervals s_i in S are committed on the time-line. Hence, for each triangle (s_i s_j n) that needs to be checked for consistency for some i and j, only two constraints could conflict with each other (n r_i s_i), and (n r_j s_j). This is a very similar situation as with the OLQTR-P problem, since the point-based reasoning is also tractable.

4.2 – Literature

As mentioned earlier, any ORD Horn relation can be expressed in a conjunctive normal form, containing only clauses, with at most one positive literal. In other words, any relation belonging to the ORD-Horn algebra can be expressed as a conjunction of literals with any number of inequalities (≠) and at most one expression of higher dimension (≤ or =).

Ligozat noted in his work that the basic interval relation could be represented as a point on a 2D Cartesian space where the X-coordinates represents the starting point of an interval, and the Y-coordinates represents its ending point. Thus, every basic relation on an interval is defined as an area on the 2D plane space. It can be noticed on figure 4.1 that 2-dimensional relations cover an area, whereas 1-dimensional relations are represented by a line segment, and the 0-dimensional relation, equality, is represented as a point.

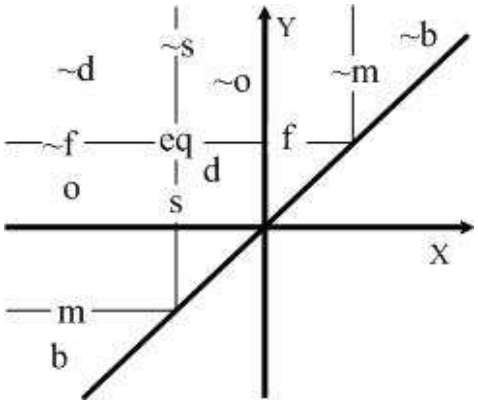


Figure 4.1: 2D plane space for interval representation

Hence, any set of relations can be expressed with respect to the interval being studied as an area on the 2D plane space. Moreover, Ligozat has discovered that the interval algebra corresponded to a lattice as shown on figure 4.2. This lattice becomes very useful when reasoning about the 'neighborhood' relation between basic interval relations. Besides, Ligozat defined a convex relation as being the set of relations included in an interval on the lattice. Thus, {m, o, s, ~f,

eq} is a convex relation since it includes every relation between the basic relations m and eq. Last, Ligozat described a pre-convex set of relations as a convex relation in which some basic lower dimensional relations might be omitted. This last definition corresponds to the set of ORD-Horn relations (Ligozat).Table 4.3 associates the basic relations with their lattice coordinates, and their point-based ORD-Horn representation.

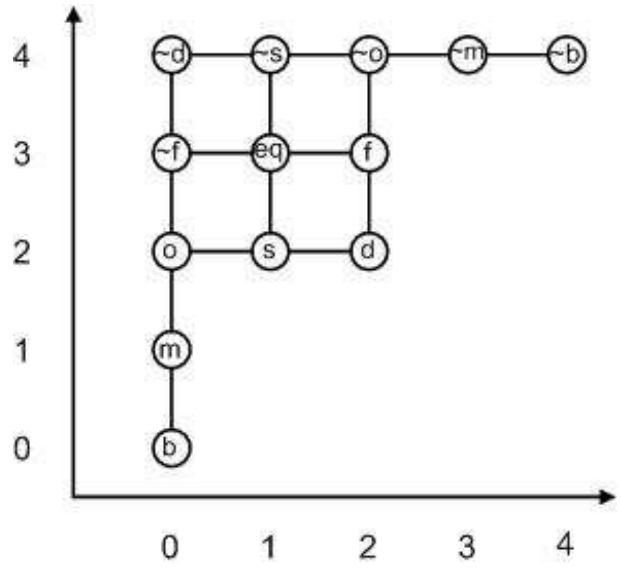


Figure 4.2: Ligozat's lattice representation

Relation	Coordinates (x,y)	ORD-Horn representation
b	(0,0)	$B^+ \leq A^- \cap A^- \neq B^+$
m	(0,1)	$A^- = B^+$
o	(0,2)	$B^+ \leq A^+ \cap A^+ \neq B^+ \cap A^- \leq B^+ \cap A^- \neq B^+ \cap A^- \leq B^+ \cap A^- \neq B^+$
s	(1,2)	$B^+ \leq A^+ \cap A^+ \neq B^+ \cap A^- = B^-$
d	(2,2)	$B^+ \leq A^+ \cap A^+ \neq B^+ \cap A^- \leq B^- \cap A^- \neq B^-$
f	(2,3)	$A^- \leq B^- \cap A^- \neq B^- \cap A^+ = B^+$
eq	(1,3)	$A^+ = B^+ \cap A^- = B^-$
~f	(0,3)	$A^+ = B^+ \cap B^- \leq A^- \cap A^- \neq B^-$
~d	(0,4)	$B^- \leq A^- \cap A^- \neq B^- \cap A^+ \leq B^+ \cap A^+ \neq B^+$
~s	(1,4)	$A^+ \leq B^+ \cap A^+ \neq B^+ \cap A^- = B^-$
~o	(2,4)	$B^- \leq A^+ \cap A^+ \neq B^- \cap A^- \leq B^- \cap A^- \neq B^- \cap A^+ \leq B^+ \cap A^+ \neq B^+$
~m	(3,4)	$A^+ = B^-$
~b	(4,4)	$A^+ \leq B^- \cap A^+ \neq B^-$

Figure 4.3: ORD-Horn representation of basic relations

As we can see on figure 4.3, some interval constraints have common point-based relations within their neighborhood as it is the case for ~s and ~o. Both of these interval relations have ($A^+ \leq B^+ \cap A^+ \neq B^+$) in common.

We combined these facts in order to create a set of models capable of expressing any ORD-Horn set of relation in its simplest ORD-Horn formula. These models are described in the next section.

4.3 – Decomposition into a point algebra problem

4.3.1 – Positive model

Any interval basic relation can be expressed in terms of its starting and ending point's relation. To do so, we use Ligozat's lattice representation of basic relations and enhanced it with point information relative to groups of basic relations on each axis. We called this model the 'positive model'. It is illustrated in Figure 4.4.

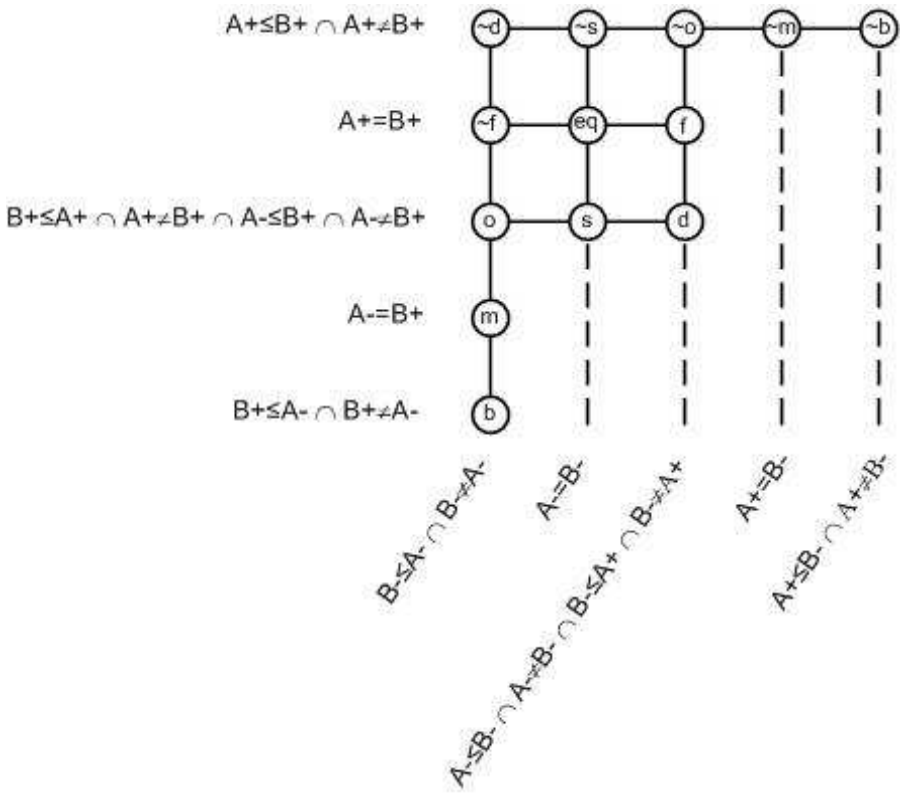


Figure 4.4: Positive model representation

This model allows us to represent a relation between two intervals in an ORD Horn form. For example, relation start's ORD-Horn representation is given by

- $B \{s\} A = (B^+ \leq A^+ \cap A^+ \neq B^+ \cap A^- \leq B^+ \cap A^- \neq B^+ \cap A^- = B^-)$

Since the constraint $A^- = B^-$ must be satisfied in order to find an assignment for the interval constraint, the equation can be simplified by removing the terms $A^- \neq B^+$ and $A^- \leq B^+$ giving:

- $B \{s\} A = (B^+ \leq A^+ \cap A^+ \neq B^+ \cap A^- = B^-)$

Moreover, this model is capable of expressing a set of relations when the involved relations are contiguous on the lattice (Figure 4.4). Thus, the ORD-Horn formulae expressing the set of relations {start, equal, start inverse} is:

- $B \{s, eq, \sim s\} A = (A^- = B^-)$

Furthermore, it is possible to state the point relations of a contiguous area of the model by associating the relations given on the axis. We will call such an area a 'convex box' of relations. Let us consider the relations {start, during, finish, equal}.

- $B \{s, d, f, eq\} A = (B^+ \leq A^+ \cap A^+ \neq B^+ \cap A^- \leq B^+ \cap A^- \neq B^+) \cap (A^+ = B^+) \cap (A^- = B^-) \cap (A^- \leq B^- \cap A^- \neq B^- \cap B^- \leq A^+ \cap B^- \neq A^+)$

After simplification, the set of interval relations can be traduced in point-based constraints as:

- $B \{s, d, f, eq\} A = (B^+ \leq A^+ \cap A^- \leq B^+ \cap A^- \neq B^+ \cap A^- \leq B^- \cap B^- \leq A^+ \cap B^- \neq A^+)$

The simplification procedure will be explained in section 4.4. We can note here that the corresponding formula respects the ORD-Horn definition. But now, we might want to remove some relations from a given convex box while keeping the ORD-Horn properties. Actually, ORD-Horn algebra permits to remove some of the lower dimensional relations within a convex box of neighbor relations. We designed a second model to serve this purpose and we call this model the ‘negative model’.

4.3.2 – Negative model

It is possible to express the negation, or nonexistence of basic relations respecting the ORD-Horn conditions using a model expressing absence of the relations. We use the '!' sign to signify that a basic interval relation does not belong to a set of interval relations. This 'negative model' is shown in figure 4.5. Any basic relation can be expressed as the disjunction of the point constraints given by the model. Note here that only absence of lower dimensional relations meet, start, finish, finish inverse, start inverse, meet inverse, equal only and the higher dimensional relations before, before inverse can be represented in an ORD-Horn way. Before and before inverse can actually be added to or removed from any preconvex relation without changing the property of the preconvex relation, as defined by Ligozat. This is simply due to the fact that the higher dimensional relations before and before inverse are located at the boundaries of the lattice. Adding the negative expression of a higher dimensional relation renders the relations non-ORD-Horn.

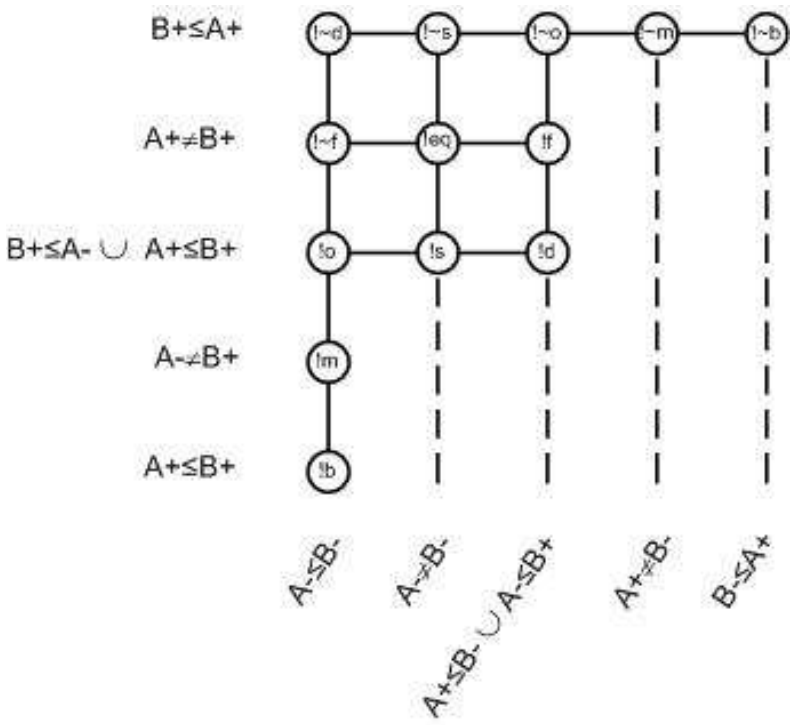


Figure 4.5: Negative model representation

With this model, we can express relations such as U-{eq} where U is the set of all basic relations, also referred to as the Universe, as shown in the example below:

- B (U-{eq}) A = ($A^- \neq B^- \cup A^+ \neq B^+$)

Now, by combining both models, it is possible to express any ORD-Horn relation from a 'box' expressed using the positive model, and removing absent lower dimensional relations with the negative model. A procedure to automate the simplification is described in section 4.4.1. This procedure first expresses the formulae of the convex relation in which every clause contains at most one positive literal (=, ≤) and any number of negative literals (≠) in a conjunctive normal form. The procedure then adds the absent relations as given in Figure 4.6, hence reducing the formulae to a pre-convex relation respecting the ORD-Horn definition.

4.4 – Algorithms

In order to transpose the incremental interval problem to a succession of point problem, we include the ORD-Horn clauses of the point to be added in the set of existing clauses. Then, we pick the unit clauses (clauses that contain only one relation) which are, by definition the tightest constraints, and some inequalities (the less constraining relations) in the remaining multiple clauses, making sure than no inequality conflicts with any possible ORD equivalence relation "=" and compute them together. The next step consists in grouping respectively the relations involving the new 'start point', and the new 'endpoint ', and apply on both sets the *FindMinSet* algorithm. If a box is found for both start and end points, and there exist a way to choose a start point < an end point, then the system is consistent. Otherwise, the *FindMinSet* algorithm finds the independent culprits, involving inconsistent start point and/or the end point.

4.4.1 – Normalization Algorithm

The normalization algorithm presented in this section converts a set of interval ORD-Horn relations into a point-based ORD-Horn formula. But first, we must introduce the 'negative lookup table' used for absent lower dimensional relations.

Relation	Coordinates (x,y) in lattice	ORD-Horn representation of absent relations
!m	(0,1)	$A^- \neq B^+ \cup A^- \leq B^-$
!s	(1,2)	$A^- \neq B^- \cup A^+ \leq B^+$
!f	(2,3)	$A^+ \neq B^+ \cup A^- \leq B^+$
!eq	(1,3)	$A^- \neq B^- \cup A^+ \neq B^+$
!~f	(0,3)	$A^+ \neq B^+ \cup A^- \leq B^-$
!~s	(1,4)	$A^- \neq B^- \cup B^+ \leq A^+$
!~m	(3,4)	$A^+ \neq B^- \cup B^+ \leq A^+$

Figure 4.6: ORD-Horn representation of lower-dimensional negative relations

This table is used in the Normalization algorithm instead of the negative model. It avoids some computation to simplify the expressions. Next, we define an 'opposite relation' as:

- $X \neq Y$ for $X = Y$
- $X = Y$ for $X \neq Y$
- $X \leq Y$ for $Y \leq X$

Before exploring the core of the Normalization algorithm, we must introduce a few linear procedures used by the Normalization.

- Algorithm get_x: returns the lattice coordinate of a basic relation on the X axis.

```
Input: A basic interval relation r
Output: an integer corresponding to the x-coordinate of r on the lattice
Algorithm get_x (basic relation r)
1       Switch (r)
2               Case b:
3               Case m:
4               Case o:
5               Case ~f:
6               Case ~d: return 0
7               Case s:
8               Case eq:
9               Case ~s: return 1
10              Case d:
11              Case f:
12              Case ~o: return 2
13              Case ~m: return 3
14              Case ~b: return 4
end algorithm get_y
```

- Algorithm get_y: returns the lattice coordinate of a basic relation on the Y axis.

```
Input: A basic interval relation r
Output: an integer corresponding to the y-coordinate of r on the lattice
Algorithm get_y (basic relation r)
1       Switch (r)
2               Case b: return 0
3               Case m: return 1
4               Case o:
5               Case s:
6               Case d: return 2
7               Case ~f:
8               Case eq:
9               Case f: return 3
10              Case ~d:
11              Case ~s:
12              Case ~o:
13              Case ~m:
14              Case ~b: return 4
end algorithm get_y
```

The two algorithms above are a set of switch statements, returning the coordinates over the lattices axis (lines 1 through 14).

- Algorithm Simplify: get the interval lattice of a given ORD-Horn relation.

```
Input: an array of clause
Output: a single clause result of the simplification of the input set of clauses
Algorithm Simplify (clause[])
1       Sort every clause by size in decreasing order
2       k = number of clauses
3       while (k>1) do {
4               Next_Clause = clause[k]
5               for each relation r in Next_Clause do {
6                       r = Current_Relation
7                       remove the opposite of r in each other clause
8               }
9               remove Next_Clause
10              k--
11      }
12      return clause[1]
end algorithm Simplify
```

Simplify translates a set of point-based relations over a lattice axis into an ORD-Horn formula. The procedure takes the largest clause and 'shrinks' with respect to smaller clauses. Line 1 sorts every clause by size. The largest clause is held in clause[1]. While there remains more than one clause (line 3), the algorithm picks the last clause (line 4) and remove every 'opposite' relation held in the last clause from the every other clause (lines 5 through 8). Lines 9 and 10 remove the last clause from the current set of clauses and decrement the total number of clauses. Finally, line 12 returns the clause containing

- Algorithm lookup_Negative_Table: Return the point based formulae of an absent basic interval relation in ORD-Horn form.

Input: a pair of integers

Output: a clause representing an absent relation

Algorithm lookup_Negative_Table (x,y)

1 if (x==1 & y == 1) return ($A^- \neq B^+ \cup A^- \leq B^-$)

2 else if (x==1 & y == 2) return ($A^- \neq B^- \cup A^+ \leq B^+$)

3 else if (x==2 & y == 3) return ($A^+ \neq B^+ \cup A^- \leq B^+$)

4 else if (x==1 & y == 3) return ($A^- \neq B^- \cup A^+ \neq B^+$)

5 else if (x==0 & y == 3) return ($A^+ \neq B^+ \cup A^- \leq B^-$)

6 else if (x==1 & y == 4) return ($A^- \neq B^- \cup B^+ \leq A^+$)

7 else if (x==3 & y == 4) return ($A^+ \neq B^- \cup B^+ \leq A^+$)

8 return error ("not a lower dimensional relation")

end algorithm lookup_Negative_Table

lookup_Negative_Table returns the point-based formula from an absent atomic lower-dimensional interval relation (lines 1 through 7) or null if the relation is not a lower dimensional relations (line 8).

We can now introduce the Normalization algorithm

- Algorithm Normalization: Transforms an ORD-Horn interval relation into its corresponding ORD-Horn point based formulae.

```
Input: an interval relation R in ORD-Horn
Output: a point based ORD-Horn formula
Algorithm Normalization (ITR-I(O))
1       //Get the convex box
2       integers x_min, y_min = 4
3       integers x_max, y_max= 0
4       initialize x_clause to be empty
5       initialize y_clause to be empty
6       initialize AbsentRelation to be empty
7       //Get ranges from the input set of relations (0 ≤ x,y ≤ 4) on the lattice
8       for each basic relation b in R
9               if (x_min > get_x(b)) x_min = get_x(b)
10              if (y_min > get_y(b)) y_min = get_y(b)
11              if (x_max > get_x(b)) x_max = get_x(b)
12              if (y_max > get_y(b)) y_max = get_y(b)
13      for (x = x_min to x_max)
14              add X_Positive_Model(x) to x_clause
15      for (y = y_min to y_max)
16              add Y_Positive_Model(y) to Y_clause
17      for (x = x_min to x_max)
18              for (y = y_min to y_max)
19                      if (relation(x,y)∉ R)
20                              add lookup_Negative_Table(x,y) to AbsentRelation
21      x_clause  = Simplify(x_clause)
22      y_clause  = Simplify(y_clause)
23      return x_clause ∩ y_clause ∩ (conjunction of all element in AbsentRelation).
end algorithm Normalization
```

4.4.2 – Sorter Algorithm

The Sorter algorithm split the set of clauses given by the Normalization algorithm into two sets of relations involving respectively the starting and the ending points of the interval to be added. It also checks for consistency on equal interval bounds. If an inconsistency is found at this level, the algorithm returns the set of inconsistent relations.

Input: A constraint set C as a conjunctive normal ORD-clausal formula $\{c_1, c_2, \ldots, c_n\}$

Output: A pair of constraint sets over boundary points of the interval *New* as L^+ and L^-, or a set of inconsistent relations

Algorithm Sorter (a set of Clause C)

1 initialize a set for the chosen literals, L := empty;

2 initialize a set for inconsistent relations, N := empty;

3 FOR each clause c_i in C DO

4 IF c_i is unit-clause DO

5 $L := L \cup c_i$;

6 ELSE IF there exist an $\neq$ type literal p from c_i such that p does not conflict with any literal in L DO

7 assign $L := L \cup \{p\}$;

8 ELSE IF the positive literal $p1$ from c_i does not conflict with any literal in L DO

9 assign $L := L \cup \{pl\}$;

10 ELSE DO

11 $N := N \cup \{p\}$;

12 IFN is not empty DO

13 RETURN N;

14 Group L into L^- with elements involving New^- only and L^+ with elements involving New^+ only;

15 RETURN L^- and L^+;

end algorithm Sorter

// {note that each element of L is only a literal and so, must involve either New^- or New^+ but not both}

Lines 1 and 2 initialize the set L in which the chosen relations will be stored, and the set N in which inconsistent relations might be stored if the system is inconsistent. Then, for each clause (line 3) store unit relations in L(Lines 4 and 5). If the current clause is not a unit clause, choose and store the relation involving inequality if it does not conflict with any other relation in L (lines 6 and 7). If the inequality relation conflicts with some relation in L, we pick the highest dimensional relation and store it in L if it doesn't conflict with L (lines 8 and 9). Otherwise, store the relation *p* in the set N as one culprit for leading the system to inconsistency (lines 10 and 11). Once every clause have been computed, if the Set of inconsistent relations is not empty, return it (lines 12 and 13). Otherwise, sort every relation into two set, L^+ and L^-, respectively involving the starting and ending point of the new interval (line 15).

When the algorithm terminates, if no inconsistency have been found among eventual *equal* relations, the *FindMinSet* Algorithm is run on both sets, L^- and L^+. Each of the algorithms described above, the *Normalization,* the *Sorter*, and the *FindMinSet* algorithm are polynomial algorithms. However, in case of non pre-convex constraints, i.e., for the general unrestricted input, the *sorter* algorithm may have to backtrack when inconsistency is detected by the *FindMinSet* algorithm.

The next section deals with the other 'more generic' tractable subclasses of Allen's Interval Algebra.

Chapter 5 – 17 maximum tractable subalgebras

The ORD-Horn Algebra was the first Maximal tractable algebra to be discovered. This algebra's properties are very different from the other algebra's. Since it has been described in details in the previous chapter, we will only consider the 17 other MTSs in the remaining of this chapter. This chapter starts with a discussion on past and current interest on the study of tractability in Allen's Algebra. Next, we establish a formal graphical approach of the MTSs and describe a simple approach to count them. This is a new way to understand and reason on these algebras. We believe this approach will to help the reader understanding the nature of the tractable algebras. Finally, we propose a systematic way to classify any interval relation in one of 17 of these algebras.

5.1 – Definition of maximum tractable subalgebra (MTS)

A maximum tractable algebra is a set of interval relations respecting the definition of algebra for which there exits an algorithm that can decide satisfiability for any problem in polynomial time. Also, if any relation is added to the algebra, either one of the condition mentioned above is violated. To resume, an MTS is:

- Closed under composition, set union, set intersection and converse operations
- Tractable, as proved by Krokhin et al
- Maximal: no super algebra of a MTS is tractable.

5.2 – Literature

After the ORD-Horn algebra was discovered Thomas Drakengren, Peter Jonsson discovered in 1997 21 large tractable subclasses and, identified eight of them as being maximal tractable algebras. Shortly after, they discovered 10 more MTSs. Finally, Krokhin proved that reasoning in a fragment not entirely included in one of the 18 MTS is NP-complete.

In their paper "Reasoning about Temporal Relations: The Tractable Subalgebras of Allen's Interval Algebra", Krokhin et al expressed the formal definition of the 18 MTSs. This is their original representation. We must inform the reader that this notation uses 'p' (precede) for our interval relation 'b' (before) basic relation, and the superscript '$^{-1}$' for our prefix '~' defining the 'converse' relation. Furthermore, the superscript '$^{\pm 1}$' is used to denote the conjunction of two conditions. For example, $(o)^{\pm 1} \subseteq r \Leftrightarrow (d^{-1})^{\pm 1}$ means that both,

- $o \subseteq r \Leftrightarrow d^{-1}$

and

- $o^{-1} \subseteq r \Leftrightarrow d$

hold.

The notation originally designed by Krokhin et al is given below:

1. $Sp = \{r \mid r \cap (pmod^{-1}f^{-1})^{\pm 1} \neq \emptyset \Rightarrow (p)^{\pm 1} \subseteq r\}$
2. $Sd = \{r \mid r \cap (pmod^{-1}f^{-1})^{\pm 1} \neq \emptyset \Rightarrow (d^{-1})^{\pm 1} \subseteq r\}$
3. $So = \{r \mid r \cap (pmod^{-1}f^{-1})^{\pm 1} \neq \emptyset \Rightarrow (o)^{\pm 1} \subseteq r\}$
4. $A1 = \{r \mid r \cap (pmod^{-1}f^{-1})^{\pm 1} \neq \emptyset \Rightarrow (s^{-1})^{\pm 1} \subseteq r\}$
5. $A2 = \{r \mid r \cap (pmod^{-1}f^{-1})^{\pm 1} \neq \emptyset \Rightarrow (s)^{\pm 1} \subseteq r\}$
6. $A3 = \{r \mid r \cap (pmodf)^{\pm 1} \neq \emptyset \Rightarrow (s)^{\pm 1} \subseteq r\}$
7. $A4 = \{r \mid r \cap (pmodf^{-1})^{\pm 1} \neq \emptyset \Rightarrow (s)^{\pm 1} \subseteq r\}$
8. $Ep = \{r \mid r \cap (pmods)^{\pm 1} \neq \emptyset \Rightarrow (p)^{\pm 1} \subseteq r\}$
9. $Ed = \{r \mid r \cap (pmods)^{\pm 1} \neq \emptyset \Rightarrow (d)^{\pm 1} \subseteq r\}$
10. $Eo = \{r \mid r \cap (pmods)^{\pm 1} \neq \emptyset \Rightarrow (o)^{\pm 1} \subseteq r\}$
11. $B1 = \{r \mid r \cap (pmods)^{\pm 1} \neq \emptyset \Rightarrow (f^{-1})^{\pm 1} \subseteq r\}$
12. $B2 = \{r \mid r \cap (pmods)^{\pm 1} \neq \emptyset \Rightarrow (f)^{\pm 1} \subseteq r\}$
13. $B3 = \{r \mid r \cap (pmod^{-1}s^{-1})^{\pm 1} \neq \emptyset \Rightarrow (f^{-1})^{\pm 1} \subseteq r\}$
14. $B4 = \{r \mid r \cap (pmod^{-1}s)^{\pm 1} \neq \emptyset \Rightarrow (f^{-1})^{\pm 1} \subseteq r\}$
15. $S^* = \{r \mid r \cap (pmod^{-1}f^{-1})^{\pm 1} \neq \emptyset \Rightarrow (f^{-1})^{\pm 1} \subseteq r$, and $r \cap (ss^{-1}) \neq \emptyset \Rightarrow (\equiv) \subseteq r\}$
16. $E^* = \{r \mid r \cap (pmods)^{\pm 1} \neq \emptyset \Rightarrow (s)^{\pm 1} \subseteq r$, and $r \cap (ff^{-1}) \neq \emptyset \Rightarrow (\equiv) \subseteq r\}$
17. $H = \{ r \mid r \cap (os)^{\pm 1} \neq \emptyset \ \& \ r \cap (o^{-1}f)^{\pm 1} \neq \emptyset \Rightarrow (d)^{\pm 1} \subseteq r$, and

    $r \cap (ds)^{\pm 1} \neq \emptyset \ \& \ r \cap (d^{-1}f\text{-}1)^{\pm 1} \neq \emptyset \Rightarrow (o)^{\pm 1} \subseteq r$, and

    $r \cap (pm)^{\pm 1} \neq \emptyset \ \& \ r \not\subseteq (pm)^{\pm 1} \Rightarrow (o)^{\pm 1} \subseteq r\}$
18. $A_{\equiv} = \{r \mid r \neq \emptyset \Rightarrow (\equiv) \subseteq r\}$

This notation can be quite confusing at first, but it is important that the reader understands which elements constitute the Maximal Tractable Subalgebras (MTS). Consider the following example for the MTS Sp:

A relation r belongs to Sp

1. if r contains any of the basic relation among $pmod^{-1}f^{-1}$, then p must belong to r,
2. if r contains any of the relations among $p^{-1}m^{-1}o^{-1}df$, then p^{-1} must belong to r.

We can now move on our graphical representation of the 17 MTSs.

5.3 – Express 17 MTS, a graphical representation

In this thesis, we assign different names to the MTSs than those defined by Krokhin et al. However, the original names will appear in parenthesis so that the reader can refer to the original work. Also, we will rearrange the order the algebras are presented. For each algebra, there is a 'lattice model' and a 'space model' to represent it. But before turning to the graphical representation of the algebras, let us introduce some vocabulary we will use in the rest of this thesis.

Definition 5: A *sublattice* stands for an MTS sub-lattice. It represents an acyclic set of relation, subset of the complete algebra. Every sublattice contains a "pivot", defined below.

Definition 6: A *pivot* is a mandatory relation for a given algebra when other relations are present in the original set. If for a given algebra and a given set of

input relation the pivot is absent, and another relation in the sublattice is present implies that the relation is not in this algebra.

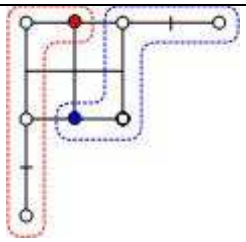
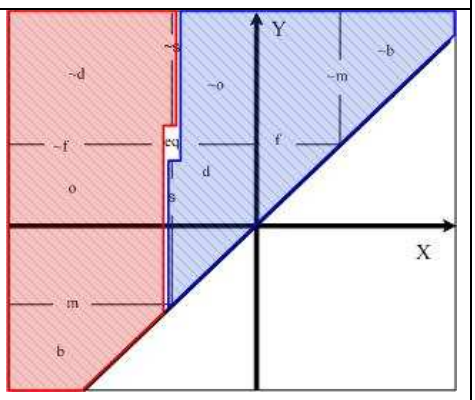
Algebras 1 through 17 can be read as follows:

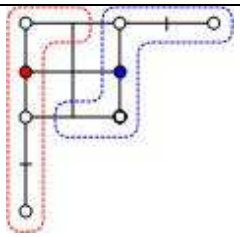
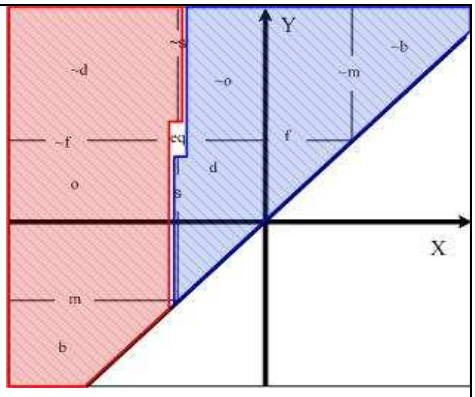
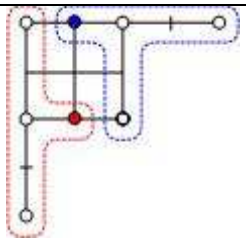
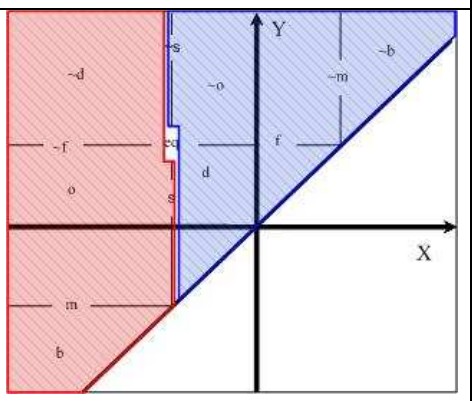
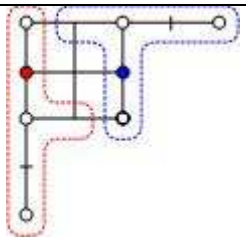
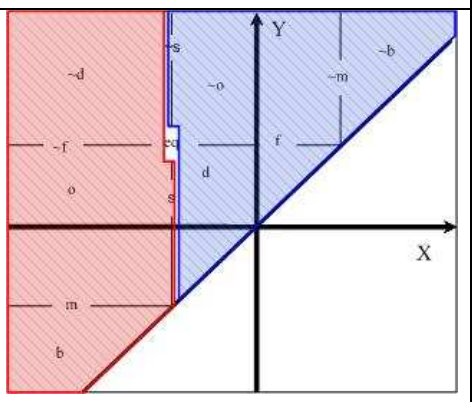
- To check if a relation is in an algebra with the lattice model:

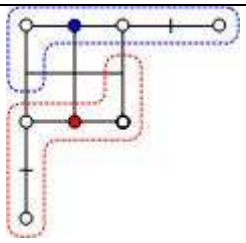
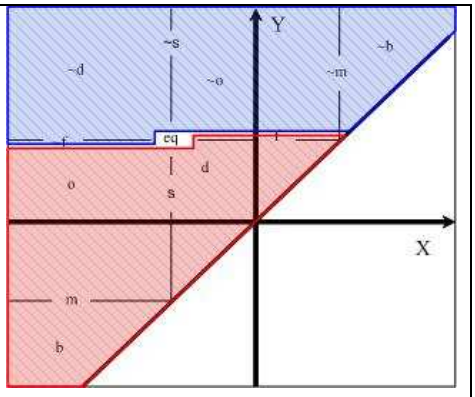
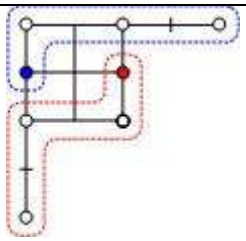
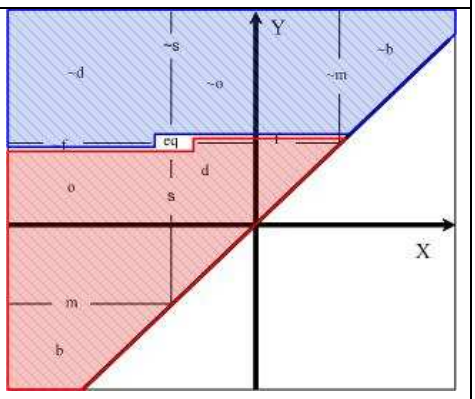
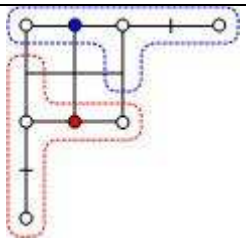
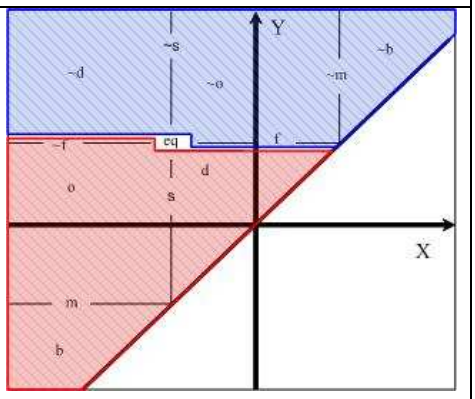
  Only consider the sublattice relations. For each atomic relation belonging to a sublattice, the corresponding pivot must be present, or the relation is not in the algebra.

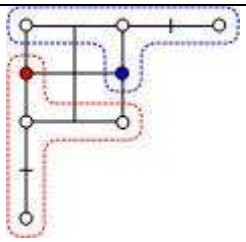
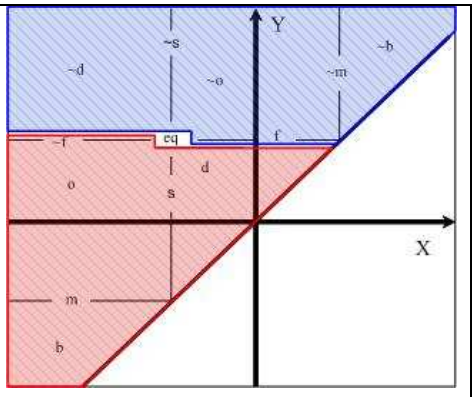
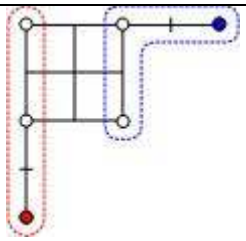
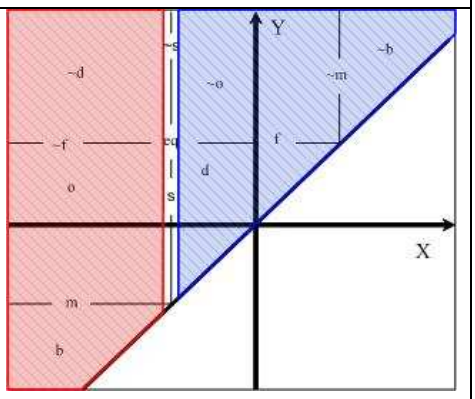
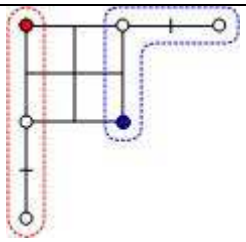
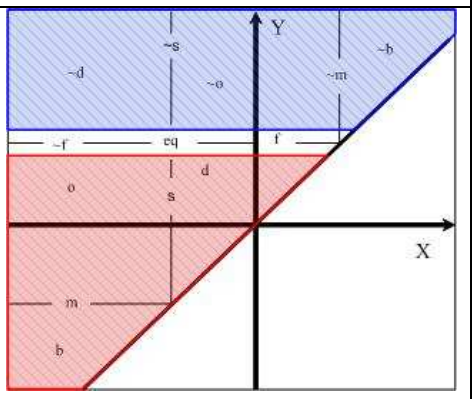
- To check if a relation is in an algebra with the space model:

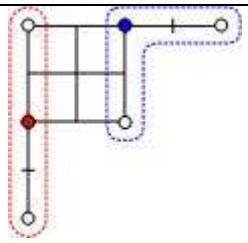
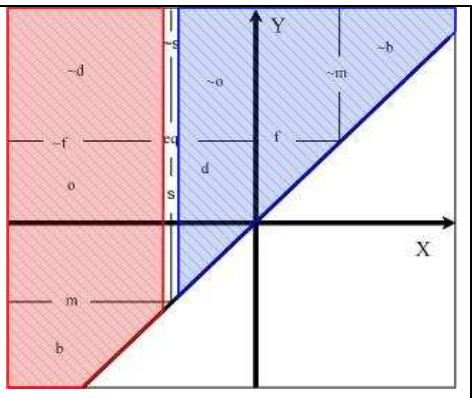
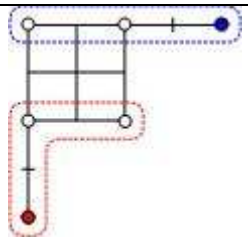
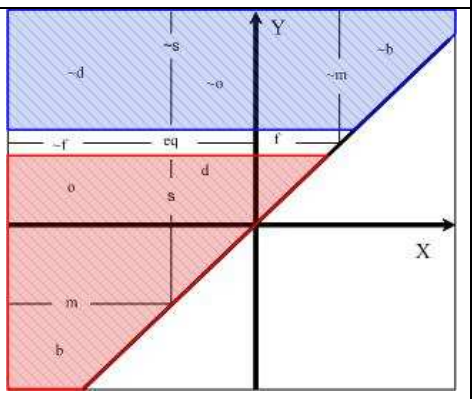
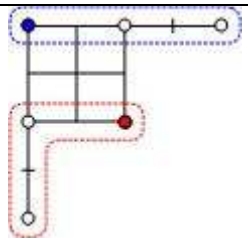
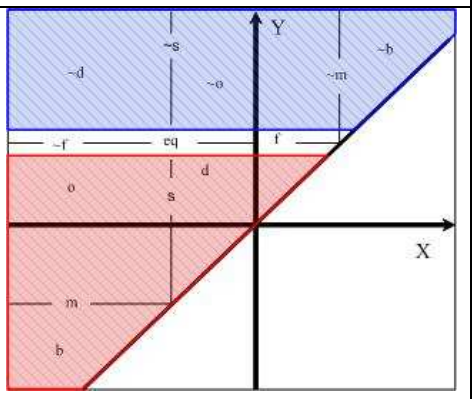
Any atomic relation matching the left side of an implication requires the right side of the implication, or the set of relations is not in the set. Figure 5.1 below shows the graphical representation of these algebras.

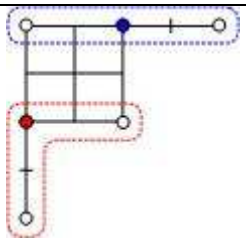
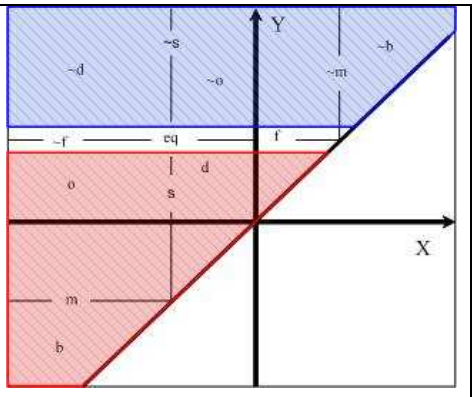
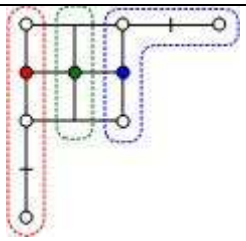
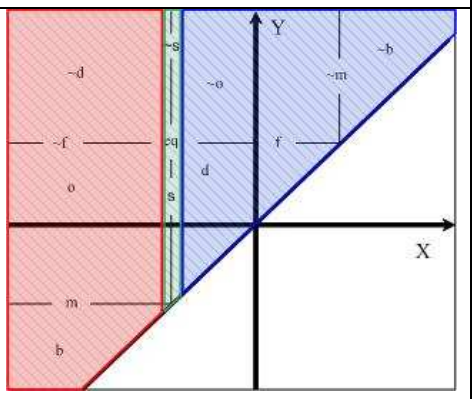
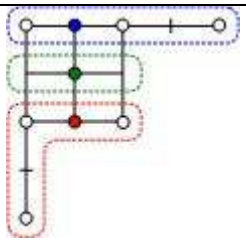
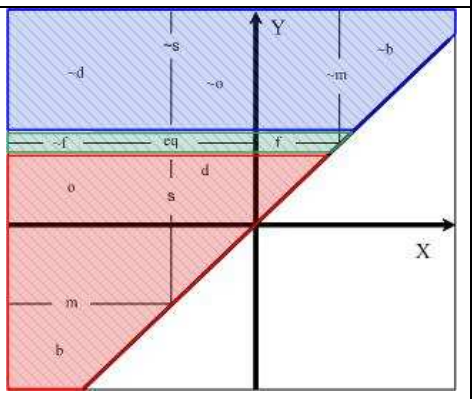
Algebra name and cardinality	Lattice model	Space model
ALJ[1] - (A1) $2^{11}+2^6+2^6+2 = 2178$ (Nx<X)->(~s) (Nx=X0)^(Ny>Y)->(~s) (Nx>X)->(s) (Nx=X)^(Ny<Y)->(s)		

ALJ[2] - (B3) $2^{11}+2^6+2^6+2 = 2178$ (Nx<X)->(~f) (Nx=X)^(Ny>Y)-> (~f) (Nx>X)-> (f) (Nx=X)^(Ny<Y) ->(f)		
ALJ[3] - (A2) $2^{11}+2^6+2^6+2 = 2178$ (Nx<X)->(s) (Nx=X)^(Ny<Y)->(s) (Nx>X)->(~s) (Nx=X)^(Ny>Y)->(~s)		
ALJ[4] - (B4) $2^{11}+2^6+2^6+2 = 2178$ (Nx<X)->(~f): (Nx=X) ^(Ny<Y)->(~f) (Nx>X)->(f) (Nx=X)^(Ny>Y)->(f)		

ALJ[5] - (A3) $2^{11}+2^6+2^6+2 = 2178$ (Ny<Y)->(s) (Ny=Y)^(Nx>X)->(s) (Ny>Y)->(~s) (Ny=Y)^(Nx<X)->(~s)		
ALJ[6] - (B2) $2^{11}+2^6+2^6+2 = 2178$ (Ny<Y)->(f) (Ny=Y)^(Nx>X)->(f) (Ny>Y)->(~f) (Ny=Y)^(Nx<X)->(~f)		
ALJ[7] - (A4) $2^{11}+2^6+2^6+2 = 2178$ (Ny<Y)->(s) (Ny=Y)^(Nx<X) ->(s) (Ny>Y)->(~s) (Ny=Y)^(Nx>X)->(~s)		

ALJ[8] - (B1) $2^{11}+2^6+2^6+2 = 2178$ (Ny<Y)->(~f): (Ny=Y)^(Nx<X)->(~f) (Ny>Y) ->(f) (Ny=Y)^(Nx>X) ->(f)		
ALJ[9] - (Sp) $2^{11}+2^7+2^7+2^3 = 2312$ (Nx<X)->(b) (Nx>X)-> (~b)		
ALJ[10] - (Sd) $2^{11}+2^7+2^7+2^3 = 2312$ (Nx<X)->(~d) (Nx>X)-> (d)		

ALJ[11] - (So) $2^{11}+2^7+2^7+2^3 = 2312$ (Nx<X)->(o) (Nx>X)->(~o)		
ALJ[12] - (Ep) $2^{11}+2^7+2^7+2^3 = 2312$ (Ny<Y)->(b) (Ny>Y)->(~b)		
ALJ[13] - (Ed) $2^{11}+2^7+2^7+2^3 = 2312$ (Ny<Y)->(d) (Ny>Y)->(~d)		

ALJ[14] - (Eo) $2^{11}+2^7+2^7+2^3 = 2312$ (Ny<Y)->(o) (Ny>Y)->(~o)		
ALJ[15] - (S*) $2^{10}+2^8+2^6+2^6+$ $2^4+2^4+2^2+2^0=1445$ (Ny<Y)->(~f) (Nx=X)->(eq) (Ny>Y)->(f)		
ALJ[16] - (E*) $2^{10}+2^8+2^6+2^6+$ $2^4+2^4+2^2+2^0=1445$ (Nx<X)->(s) (Ny=Y)->(eq) (Nx>X)->(~s)		

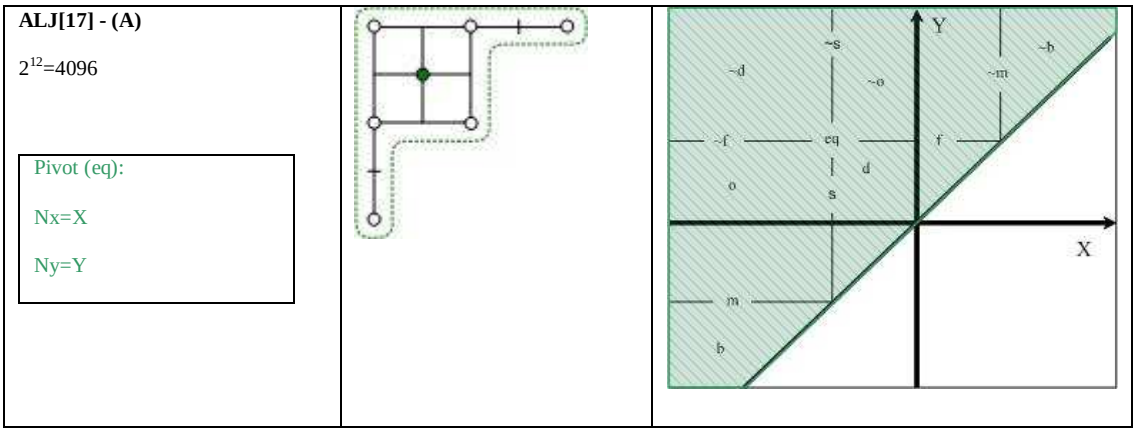


Figure 5.1: Graphical representation of 17 MTSs

Example: The sublattices for ALJ[1] are {b, m, o, ~f, ~d, s} and {~b, ~m, ~o, f, d, ~s}, and there respecting pivots are {s} and {~s}.

The cardinality of the 17 algebras is shown in table 5.2. The elements in any of these algebras can easily be counted using the graphical representation. The following example shows how the cardinality of ALJ[1] is computed.

Example: The number of elements in ALJ[1] is given by all the possible assignments of relation $R \in$ ALJ[1]:

- If (s) and (~s) belong to the relation R, then any of the 11 remaining relations might belong to R: 2^{11} elements.
- If (s) belong to the relation R and (~s) doesn't, then any of the 5 remaining relations from the sublattice of (s), and eventually the equal relation might belong to R: 2^6 elements

- If (~s) belong to the relation R and (s) doesn't, then any of the 5 remaining relations from the sublattice of (~s), and eventually the equal (eq) relation might belong to R: 2^6 elements
- If neither (s) nor (~s) belong to the relation R, then only (eq) or no relation might belong to R: 2 elements

Hence, the total number of elements for ALJ[1] is: $2^{11}+2^6+2^6+2 = 2178$. A similar approach will give the cardinality of the remaining algebras.

5.4 – A generic algorithm

In this section, we will propose a generic algorithm capable of solving the OLQTR-I for any of the 17 MTSs in polynomial time. Moreover, in case of inconsistency, the algorithm proposes a minimum number of modifications of the original problem to set back consistency to the problem. However, the class of solvable problems with this technique is far less interesting than the possibilities offered by the ORD-Horn algebra. This issue is discussed in the following section

5.4.1 – Classes of solvable problems

The algorithm presented in this section can only solve problems belonging to one of the MTS presented earlier in this chapter. This considerably reduces the usability of this algorithm. As a matter of fact, since a relation between two committed intervals consists of a single basic relation, only the pivots and the equal relations are allowed between two committed intervals. Consider the two constrained temporal networks in Figures 5.2 and 5.3. Every relation in the first

figure belongs to the set {o, eq, ~o} and is therefore tractable with ALJ[8] or ALJ[14], whereas the committed relations in Figure 5.3 does not belong to any MTS. The set of relations between the intervals are {~b, o, d, eq, ~o} and hence is not tractable with the algorithm presented in this section. Note that it might be tractable within, and only within the ORD-Horn class, depending on the constraints between the new interval to be added and the committed intervals.

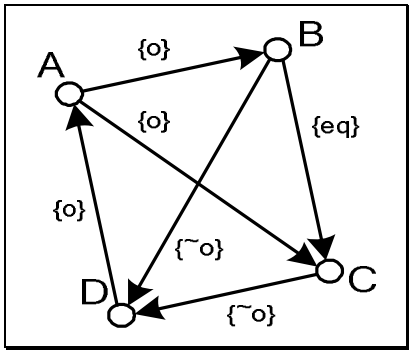


Figure 5.2: Constrained temporal network in ALJ[11] and ALJ[14]

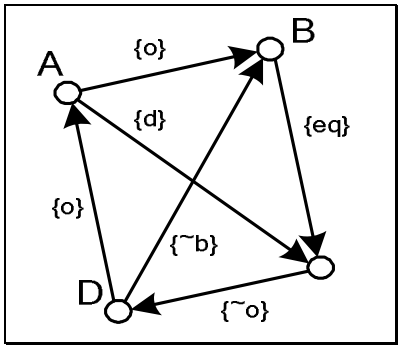


Figure 5.3: Constrained temporal network that does not belong to any MTS

Hence, the class of solvable problems requires that the relations between committed intervals satisfy the following condition:

Assertion 5: For any pair of committed intervals I_1 and I_2:

- I_1 r $I_2 \Rightarrow$ r $\in$ {pv, ~pv, eq} where 'pv' is any basic interval relation except meet.

Proof: Consider a constrained temporal network T where basic relations belong to ALJ[i], 1<i<17 with pivots $\in$ {pv, ~pv, eq}. Suppose a relation r' does not belong to { pv, ~pv, eq}, then, obviously, r' $\notin$ ALJ[i].

5.4.2 – Ordering the committed intervals

Knowing that for a tractable problem, the constraints between committed intervals respects assertion 5; it is possible to assign a total ordering to the set of intervals by redirecting the edges of 'inverse' intervals to set them in a 'forward' type of relations only. Hence the previous tractable example in Figure 5.2 can be modified to give the constrained temporal network of Figure 5.4.

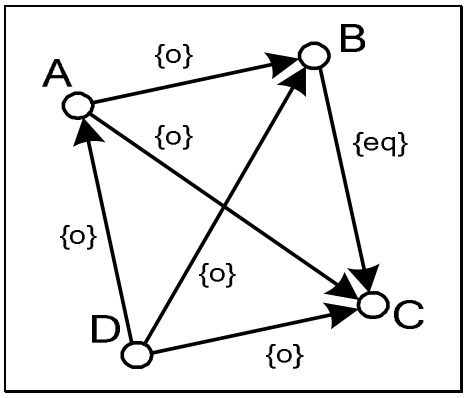


Figure 5.4: Forward relations of figure 5.2

The ties generated by equality are assigned arbitrarily. Hence, the total order T over the basic relation overlap defining the temporal network above is:

- $T_{(o)} = \{D \succ A \succ C \succ B\}$.

This total ordering can be viewed as a combination of constraints like:

- (D {o} (A {o} (C {eq} B)))

Assertion 6: there exist a total order $T_{(pv)}$ for any temporal constraint network belonging to one of ALJ[i] 1<i<17.

Proof: from assertion 5, the relation between any two intervals in a constrained network must belong to {pv, ~pv, eq}. Since for any intervals I_1 and I_2, the trivial property I_1 {pv} $I_2 \equiv I_2$ {~pv} I_1, every relation r can be expressed as its non-inverse basic relation by reordering the intervals. The total order generated by equality relation does not matter and can be chosen arbitrarily.

To summarize:

- The total ordering for the algebras involving before (b), overlap (o) or during (d) pivots and their respective inverses is given by the order of either boundaries point.
- The total ordering for the algebras involving start (s) pivot and its inverse is given by the end point.
- The total ordering for the algebras involving finish (f) pivot and its inverse is given by starting point.
- The basic relation equality belongs to every algebra and the total ordering for the algebra with equality as basic relation or as pivots is arbitrary.

5.4.3 – Deciding satisfiability

In order to remain in the tractable class, the interval relation constraining a new interval in OLQTR-I must also belong to the ALJ[i] and hence includes one of the pivots relation or the equality relation.

Proposition 7: The satisfiability of the OLQTR-I problem for a set of constrained temporal network is equivalent to finding a new non-null total order $NewT_{(pv)}$ including the new interval.

Example: Suppose there exist a total order T on some temporal constraint network such as:

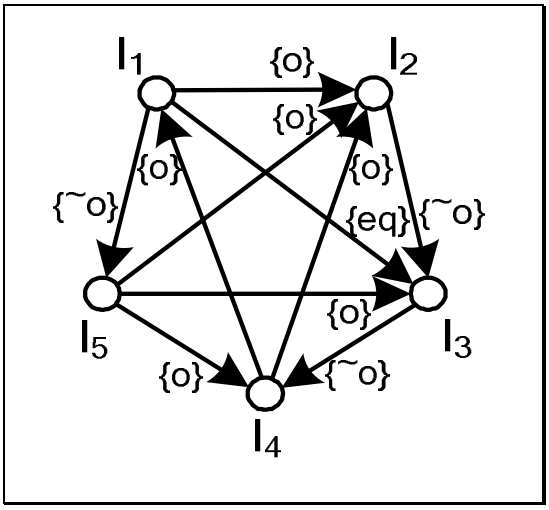


Figure 5.5a

Then, one can easily construct a scenario where every constraint is satisfied. Such a scenario is presented in the following figure:

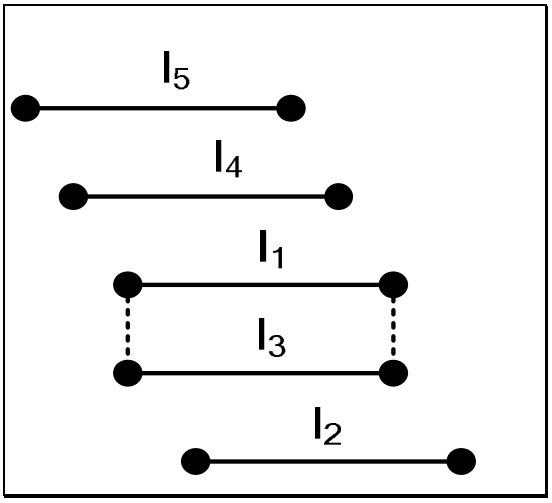


Figure 5.5b

Suppose we now want to add a new interval satisfying the following constraints:

- New {o, f, d, eq, ~o} I_1
- New {b, o, s, ~f} I_2
- New {b, m, o} I_3
- New {~o, ~d} I_4
- New {~b, ~o, ~s} I_5

Obviously, each relation belong to the ALJ[14] with {o, ~o} as pivots. Thus we can construct the total order T of committed intervals as follow:

- $T = (I_5 \{o\} (I_4 \{o\} ((I_1 \{eq\} I_3) \{o\} (I_2))))$

$\equiv$

- $T = (I_5 \succ I_4 \succ I_1 \succ I_3 \succ I_2)$

We now must find a position where to insert interval New respecting the preceding order. To do so, we only redirect the constraints defined above only considering the pivots in forward way:

- $[New \{o\} I_1 \text{ OR } I_1 \{o\} New] \equiv [New \succ I_1 \text{ OR } I_1 \succ New]$
- $[New \{o\} I_2] \equiv [New \succ I_2]$
- $[New \{o\} I_3] \equiv [New \succ I_3]$
- $[I_4 \{o\} New] \equiv [I_4 \succ New]$
- $[I_5 \{o\} New] \equiv [I_5 \succ New]$

It now becomes trivial to find an assignment for New if it exists.

Lemma 7: inconsistency can be found by looking for a cycle on the directed temporal constrained network with the new interval.

Assertion 8: The directed temporal constrained network is a complete graph.

Proof: (Trivial) there exist a relation between any two intervals in the network.

Assertion 9: in OLQTR-I, choosing a relation other than one of the pivots or the equality relation translates the problem into a different algebra on the next iteration on ALJ[i].

Proof: This proof directly follows Assertion 5. After the new interval has been added to the set of committed intervals, at least one relation between New and a committed interval $\notin$ ALJ[i] and hence, becomes only tractable in the next set of iterations if the relation between a new interval and every committed interval belongs to the ORD-Horn algebra, only algebra to include every atomic relation.

Assertion 10: on detecting inconsistency in an OLQTR-I, relaxing at most n/2 relations by adding a pivot to these relations, with n being the number of intervals in the problem, is sufficient to restore consistency of the problem.

Proof: This proof relies on the bipartite nature of the OLQTR-P. At most n/2 constraints conflict with each other. Adding a 'pivot' to at most n/2 relation that only contain 'inverse pivot' relaxes every constraint, resulting in a directed temporal constraint network with at least 'pivot' as a possible assignment for each relation , thus eliminating every cycle created by more constrained relations (lemma 7).

5.4.4 – Algorithm

This algorithm transposes the OLQTR-I to an OLQTR-P problem to decide the satisfiability of the overall problem since considering only one of the boundary points of each pivot is sufficient to decide consistency as mentioned in assertion 6.

Input: A OLQTR(I) problem within ALJ[i] (1<i<17)

Output:

Algorithm TrSAT

```
1       for each relation Ri in T(pv) that does not contain a pivot and it's inverse
2               if N (~pv) Ri
3                       mark Ri as '>'
4               else if N (pv) Ri
5                       mark Ri as '<'
6               else if N (=) Ri
7                       mark Ri as '='
8               end else
9       end for
10      run PoSeq on T(pv)
11      if inconsistent do
12              if inconsistent points are marked < do
13                      suggest (pv) for each inconsistent point
14              else if inconsistent points are marked > do
15                      suggest (~pv) for each inconsistent point
16              end else
17      end if
end Algorithm
```

Chapter 6 – Conclusions

In this work, we first develop an algorithm that decides satisfiability for the OLQTR-P problem, and on inconsistency detection, proposes a minimal set of relations to remove to 'roll back' to consistency. We then use this algorithm to extend the capabilities of the minimum culprit detection to a subset of Allen's interval algebra known as ORD-Horn algebra. Finally, we introduce a graphical way to express every other maximal tractable algebras of Allen's interval algebra, and propose a battery of algorithms to solve the culprit detection problems for these maximal tractable subalgebras. Hence, every tractable case of the Interval Algebra is taken care of in this Thesis.

However, it might be interesting to be able to deal with the full algebra even though the reasoning would not end up in polynomial time. Future works could involve non-polynomial algorithms to deal with such type of problems. Nevertheless, depending on the number of entries for a given problem, the algorithm is not guaranteed to finish ever.

Chapter 7 – Related works

Identification of “causes” behind inconsistency for solely the purpose of improving the search algorithms has been occasionally explored in the CSP literature. Recent works like Jussien and Lhomme proposes optimization for existing conflict based-heuristics algorithms as in the Tabu search (Glover) and in backtracking-based search algorithms. For example, *backjumping* and *dynamic backtracking* attempts to find the appropriate constraints at the time of backtracking. Other works using *Dynamic Variables Ordering* (Bacchus and Van Run) tend to enhance performance by looking forward in the tree search, applying a method to sort the variables in such a way that the general computation of known algorithms becomes faster and more efficient. Another technique using the ‘QUICKXPLAIN’ (Junker) algorithm looks for relaxation for over constrained problems. The purpose of another algorithm called "Ng learning" (Hirayama and Yokoo) is to build and maintain a set of *No-Good* constraints leading to inconsistencies, or for which optimization is not efficient. Hirayama and Yokoo combined this method with asynchronous weak-commitment search algorithm and improved the technique of no-good learning. Similarly, *tabu-search* explores different arrangements of conflicting sets, keeping track of some solutions that should not be explored in the next iterations, rendering them *Tabu*. Several of such ‘optimization algorithms’ have been attempted within the last few years as in the “Guided Tabu search” (Klau et al), where the user is solicited to aid in enhancing the efficiency of the search. *No-Good backmarking* (Richards et al) works almost

the same way. It processes learning of constraints during search for which a failure occurred and intends to ‘repair’ some non-optimal paths. A sense of topological relations between the solutions is implicit in this heuristic. However, only a few works have investigated the direction of providing suggestions to the user (as an output of the system on detection of inconsistency) for the purpose of his/her taking appropriate actions by removing or modifying some constraints. Amilhastre et al suggest considering the user's choices as assumptions.

Identification of “causes” behind inconsistency for the solely purpose of improving the search algorithms has been occasionally explored in the CSP literature. Several of such ‘optimization’ algorithms have been attempted within the last few years as in Klau et al, where the user is solicited to aid in enhancing the efficiency of the search. Richards et al no-Good backmarking, works almost the same way. It processes learning of constraints during search for which a failure occurred and intends to ‘repair’ some non-optimal paths. This thesis proposes an extension of the incremental CSP framework for which interactive decisions involve a method for computing a maximum subset of the user’s choices to ensure consistency. Such incremental temporal scheme is being thoroughly investigated as in Gerevini, but no work combines the incremental scheme with minimum relaxation as we propose in this work. Thus, many of such “intelligent” backtracking heuristics (or other extensions of the classical CSP) may be useful for further investigation on *consistency restoration* in constraint reasoning systems.

Boguraev and Ando attempt to propose a solution to ‘unify’ information extracted from temporal problems using TimeML modeling.

References

[1]. Allen, J. F., (1983). "*Maintaining knowledge about temporal intervals.*" Communications of the ACM, v.26 n.11, pages 832-843.

[2]. Amilhastre, J., Fargier, H., and Marquis, P., (2002). "*Consistency restoration and explanations in dynamic CSPs-Application to configuration*" Artificial Intelligence journal, Vol 135, No. 1-2, pages 199-234.

[3]. Bacchus, F., van Run, P., (1995). "*Dynamic Variable Ordering In CSPs*" Principles and Practice of Constraint programming (CP95), pages 258-275.

[4]. Boguraev, B., Ando, R., (2005). "TimeML-Compliant Text Analysis for Temporal Reasoning" IJCAI-05, page 997.

[5]. Drakengren, T. and Jonsson, P., (1997). "*Twenty one large tractable subclasses of allen's algebra.*" Artificial Intelligence journal, Vol. 93, pages 297-319.

[6]. Gerevini, A. (2003) "Incremental Tractable Reasoning about Qualitative Temporal Constraints." Proceedings of International Joint Conference on Artificial Intelligence, pp1283—1288.

[7]. Glover, F., (1989). "*Tabu search – part I*" ORSA Journal on Computing, Vol 1, No 3, pages 190-206.

[8]. Hirayama, K., Yokoo, M., (2000). "*The Effect of Nogood Learning in Distributed Constraint Satisfaction*" 20th IEEE International Conference on Distributed Computing Systems.

[9]. Junker, U. (2004) "QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems," Proceedings of the National Conference on Artificial Intelligence, pp 167-172

[10]. Jussien, N., and Lhomme, O., (2000). "*Local search constraint propagation and conflict-based heuristics*" Artificial Intelligence journal, Vol. 139, pp 21-45.

[11]. Klau, G. W., Lesh, N., Marks, J., and Mitzenmacher, M., (2002). "*Human guided Tabu search*" In Proceedings of the 18th National Conference on Artificial Intelligence, pages 41-47.

[12]. Krokhin, A., Jeavons, P., and Jonsson, P., (2003). "*Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra*." Journal of the ACM, Vol. 50, No. 5, pages 591-640.

[13]. Launay, F., and Mitra, D. (2005) "Incrementally scheduling with qualitative temporal information," Springer Lecture Notes on AI - 3533, Proceedings of The 18th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, Bari, Italy.

[14]. Ligozat, G., (1996). "*A new proof of tractability for ORD-Horn relations.*" In Proceedings of the 13th National (US) Conference on Articial Intelligence (AAAI-96), pages 395-401, Menlo Park, CA,. AAAI Press.

[15]. Mitra, D., Ligozat, G., and Hossein, L. (1999). "Modeling of multi-dimensional relational constraints between point objects." Proceedings of the Florida AI Research Symposium.

[16]. Nebel, B., and Burckert, H. J., (1995). "*Reasoning about temporal relations: A maximal tractable subclass of Allen's algebra*." *Journal of the ACM*, Vol. 42, No. 1, pages 43-66.

[17]. Richards, T., Jang, Y., Richards, B., (1995). "*Ng-backmarking – an algorithm for constraint satisfaction*" BT technol J Vol 13 No 1 pages 102-109

[18]. Van Beek, P., (1992). "Reasoning about qualitative temporal information." *Artificial Intelligence* Vol 58, pages 297-326

[19]. Vilain, M.B. and Kautz, H. A., (1986). "*Constraint propagation algorithms for temporal reasoning.*" In Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86), pages 377- 382.

[20]. Vilain, M.B., Kautz, H. A., and van Beek, P.G., (1989). "*Constraint propagation algorithms for temporal reasoning; A revised report.*" In Readings in Qualitative Reasoning about Physical Systems, pages 373-381. Morgan Kaufmann, Sant Mateo, CA.