Temporal Reasoning and Constraint Programming
A Survey

Rosella Gennari

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Temporal Reasoning

Temporal Reasoning naturally arises when dealing with problems involving time; the ability to represent and manage temporal knowledge is fundamental in human as well as in artificial agents. This explains why Temporal Reasoning appears in so many areas, including planning, discourse analysis, natural language understanding, medical knowledge representation systems etc. In any activity that involves change, time is an essential feature.

The main goals of Temporal Reasoning are the *formalization* of the notion of time and the construction of a *computational rule–based system* to reason about time.

When we adopt intervals or points as ontological entities to represent time and the constraint programming apparatus as the logical and computational framework, we embark on an important area of Temporal Reasoning: *Temporal Constraint Programming*. In this thesis, we shall study the main literature on this subject which adopts *time as the primitive and unique ontological object*, therefore *events*[1] are identified with their time of occurrence. We shall see the expressive power and computational complexity of these approaches and, above all, how constraint programming techniques can be used to answer temporal queries.

[1] Some authors make a distinction, cfr. [48]: *events* happen instantaneously, that is their time of occurrence is a time point; when they hold during an interval, they are usually called *fluents*. However, referring to events, we mean both concepts.

1.2 Constraint Programming

Constraint Programming can be traced back to research in Artificial Intelligence and Computer Graphics in the sixties and seventies. However only in the last decade, it has emerged as a separate area of research.

1.2.1 Constraint problems and constraint satisfaction

We begin by providing the basic definitions.

A *constraint satisfaction problem*, from now on CP, consists of:

1. a *finite* set of n variables, $x_1, \ldots, x_n$,
2. n sets, $D_1, \ldots, D_n$, that, for every $i = 1, \ldots, n$, D_i is the *domain* of x_i,
3. a *finite* number of *constraints*, where each constraint $C_{i_1 \ldots i_m}$ is a subset of some cartesian product $D_{i_1} \times \cdots \times D_{i_m}$, $m \leq n$, where all i_j are different; $C_{i_1 \ldots i_m}$ is a *constraint on the variables* $(x_{i_1}, \ldots, x_{i_m})$.

The semantic is given by means of an interpretation function in the way described as follows.

Definition 1.0.1 *Let s be a set of variables of a CP; an* instantiation *I is a function defined on a subset t of different variables of s and which assigns to each variable $x_i \in t$ a value in D_i. An instantiation of m variables is called* m–consistent *with a constraint $C_{1 \ldots m}$ on the same variables, or to* satisfy *this constraint, if $(I(x_1), \ldots, I(x_m))$ is in $C_{1 \ldots m}$; whenever there is such a consistent instantiation for $C_{1 \ldots m}$, then this constraint is said to be* solved.

Example 1.1 *Let $\mathcal{C}$ a CP on two variables, x_1 and x_2, whose domains are respectively $D_1 = \{0\}$ and $D_2 = \{0, 1\}$ and just one constraint, that is $C_{12} := \{(0, 0)\}$. The instantiation I of domain $\{x_1, x_2\}$ defined as*

$$\begin{cases} I(x_1) := 0 \\ I(x_2) := 0 \end{cases}$$

is 2–consistent with C_{12}; therefore C_{12} is solved.

Definition 1.0.2 *Let s be a subset of variables of a CP, I an instantiation with domain s and t a sequence consisting of different variables of s; I_t is the instantiation obtained by restricting I to the elements of t and we call it* projection of I *onto* t. *We will call an instantiation, with domain* s, consistent *iff, for every constraint* C *of CP on a sequence* t *of different variables from* s, *the projection* I_t *satisfies* C. *An instantiation is called* m–consistent *(*with the given CP*) iff its domain consists of* m *variables.*

By means of instantiations we can define what a solution for a CP is.

Definition 1.0.3 *Let* $\mathcal{C}$ *be a CP with variables* $x_1, \ldots, x_n$ *and* I *an instantiation of its variables* n–*consistent with it; the* n–*tupla* $(I(x_1), \ldots, I(x_n))$ *is a* solution *for* $\mathcal{C}$. *The* solution set *of* $\mathcal{C}$ *is the set of all solutions for it. If this set is not empty then* $\mathcal{C}$ *is said to be* consistent*; otherwise it is said* inconsistent.

Example 1.2 *Let's consider the CP in example 1.1; the instantiation* I *given in that example is* 2–*consistent (with the given CP) and* $(I(x_1), I(x_2)) := (0,0)$ *is a solution for the CP; therefore this last one is consistent.*

In dealing with Temporal Reasoning, we will see that variables may have infinite domains. In this case, the constraint solving algorithms are based on the algebraic and metric properties of domains.

In general, we can identify these main tasks:

- determining whether the given problem has a solution, that is if it is *satisfiable*,
- producing a solution,
- building up the whole set of solutions.

Definition 1.0.4 *Let* C_1 *and* C_2 *be two CPs on the same set of variables. They are said to be* equivalent *iff they have the same set of solutions.*

In Temporal Constraint Programming, one of the key techniques in the search of all solutions is to reduce the given temporal constraint problem (from now on TCP) to an equivalent one till a minimal TCP, equivalent to the input one, is gained. First we need to define what a "minimal" constraint problem is.

Definition 1.0.5 *A CP is* minimal *iff, for any of its constraint* $C_{1\ldots n}$, *for any* $\overline{a} := (a_1, \ldots, a_n) \in C_{1\ldots n}$, *there is a solution* $\overline{b}$ *for the CP so that* $\overline{a}$ *is a subsequence of* b.

Intuitively, a constraint problem is minimal iff each solution to each of its constraint can be extended to a solution of the problem.

Definition 1.0.6 *Let $\mathcal{C}$ be a constraint problem; if $m(\mathcal{C})$ is minimal and equivalent to $\mathcal{C}$, then it is called a* minimal constraint problem equivalent *to $\mathcal{C}$.*

Reducing a given constraint problem $\mathcal{C}$ to an equivalent minimal constraint problem allows to compute the set of all solutions to $\mathcal{C}$; in general the task of computing $m(\mathcal{C})$ turns out to be an NP–hard problem; we will see it better in chapter 2. Now we are going to see, in general, the main techniques adopted to solve constraint problems.

1.2.2 Algorithms to solve constraints

Over the last two decades, research has been focused on algorithms for solving constraint problems and on identifying those problems whose satisfiability is tractable. Techniques for processing constraints can be broadly divided into two classes.

1. *Constraint enforcing* rules, also known as *constraint propagation* or *local consistency* technique: they enforce various forms of "local consistency"[2] adding inferred constraints to the given problem, which may reduce the search space by eliminating inconsistent values and building up partial solutions.

2. *Search algorithms* to find a solution traversing either the whole space of variable domains or a subset of it given by partial solutions. The best known algorithm for searching a solution is *backtracking*[3]: at every stage of backtracking search, the algorithm tries to extend a partial solution by instantiating a variable towards a solution. In this process we can distinguish three sets of variables, of *past* (already instantiated), of *current* (being instantiated) and of *future* (not yet instantiated) variables. When the algorithm cannot find a value for the current variable to extend the partial solution, then it *backtracks* to the previously instantiated variable x; the value previously assigned is removed from the domain of x and x becomes the current variable. This algorithm is sound and complete, which means that if and only if the given problem is consistent the algorithm finds a solution; however, it is in general time–consuming, so a series of improvements of it have been proposed (*look–ahead*, *look–back* schemes); moreover its performance can be enhanced with heuristic methods, for instance choosing an "optimal" order to process variables or to assign values.

Algorithms to solve constraints generally interleave these two techniques; for example, the first algorithm proposed to solve problems whose constraints

[2]We are going to precisely state what we mean by "local consistency" in definition 1.0.7.
[3]For an account on backtracking algorithms in Constraint Programming, cfr. [29].

are relations of Allen's algebra[4] first reduces the given problem to a local consistent and equivalent one (namely a "path–consistent" one) and then it uses backtracking for finding a solution.

The peculiarity of Constraint Programming is given by its inference rules; we are going to introduce them in the next paragraph and see how we can apply them in each temporal frameworks in chapter 2.

Constraint propagation and local consistency

Constraint propagation algorithms transform a given constraint problem into an equivalent one deducing new constraints; this procedure restricts the set of partial solutions. In fact these algorithms aim neither at checking or finding a solution, nor at constructing the set of all solutions; instead of dealing with the (global) consistency of the constraint problem, they try to approximate it in some loose sense, that is they look for *local consistency*, defined in the way stated below.

Definition 1.0.7 *Let C be a CP on n and $1 \leq k \leq n$:*

1. *C is called 1–consistent iff, for every variable x_i of the problem, $D_i \neq \emptyset$ and $D_i = C_i$ whenever there is a constraint C_i on x_i;*
2. *if $1 < k$, C is called k–consistent iff, taken any $(k-1)$–consistent instantiation I on variables $x_1, \ldots, x_{k-1}$ and any variable x_k different from this one, I can be extended to a k–consistent instantiatiation on $x_1, \ldots, x_{k-1}, x_k$.*

We will call a problem *locally consistent* if it is $(k+1)$–consistent for some k less than the number of variables of the constraint problem.

In the literature different forms of local consistency were introduced before the notion of k–consistency was; but we will see that they can be in general characterized as k–consistency for some k.

Arc–consistency is one of the best known and widely used notions of local consistency; it was introduced for binary constraints and then extended to arbitrary ones; for our purposes the original version, as stated below, is sufficient.

Definition 1.0.8 *Let C_{ij} be a binary constraint on (x_i, x_j), D_i the domain of x_i and D_j of x_j; C_{ij} is* arc–consistent *iff the following conditions are fulfilled:*

[4] Cfr. section 2.2.

AC(CP)

1. $S_0 \leftarrow \mathcal{C} \cup \{C^{-1}\} \,:\, C \in \mathcal{C}$
2. $S \leftarrow S_0$
3. **while** $S \neq \emptyset$ **do**
4. choose $C_{ij} \in S$
5. $D_i' \leftarrow D_i$
6. $D_i \leftarrow \{a_i \in D_i \,:\, \exists a_j \in D_j,\, (a_i, a_j) \in C_{ij}\}$
7. **if** $D_i \neq D_i'$ **then** $S \leftarrow S \cup \{C_{kl} \,:\, C_{kl} \in S_0,\, k = i \vee l = i\}$
8. $S \leftarrow S - \{C_{ij}\}$

Table 1.1: Algorithm to enforce arc–consistency

1. for all $a_i \in D_i$, there is $a_j \in D_j$ such that $(a_i, a_j) \in C_{ij}$;

2. for all $a_j \in D_j$, there is $a_i \in D_i$ such that $(a_i, a_j) \in C_{ij}$.

It is immediate from the definitions to see that a *binary* CP, that is a CP with only binary constraints, is arc–consistent iff it is 2–consistent; this is the only case we will be interested in. An algorithm to enforce arc–consistency is presented in table 1.2.2. In the algorithm, $\mathcal{C}$ is the set of all constraints, C^{-1} indicates the *transposed* of the binary constraint C:

$$C^{-1} := \{(b,\, a) :\; (a,\, b) \,\in\, C\}$$

In this algorithm, $\mathcal{C}$ is the set of given constraints C; S is instantiated to the union of $\mathcal{C}$ and the set of transposed constraints of C belonging to $\mathcal{C}$. In the while loop, domains are reduced trying to to enforce arc–consistency. In general, when enforcing k–consistency, we are trying to reduce the input CP to an equivalent one which is k–consistent; if the given constraint problem is inconsistent, then the algorithm detetects it, for instance returning an empty domain.

In our framework, *path–consistency* is a more important concept than that of arc–consistency, as we will see better in chapter 2. Defining path–consistency requires an operation of *composition* between binary constraints; let's assume it is given and denote it by $\odot$[5].

Definition 1.0.9 *Let $\mathcal{C}$ be a CP and C_{1n} a constraint on (x_1, x_n). The constraint C_{1n} is* path–consistent *iff, for any n-tupla $t := (x_1, \ldots, x_n)$ of variables of $\mathcal{C}$, any consistent instantiation $(a_1, \ldots, a_n)$ of t satisfies the following condition: for any couple of constraints C_{ik} and C_{kj} so that x_k is a variable in t, $(a_i, a_j) \in C_{ik} \odot C_{kj}$.*

[5]Its definition depends on the particular framework we are using.

LRPC(C)

1. **function** LRPC (**var** M: **matrix**): **boolean**
2. **repeat**
3. $M \leftarrow M^2$
4. **until** $M = M^2$
5. **return** $M \neq \emptyset$

Table 1.2: LRPC algorithm to enforce path–consistency

In [41] Montanari proved, arguing by induction, that path–consistency is equivalent to 3–consistency.

Proposition 1.0.1 (Montanari74) *A CSP is path–consistent iff it is* 3*–consistent.*

In table 1.2, we introduce the algorithm we will use in our thesis to enforce path–consistency; as we will see, in each TCP, the operation of *intersection between constraints* is the set–theoretic one. Given a TCP on n variables, its *constraint matrix* M is just an $n \times n$ matrix whose entry M_{ij} is the constraint on (x_i, x_j)[6]; a solution of a constraint matrix M is an n–tupla $(a_1, \ldots, a_n)$ such that, for every $i, j \leq n$, (a_i, a_j) satisfies the constraint M_{ij}; if, for some $i, j \leq n$, we get $M_{ij} = \emptyset$, then we know the TCP is inconsistent.

The $n \times n$ matrix *intersection* and *composition* between $n \times n$ constraint matrices M and N are so defined:

$$(M \cap N)_{ij} := M_{ij} \cap N_{ij}$$

$$(M \odot N)_{ij} := \bigcap_{k \leq n} M_{ik} \odot N_{kj}$$

The power M^n is defined by induction as usual: $M^1 := M$; $M^{n+1} := M^n \odot M$.

In order to prove the soundness of this algorithm (e.g., the resulting TCP is indeed path–consistent), we have to define the operation of composition; therefore we will demonstrate this for each of our framework in a different manner.

[6] We will see, in each temporal framework, how to reduce every TCP to an equivalent one which has exactly one constraint between each pair of variables.

1.3 Temporal reasoning and Constraint Programming

A Temporal Reasoning system consists of a temporal knowledge base, a procedure to check its consistency and an inference mechanism able to derive new information and get a solution or all solutions to queries. Temporal Reasoning tasks are formulated as constraint satisfaction problems; therefore, the constraint satisfaction tecniques can be used to check consistency, to search for solutions or all solutions to the given problem.

Events are the primitive entities in the knowledge base; in Temporal Constraint Programming they are characterized by means of their time of occurence, which can be given by intervals or points.

Temporal information can constraint events to happen at a particular time (e.g., "E happens at 5:00 pm") or to hold during a time interval (e.g., "E takes three hours"); moreover it can state relations between events, of a qualitative type (e.g., "E1 is before E2") or of a metric one (e.g., "E1 has started at least three hours before E2"). Given temporal information of this kind, in temporal constraint programming we are able to answer queries of the following kinds.

1. Is it consistent to believe that E holds at time t or that E1 happens before E2?
2. At what time can E happen\hold? At what time can E1 and E2 hold?
3. What are all the possible times when E can happen\hold? What are all the possible temporal relations between event E1 and E2?

In Temporal Constraint Programming, we are going to study better in chapter 2, variables are always interpreted over rational\real points or intervals. A solution to a query on events is achieved assigning to variables values on the rational\real line so that these values are consistent with the constraints stated in the problem; these values represent possible time of occurence of the events of the query.

Constraints can be either extensionally characterized by means of either real or rational numbers[7], or intensionally represented as (finite) set or relations of some algebra[8]. According to the formalization of constraints and the time unit chosen, we have been able to classify the research in this field into three main streams:

- temporal reasoning with metric information,

[7] Cfr. section 2.1.
[8] Cfr. section 2.2.

- qualitative approach based on Allen's interval algebra,
- a mixed approach based on metric and qualitative constraints.

We are going to introduce them in the next thre subsections.

1.3.1 Temporal Reasoning with metric information

In the quantitative–metric approach, temporal primitive entities are points, ranging over real or rational numbers. Constraint propagation algorithms are based on the metric properties of the variable domain. In what we call the "original" temporal framework, constraints are unions of finite sets of real intervals; lately, they have been extended to unions of interval–sets like $[l, r] - \{b_1, \ldots, b_n\}$. The satisfiability problem for general temporal constraints is NP–hard; therefore authors have studied particular classes of Temporal Constraint Problems, namely Simple Temporal Constraint Problems, backtracking algorithms and constraint propagation algorithms to achieve some forms of local consistency or to approximate it[9].

1.3.2 Qualitative approach based on Allen's interval algebra

In the qualitative approach, constraints are intensionally defined as relations between intervals or points. However, the main work concerns Allen's interval algebra, IA, where constraints are relations between intervals; we examine it in section 2.2 and there we see how qualitative point relations can be reduced to Allen ones. Being the satisfiability of IA an NP–hard problem, series of alternatives have been proposed:

- **solving the problem exactly** but designing algorithms which are efficient in practice, although the worst case analysis shows them to be exponential in time (e.g., various forms of backtracking algorithms)[10];
- producing **approximation algorithms** which run in polynomial time and prune the search space (path–consistency, ordering heuristics etc.);
- **reduction to IA subalgebras**, of which the satisfiability can be computed in polynomial time, to assemble a solution for IA constraint problems.

[9] We will see two algorithms to approximate path–consistency on p. 29.
[10] Cfr. [29].

These three approaches are not so clearly distinct: authors prefer to interleave these techniques in order to improve the search.

1.3.3 Mixed approaches

In this framework, the other ones are mixed in order to gain in expressiveness, trying not to loose the tractability of the problem; however not always the complexity results are optimal. If in the first approach the ontological entities are only points and in the second one, based on Allen's interval algebra, the primitive entities are intervals, in this third approach points and intervals are both primitive objects of the language; therefore new relations\constraints are introduced in order to "relate" points and intervals.

Some authors have studied particular metric TCPs in order to find new subalgebras of IA; we have classified this work as part of the qualitative approach, because its main goal is IA. Instead, we consider an approach "mixed" when it aims at using both the expressive power of the qualitative and of the quantitative approaches to create "new" temporal frameworks, of which the satisfiability can be decided in polynomial time. The research in this direction is one of the most promising[11], anyway the relative literature is still scarce.

[11] Cfr. [49].

Chapter 2

Temporal Reasoning and Constraint Programming

2.1 Temporal Constraints with metric information

In the quantitative approach to temporal reasoning with constraints there is a finite number of variables ranging over rational or real numbers; variables stay for time–points.

2.1.1 A first order language

A first order language with equality, L_m, is introduced to formalize the problem; its non logical symbols are:

1. a 2–place relation symbol, $\leq$, whose intended interpretation is the non–strict canonical order relation over rational\real numbers;
2. a 2–place function symbol, $-$, whose intended interpretation is the subtraction operation between rational\real numbers;
3. as many constant symbols as the rational\real numbers are.

We will refer to constant symbols as rational\real numbers, because their interpretation is fixed; the same convention is adopted for $\leq$ and $-$. Additional symbols are defined by means of the primitive ones $\leq$ and $=$ in the usual way:

$$
\begin{aligned}
x \neq y &:= \neg(x = y) \\
x < y &:= x \leq y \wedge x \neq y \\
x \geq y &:= y \leq x \\
x > y &:= y < x
\end{aligned}
$$

In that we call the "original" temporal constraint problem ([13]) variables range over a *continuous* domain, so $\neq$ is not allowed to logically formalize constraints; this formalization has been further extended in [30] allowing constraints like $x \neq r$ (unary) and $x - y \neq r$ (binary), x, y variables and r a rational\real number, or their disjunctions; which implies that we may have as constraint a finite union of almost–convex sets like $[l, r] - \{r_1, \ldots, r_n\}$.

2.1.2 The original Temporal Constraint Problem

In [13], the general *Temporal Constraint Problem* (TCP) is so formalized:

constraint variables: a finite number of variables ranging over real points;

binary constraints: each of them is the set of solutions of

$$l_1 \leq x_j - x_i \leq r_1 \vee \ldots \vee l_n \leq x_j - x_i \leq r_n \tag{2.1}$$

where all real intervals $[l_1, r_1], \ldots, [l_n, r_n]$ are pairwise disjoint.

So a TC is *explicitly* given as an interval set $I_1 \cup \ldots \cup I_n$, where $I_i := [l_i, r_i]$. Unary constraints on variables are represented as binary ones introducing a new fresh variable, x_0, whose domain is always reduced to a real number, usually 0: that is the constraint $l \leq x_i \leq r$ is expressed as the binary constraint $[l, r]$ on the difference $x_i - x_0$.

Example 2.1 *It is evening; Paulo and Nikos decide to eat a pizza together. Paulo needs at least* $30 - 40$ *minutes to reach the pizzeria. Nikos gets there by bike; depending on his mood, it takes him either* $10-20$ *or* $30-40$ *minutes. Nikos leaves the office between* $7:20$ *and* $7:30$*; Paulo leaves home between* $7:00$ *and* $7:10$*. We wish to answer queries like "Is the whole story consistent?" or "What are the possible time at which they can meet together at the pizzeria, if any?". We can associate the event "Paulo leaves his home" to* x_1*, "Paulo arrives at the pizzeria" to* x_2*, so that we have the binary constraint* $(x_2 - x_1) \in [30, 40]$*. The event "Nikos leaves the office" is associated with the variable* x_3*; introducing a fresh variable* x_0 *to represent the starting point of time and assuming its domain is reduced to* $\{7\}$*, we get, for instance, the constraints* $(x_3 - x_0) \in [20, 30]$ *and* $(x_1 - x_0) \in [0, 10]$.

Definition 2.0.10 *Given two interval sets,* $T := I_1 \cup \ldots \cup I_n$ *and* $S := J_1 \cup \ldots \cup J_m$*, corresponding to either unary or binary constraints, the binary operations of* union, intersection *and* composition *are defined in the following way.*

- *The union of the constraints* T *and* S *is their set–union* $T \cup S$*.*
- *The intersection of the constraints* T *and* S *is their set–intersection* $T \cap S$*.*
- *The composition of the constraints* T *and* S*, written as* $T \odot S$*, is the set of numbers* r *such that there are* $t \in T$ *and* $s \in S$ *so that* $r = t + s$*.*

The composition operation might result in constraints that are not pairwise disjoint; therefore additional process might be required in order to re–establish this condition.

Definition 2.0.11 *Given TCP1 and TCP2 on the same set of variables, we can define their* intersection *TCP1* $\cap$ *TCP2,* union *TCP1* $\cup$ *TCP2 and* composition *TCP1* $\odot$ *TCP2 by means of their constraints:*

> *for any couple of variables* (x_i, x_j), C^1_{ij} *of TCP1 and* C^2_{ij} *of TCP2,* $C^1_{ij} \cup C^2_{ij}$ *is the relative constraint on the same couple of variables of TCP1* $\cap$ *TCP2; if* C^1_{ij} *or* C^2_{ij} *are missing, they are set to* $(-\infty, \infty)$*;*
>
> *for any couple of variables* (x_i, x_j), C^1_{ij} *of TCP1 and* C^2_{ij} *of TCP2,* $C^1_{ij} \cap C^2_{ij}$ *is the relative constraint on the same couple of variables of TCP1* $\cap$ *TCP2; if* C^1_{ij} *or* C^2_{ij} *are missing, they are set to* $(-\infty, \infty)$*;*
>
> *for any couple of variables* (x_i, x_j), C^1_{ij} *of TCP1 and* C^2_{ij} *of TCP2,* $C^1_{ij} \odot C^2_{ij}$ *is the relative constraint on the same couple of variables of TCP1* $\odot$ *TCP2; if* C^1_{ij} *or* C^2_{ij} *are missing, they are set to* $(-\infty, \infty)$*.*

A TCP is represented by means of its associated *directed graph*, in which nodes stay for variables and labels on nodes represent the constraints specified by the problem.

The natural *relation of order* between constraints is the one of set inclusion: $TC1 \subseteq TC2$ iff for every interval of $TC1$ there is one in $TC2$ which includes it. An *order relation between TCPs* having the same set of variables is thereby introduced.

Definition 2.0.12 *Let* $TCP1$ *and* $TCP2$ *be on the same set of variables;* $TCP1$ *is* included *in* $TCP2$*, briefly* $TCP1 \subseteq TCP2$*, iff for any constraint* $TC1$ *of* $TCP1$*, taken the corresponding (i.e. on the same variables) one* $TC2$ *of* $TCP2$*,* $TC1 \subseteq TC2$*.*

Given a fixed set of variables, it is now possible to divide the constraints on these variables into equivalence classes, so that two constraints are *equivalent*[1] iff they have the same set of solutions; every such a class is totally ordered by means of constraint–inclusion (cfr. definition 2.0.12) and so there *exists* a minimal constraint problem in every equivalence class wrt inclusion[2]; as equivalent constraint problems are closed under intersection, this minimal constraint problem is *unique*.

Proposition 2.0.2 *For every constraint problem TCP there exists a unique minimal constraint problem equivalent to it.*

We will call it *the minimal constraint problem of TCP* and we will write $m(TCP)$. Reasoning about temporal constraints there are two main goals to achieve:

1. checking consistency and\or finding a solution to the problem;
2. deriving new constraints from the given ones, which amounts to computing the minimal constraint problem of the given one, that is the whole set of solutions.

If one solves the second problem in polynomial time, then the first one can be solved in polynomial time as well; unfortunately, with general TCP, the first task turns out to be already an NP–hard problem.

Theorem 2.1 *The satisfiability problem for the general TCP is NP–hard.*

pf. By reduction from the 3–coloring problem, cfr. [13].

Since this result, it is worth finding subclasses of TCP for which the problem of satisfiability and of computing the minimal constraint problem can be solved in polynomial time.

The Simple Temporal Constraint Problem

A *simple temporal constraint problem*, STCP, is a TCP whose binary constraints reduce to an interval. The notion of a *distance graph* is thereby introduced: it has the same nodes as the directed graph but labels report as weight r_{ij}. Each

[1] Cfr. definition 1.0.4.

[2] It is immediate to see that a constraint problem, equivalent to the given one, is minimal according to definition 1.0.6 iff it is minimal wrt inclusion.

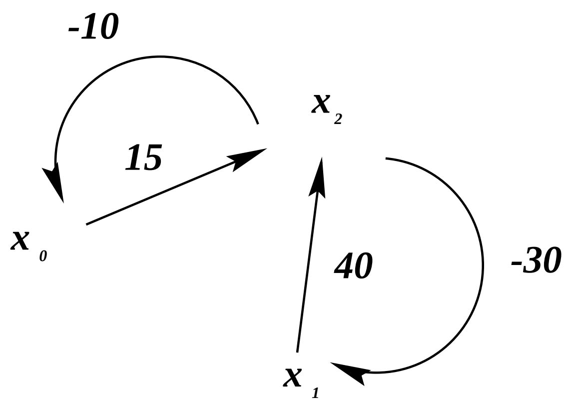

Figure 2.1: A distance graph representing a STCP of example 2.1.

"$(k+1)$–path" $(x_i, \ldots, x_j)$ from i to j induces an eventually new constraint on x_i and x_j, namely $(x_j - x_i) \leq \sum_{n=1}^{k} r_{n_{l-1},n_l}$, where, for each l, $(x_{n_l} - x_{n_{l-1}}) \leq r_{n_{l-1},n_l}$ and $x_{n_0} = x_i$, $x_{n_{k+1}} = x_k$; the intersection of all these path constraints yelds the constraint

$$(x_j - x_i) \leq d_{ij}$$

in which d_{ij} is the length of the "shortest" path from i to j; if there is no such one, then d_{ij} is ∞ and is not usually reported.

Example 2.2 *In figure 2.1, we have represented the distance graph of a STC subproblem of the TCP in example 2.1; the distance graph has the following labels.*

$$\begin{cases} d_{01} = -15 \\ d_{02} = 15 \\ d_{12} = 40 \\ d_{10} = 30 \\ d_{20} = 0 \\ d_{21} = -30 \end{cases}$$

Starting from the distance graph of a given STP, *Floyd–Warshall*'s all–pair–shortest–paths algorithm[3] produces the *d–graph* of this STP, namely the complete directed graph having the same nodes as G_d and edges labeled by the shortest path between i and j; this algorithm runs in $O(n^3)$ time, where n is the number of variables of the problem; if there are no negative cycles, then

[3]Cfr. table 2.1.

for $i = 1, \ldots, n$ **do** $d_{ii} \leftarrow 0$
for $i,\, j = 1, \ldots, n$ **do** $d_{ij} \leftarrow r_{ij}$
for $k = 1, \ldots, n$ **do**
    for $i,\, j = 1, \ldots, n$ **do**
        $d_{ij} \leftarrow \min(d_{ij}, d_{ik} + d_{kj})$

Table 2.1: Floyd–Warshall's algorithm

we can use Dijkstra's algorithm, which runs in $O(n^2)$ time, once for each vertex of the graph; this is a well known problem in the literature about linear programming, cfr. [12].

The main theorems concerning STPs are the following ones.

Lemma 2.1.1 (Consistency–check) *Let STP be a constraint problem: it is consistent iff its associated distance graph has no negative cycles.*

pf. Cfr [12].

Theorem 2.2 (Solution–search) *Let STP be a constraint problem on n variables; for any $k < n$, any k instantiation, consistent wrt the shortest path constraints induced by the associated d–graph, can be extended to a solution to the given STP.*

pf. Cfr [12].

As an important consequence of theorem 2.2 we get the following statement.

Theorem 2.3 *Given a consistent $STCP$, the equivalent one induced by its distance graph is the minimal $STCP$ equivalent to it.*

pf. Cfr [12].

This way we gain an effective procedure to check consistency and construct a solution to a given STCP: see table 2.2.

So the tasks of checking consistency or finding a solution and of building up the minimal constraint problem take a time in $O(n^3)$.

There is another procedure to find the minimal constraint problem equivalent to the input STCP; it is obtained by applying path–consistency; an algorithm to get path–consistency is shown in table 1.2. As usual, whenever there are no constraints on (x_i, x_j), we set the constraint P_{ij} to be $(-\infty, \infty)$.

1.	apply *Floyd–Warshall*'s all–pair–shortest–paths algorithm[4] to produce the *d–graph* of this STP starting from its G_d; it runs in $O(n^3)$ checking consistency by inspecting the signs of the diagonal elements d_{ij};
2.	assemble a solution "only" assigning to each variable any value which satisfies the d–graph constraints relative to the assignments stated for the previously instantiated variables; this process takes a time in $O(n^2)$, because once a value is assigned to a variable it remains unaltered.

Table 2.2: Finding a solution for STCPs

The following lemma ensures that indeed, iterating the relaxation operation in step 3, we get a path–consistent problem.

Lemma 2.3.1 *A constraint PC_{ij} is path–consistent iff it is a subset of the set $\cap_k(PC_{ik} \odot PC_{kj})$.*

pf. Let PC_{ij} be path–consistent, that is 3–consistent[5]; take any $a_{ij} \le r_{ij}$, that is any instantiation I so that $I(x_j) - I(x_i) = a_{ij} \le r_{ij}$; by 3–consistency I can be extended to "any" x_k so that $I(x_k) - I(x_i) = a_{ik} \le r_{ik}$ and $I(x_j) - I(x_k) = a_{kj} \le r_{kj}$; this implies

$$a_{ij} = I(x_j) - I(x_i) \; = \; I(x_k) - I(x_i) + I(x_j) - I(x_k) = a_{ik} + a_{kj}$$

and so $a_{ij} \in PC_{ik} \odot PC_{kj}$.
Let's now suppose that, for "any" x_k, $PC_{ij} \subseteq PC_{ik} \odot PC_{kj}$; let's take any instantiation I of x_i and x_j so that $a_{ij} = I(x_j) - I(x_i) \le r_{ij}$; by hypothesis, for any x_k there are $a_{ik} \le r_{ik}$ and $a_{kj} \le r_{kj}$ such that

$$I(x_j) - I(x_i) = a_{ij} = a_{ik} + a_{kj} \tag{2.2}$$

Just choose $I(x_k)$ so that the following two conditions hold.

$$\begin{aligned} I(x_k) &= I(x_i) + a_{ik} \\ I(x_k) &= I(x_j) - a_{kj} \end{aligned}$$

There is a solution because of 2.2.

Proposition 2.3.1 *Let S a STCP; the path–consistency algorithm produces $m(S)$.*

[5]Cfr. theorem 1.0.1.

pf. Since the *relaxation* step in the path–consistency algorithm[6] amounts to two operations of updating the shortest path–length in Floyd–Warshall's algorithm, applying the first algorithm is equivalent to applying the second one.

STCP augmented with strict inequalities

In [22], Gerevini and Cristani consider the case when strict inequalities are explicitly introduced in constraints: this means that a constraint can be given as a closed, a semi–open or an open interval. Let's write STCP$^<$ for this new kind of simple temporal constraint problem. The binary operations between constraints as well the relation of order are naturally extended; so the notion of minimal equivalent problem.

A weaker version of theorem 2.1.1 still holds as stated below. First we need to modify the definition of *distance graph*[7], arcs are labeled in this new way: if the constraint on (x, y) is $y - x \leq n$, then the label will be $(n, 1)$; if the constraint is a strict inequality then the label will be $(n, 0)$. The shortest distance and path between two points are computed using the following definitions for comparison and addition:

$$(m, x) < (n, y) \text{ iff } m < n \lor (m = n \land x < y);$$

$$(m, x) + (n, y) = (m + n, \min(x, y)).$$

In the resulting d–graph an arc appears between every pair of nodes and the inequalities corresponding to the arcs give the minimal constraint problem equivalent to the input one. This way we get the following two results in the same way as we obtained the similar ones for STCPs without strict inequalities.

Lemma 2.3.2 *Let S be a STCP$^<$: if its associated distance graph has no negative cycles, then it is consistent.*

Theorem 2.4 *Let S be a STCP$^<$: the equivalent one induced by its distance graph is the minimal $STCP$ equivalent to it.*

The algorithm proposed by Gerevini and Cristani to find a solution in this framework performs the following steps, the input being a STCP S and the output a solution, if it exists, `nil` otherwise.

[6] Cfr. line 3 in table 1.2.
[7] Cfr. [28].

1. Check the consistency of S by the criterium of lemma 2.3.2; if it is not consistent, then return `nil` and exit;
2. relax every strict inequalities of S, non involving ∞, to a non–strict one; call S' the resulting temporal constraint problem;
3. compute $m(S')$, by means of the algorithm in table 2.2; call it M;
4. for each left–open interval (label of the graph) bounded from the left by a, replace it with the left–closed interval bounded from the left by $b + (\delta/(n^2 + 1))$, where δ is either the finite length of the shortest interval constraint of M or any finite number if every interval constraint of M has ∞ as one of its bounds; call it M';
5. compute $m(M')$ and so a solution by means of the algorithm in table 2.2.

The following theorem also states that indeed this algorithm is sound and complete.

Theorem 2.5 *The above algorithm computes a solution for a given STCP augmented with strict inequalities, whenever it exists; otherwise it returns* `nil`*. It takes a time in $O(n^3)$ where n is the number of variables involved in the problem.*

pf. See p. 1463 of [22].

STCPs augmented with inequations

In [30], the original STCP framework is extended with inequations and their disjunctions. Constraints can be implicitly defined in the way below.

1. Unary constraints are conjunctions of a formula as $x_j \leq d_{0j}$ or $x_j \geq -d_{j0}$ and of inequation formulae as $x \neq r^1_{ji}, \ldots, x \neq r^{h_{ji}}_{ji}$, where we require that the following condition holds.

$$-d_{j0} < r^1_{ji} < \ldots < r^{h_{ji}}_{ji} < d_{0j} \tag{2.3}$$

   As usual, by introducing a new fresh variable x_0, any unary constraint can be translated into an equivalent binary one.

2. Binary constraints are conjunctions of a formula as $(x_j - x_i) \leq d_{ij}$ or $(x_j - x_i) \geq -d_{ji}$ and of inequations as $(x_j - x_i) \neq r^1_{ji}, \ldots, (x_j - x_i) \neq r^{h_{ji}}_{ji}$, where we require that condition 2.4 holds.

$$-d_{ji} < r^1_{ji} < \ldots < r^{h_{ji}}_{ji} < d_{ij} \tag{2.4}$$

3. Finally we define d–ary constraints, which we will briefly call d–constraints: they are disjunctions of inequations involving d distinct variables.

These constraints can be set theoretically defined in the obvious way; in particular binary constraints can be explicitly given by *almost–convex intervals* like $I := [-d_{ji}, d_{ij}] - \{r^1_{ji}, \ldots, r^{h_{ji}}_{ji}\}$; $conv(I)$ will stay for the *convex hull* of I[8].

Disjunctions of inequations have been introduced in [30]: the motivation behind this choice is that, eliminating variables from a set of temporal constraints, an inequation may give rise to a disjunction of inequations.

Starting with a STCP with inequations but *without* d–constraints, let's say P_1, replacing each almost–convex interval I with its convex hull $conv(I)$, we get a STCP, P_2, to which path–consistency algorithm can be successfully applied to check satisfiability and find (all) solutions[9]. In [31] Koubarakis proves the following important results.

Theorem 2.6 *Enforcing 5–consistency on a STCP with inequations but without d–constraints is necessary and sufficient to get global consistency; the algorithm runs in time $O(kn^4)$, where k is the number of inequations and n that of variables.*

pf. Cfr. [31].

Modifying step 2 of this algorithm, Koubarakis designs an algorithm to produce the minimal (equivalent) constraint problem of a STCP with inequations without d–constraints, which runs in $O(\max(kn^2, n^3))$ time.

The algorithm to enforce global consistency in the case of STCPs with d–constraints is a generalization of the one to gain 5–consistency: instead of enforcing 5–consistency, it enforces $(2V+1)$–consistency, where V is the maximum number of variables in any disjunctions of inequations. It is exponential in V, but if this number is fixed, then the time complexity of this algorithm is polynomial in the number of variables and of constraints.

[8] The class of almost–convex intervals is closed under intersection and composition of constraints as given in definition 2.0.11.

[9] This procedure is sound because path–consistency is complete for simple temporal constraint problems, cfr. theorem 2.3.1.

The Simple Temporal Constraint Problem augmented with inequations and inequalities

We can go further on and take into consideration the case when constraints are almost–convex open, semi–open or closed intervals; this means that they can be implicitly given by formulae involving $<$, $\leq$ or $\neq$.

- A unary constraint can be implicitly given by a formula either as $x \leq r \vee (x \neq r_1 \wedge \ldots \wedge x \neq r_n)$, as $r \leq x \vee (x \neq r_1 \wedge \ldots \wedge x \neq r_n)$, as $x < r \vee (x \neq r_1 \wedge \ldots \wedge x \neq r_n)$ or as $r < x \vee (x \neq r_1 \wedge \ldots \wedge x \neq r_n)$.
- A binary constraint can be formalized by a conjunction of either $x_j - x_i \leq d_{ij}$, $x_j - x_i \leq -d_{ji}$, $x_j - x_i < d_{ij}$ or $x_j - x_i < -d_{ji}$ and of a formula as $(x_j - x_i \neq r_{ij}^1 \wedge \ldots \wedge x_j - x_i \neq r_{ij}^{h_{ij}})$.

The algorithm, briefly sketched below, receives as input a STCP S so augmented and returns `true` if S is consistent, `nil` otherwise:

1. substitute every non convex constraint by its convex hull; call S' the resulting constraint problem;
2. compute the d–graph of S'; call it D;
3. if D containts negative cycles, then return `nil`;
4. else, for each inequation $x_j - x_i \neq r$ in S, do: if the label from x_i to x_j in D is d and that from x_j to x_i in D is $-d$, then return `nil`; else return `true`.

In order to prove that this algorithm is sound and complete, we need the following lemma; if S is a simple temporal constraint augmented with disjunctions of inequations, let's call *relaxation* of S, writing $conv(S)$, the constraint problem obtained by replacing each constraint which is a non convex interval by its convex hull.

Lemma 2.6.1 *A simple temporal constraint problem S augmented with disjunctions of inequations and strict inequalities is consistent if the distance graph of $conv(S)$ does not have negative cycles and $conv(S)$ does not entail $x_j - x_i \neq d$ whenever $x_j - x_i = d$ is among the constraints of S.*

pf. Cfr. [22] pg. 1464.

Theorem 2.7 *If the input S is consistent then the algorithm sketched above detects it, otherwise it returns inconsistency; it runs in $O(n^3 + k)$ time, where n is the number of variables and k is that of inequations.*

pf. Cfr. [22] pg. 1465.

The following algorithm finds a solution, if it exists, taking as input a STCP S augmented with disjunctions of inequations and inequalities; if it does not exists, then it returns `nil`:

1. check the consistency of S by means of the previous algorithm; if it is not consistent, then exit and return `nil`;
2. compute $m(conv(S))$;
3. add to $m(conv(S))$ the input inequations $x_j - x_i \neq d$ such that d is a boundary of the constraint of $m(conv(S))$ on (x_i, x_j); call S'' the resulting constraint problem;
4. compute $m(S'')$;
5. add to $m(S'')$ the input inequations $x_j - x_i \neq d$ such that d is *not* a boundary of the constraint of $m(conv(S))$ on (x_i, x_j); call M the resulting constraint problem;
6. for each left–open interval (label of the graph) bounded from the left by b, replace it with the left–closed interval bounded from the left by $b + (\delta/(n^2 + 1))$, where δ is so defined:
   - $\min(\delta_{ij})$ if at least one interval has either finite bounds or the lower bound is finite and the interval is not convex;
   - any finite number otherwise;

   $i, j = 1, \ldots, n$, $i \neq j$, and δ_{ij} is the lenght of the first convex subinterval of the possibly non convex intervals of M; call M' the resulting constraint;
7. omit all the non convex inequations from M'; call M'' the minimal constraint problem of the constraint problem so obtained;
8. compute a solution by means of algorithm in table 2.2.

Theorem 2.8 *The previous algorithm is sound and complete; it runs in $O(n^3 + k)$ time.*

pf. Similar to that of theorem 2.7.

The General Temporal Constraint Problem

Since theorem 2.1, different strategies to approach the general problem have been proposed. The main approaches can be so classified:

i. *splitting* the TCP into simple temporal constraint problems tractable in polynomial time;

ii. backtracking search tecniques[10];

iii. *local consistency* pruning algorithms: path–consistency, 5–consistency etc.

i. Splitting

A constraint in a TCP is a disjunction of simple temporal constraint problems; selecting a disjunct from each constraint of TCP, we get a single STCP that can be solved by means of one of the appropriate algorithm given above, depending on the kind of simple temporal constraints we choose to deal with. Let's call *labeling* a selection of one disjunct from each constraint: so there is a solution of the given TCP iff there exists a labeling whose associated STCP is consistent (any solution of the given TCP is a solution of at least one of the STCPs generated by means of labelings and a solution of any of these STCPs is a solution of the given TCP). Moreover, by definition, it is immediate to prove that the minimal network of a given TCP is the union of M_l, the minimal networks of the STCPs given by means of all the possible labelings of TCP. The algorithm given above can be improved by a backtracking search.

iii. Path–consistency and its improvements

The path–consistency algorithm[11] gives the minimal constraint problem equivalent to the given one if we deal with STCPs or STCPs$^<$, as we saw in proposition 2.3.1. In the general case, this does not hold: take, for example, the constraint problem on three variables, x, y, z, and temporal constraints $C_x = [0, 1] \cup [10, 20]$, $C_{xy} = [10, 20]$, $C_{yz} = [0, 20] \cup [40]$, $C_{xz} = [25, 50]$.

As Schwalb and Dechter observe in [45], path–consistency is achieved by means of a relaxation operation, namely $C_{ij} \leftarrow C_{ij} \cap (C_{ik} \odot C_{kj})$ which may increase the number of intervals of the associated constraint problem; this means that the number of intervals in the resultant path–consistent TCP may be exponential in the number of intervals per constraint in the input problem; this is known in the literature as the *fragmentation* problem ([45]).

In [45], they propose an algorithm, called ULT, that, used before performing path–consistency, can reduce the number of intervals; it relies on the fact that

[10] Cfr. [29].
[11] Cfr. table 1.2: for improved versions of this algorithm, see section 6 of [13].

path–consistency is enough to gain consistency for STCPs or STCPs$^<$.

The key–idea is quite simple: a generic metric constraint P^1 is expressed as a disjunction of intervals $I_1 \cup \ldots \cup I_n$; the lowest and the greatest extreme points among those of these intervals are selected in order to define a STCP called P^2; path–consistency algorithms can be applied to this constraint problem to get its equivalent minimal constraint problem P^2 in time $O(n^3 R^2)$, where n is the number of variables and R is the range of the constraints[12]. Then a constraint problem P^3 equivalent to P^1 is obtained intersecting the corresponding constraints of P^1 and of P^3.

Algorithm ULT runs in $O(n^3 ek + e^2 k^2)$ time, where n is the number of variables, e is the number of edges and k is the maximal number of intervals in each constraint.

They call a constraint C_{ij} *redundant–prone* if, after running ULT, the resultant constraint C^3_{ij} is not path–consistent yet. As they prove, if $C^2_{ij} = \cap_k (C^2_{ik} \odot C^2_{kj})$, then C^2_{ij} is redundant–prone; after applying ULT to a constraint problem P_1, one can check if this condition is fulfilled in order to remove some constraints which are not path–consistent yet. In a subsequent paper, [46], they improve ULT, introducing a new algorithm, called LPC, and some of its variants to better approximate path–consistency simply modifying the intersection operation.

Definition 2.8.1 *If* $T := I_1 \cup \ldots \cup I_n$ *and* $S := J_1 \cup \ldots \cup J_m$ *are two constraints, their* loose intersection, $T \overset{loose}{\cap} S$, *is the set of intervals* $\{L_1, \ldots, L_n\}$ *so defined: for every* $i := 1, \ldots, n$, $L_i = [l_i, u_i]$ *where* l_i *and* u_i *are respectively the lower and upper bound of the interval set* $I_i \cap S$.

Remark 2.1 *Since* $\#(T_{ij} \overset{loose}{\cap} S) \leq \#T_{ij}$, *the number of intervals of* T_{ij} *is not increased during the relaxation* $T_{ij} \overset{loose}{\cap} (T_{ik} \odot T_{kj})$.
By definition 2.8.1, it is always the case that

$$T_{ij} \cap S \subseteq T_{ij} \overset{loose}{\cap} S \subseteq T_{ij}$$

and so, in particular, it is true when $S = T_{ik} \odot T_{kj}$.
The operation of loose intersection is not commutative: for instance, if $T := \{[0,2], [1,4]\}$ *and* $S := \{[0,3]\}$, *then* $T \overset{loose}{\cap} S$ *is the set* $\{[0,2], [1,3]\}$, *while* $S \overset{loose}{\cap} T$ *is the set* $\{[0,3]\}$. *However, contrary to what it is remarked in [46], it may be that* $T \overset{loose}{\cap} S = T \overset{loose}{\cap} S$; *take for instance* $S = \{[0,2]\}$ *and* $T = \{[1,3]\}$; *then* $T \overset{loose}{\cap} S = \{[1,2]\}$ *and so it is* $S \overset{loose}{\cap} T$.

[12]The range of the constraints is the difference between the lowest and the highest numbers specified, [13].

1. **input** P^1
2. $P^3 \leftarrow P^1$
3. **repeat**
4. $\quad P^1 \leftarrow P^3$
5. $\quad$ compute P^2, P^3.
6. **until**
   $\exists\, i, j\, (T^3_{ij} = \emptyset)$
   or $\forall\, i, j\; \#T^3_{ij} = \#T^1_{ij}$
7. **if**
8. $\quad$ **then return** "inconsistent"
9. $\quad$ **else return** P^3

Table 2.3: LPT algorithm

Replacing the intersection operation with the loose intersection operation, the fragmentation problem disappears. A sketch of the algorithm LPC is given in table 2.3:

- P^2 is obtained from P^1 by $T^2_{ij} := \cap_k (T^1_{ik} \odot T^1_{kj})$;
- P^3 is derived from P^2 by $T^3_{ij} := T^1_{ij} \overset{loose}{\cap} T^2_{ij}$.

As P^2 is, by construction, equivalent to P^1 (cfr. lemma 2.3.1), and the loose intersection operation does not introduce new solutions and preserves old ones, we get the following lemma.

Lemma 2.8.1 *The input and the output constraint problem of LPT are equivalent; moreover every iteration of this algorithm removes* at least *one interval from some of the constraints.*

pf. The first claim follows immediately from the previous remark. Since step 6 of LPT, if this algorithm does not remove any constraint then it stops.

Theorem 2.9 *Algorithm LPT takes a time in $O(n^3k^2e)$, where n is the number of variables, e is the number of constraints and k is the maximal number of intervals in each constraint.*

In [46] they refine also algorithms to approximate path–consistency, simply substituting intersection by loose intersection.

If LPT and ULT are not able to find a solution, they are useful when propagating constraints *during* backtracking search or *before* starting search: their effectiveness lays in the fact that they reduce the number of intervals in the given constraints, otherwise they stop.

2.2 Allen's Interval Algebra

2.2.1 Introduction to Allen's interval algebra

In 1983 Allen's article on Temporal Reasoning appeared under the self–explanatory title *Maintaining Knowledge about Temporal Intervals*: he describes a temporal representation that takes the notion of *temporal interval* as primitive; constraints are therefore represented as relations holding between intervals.

> [...] This representation is designed explicitly to deal with the problem that much of our temporal knowledge is relative, and hence cannot be described by a date (or even a fuzzy date). [...]

Metric TCPs are useful to express metric information; but a statement like "Event E1 and event E2 are disjoint" cannot be expressed by binary metric constraints. Whenever temporal information reduces to qualitative relations between the intervals at which events occur, like "Alessandra was away when Eyal defended his thesis" or "After defending our thesis, we will go to London on holidays", most of applications adopt Allen's interval algebra.

As Allen further arguments in his paper [1], his framework is particularly designed for these reasons:

- it allows "significant imprecision": much temporal knowledge is relative and sometimes it has no relation to absolute dates;
- "uncertainty of information" can be represented by means of disjunctions of relations between two intervals;
- since the qualitative representation of the constraints in this apparatus, one has a certain freedom when modeling knowledge and can choose the grain of reasoning she or he prefers, for instance expressing time in terms of days, weeks or business–days;
- the reasoning machinery allows *default reasoning* of the type "If I parked my car in lot A this morning, then it should still be there now".

But Allen's framework has gained its popularity because it represents a good balance between expressiveness and computational efficiency: it allows disjunctive information but only between pairs of intervals.

> [The temporal representation of Allen] does not insist that all events occur in a known fixed order [...] and it allows disjunctive knowledge, such as that event A occurred either before or after event B [...].

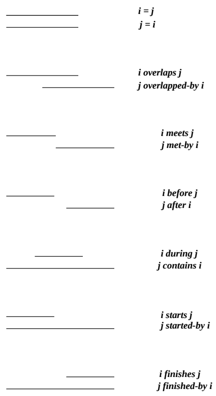

Figure 2.2: Allen relations.

A first order language

In Allen's framework, variables range over real or rational valued intervals. Constraints are specified as unions of *atomic* (*basic*) relations, which are pairwise disjoint: *before*, *starts*, *during*, *overlaps*, *meets*, *finishes* and their converse relations, *after*, $started - by$, *includes*, $overlapped - by$, $met - by$, $finished - by$ plus the equality relation $=$. This way we are not committed with a "particular" representation over a set, that is over rational or real intervals.

The class of all possible unions of the atomic relations forms a boolean algebra; that is Allen's interval algebra, IA: there are 13 atomic relations and so 2^{13} relations in IA.

Allen introduced further operations in his framework[13]; these operations can be generally defined among binary relations over a universe U in the way stated below.

Definition 2.9.1 *Given two binary relations R and S on the same universe U, their* composition*, written as $R \odot S$, is the set of all (x, y) so that there exists z satisfying this condition: $(x, z) \in R$ and $(z, y) \in S$.*
The converse *of a binary relation R is the set of all (x, y) such that $(y, x) \in R$; this new relation is written as R^{-1}.*

Since IA is closed under these operations and contains the equality relation, it is a relation algebra. In [35], Peter B. Ladkin and Alexander Reinefeld describe the framework from Allen as a finite relation algebra[14] this way:

[13] Cfr. [1].
[14] For a complete description, cfr. [33]. In [34] they state that IA does not have a representation over a finite set[15].

1. the *universe* U is a set of atomic relations which correspond to the 13 pairwise disjoint basic relations given by Allen;

2. the *operations* among them are the binary of set union (which corresponds to the logical operator of disjunction $\vee$), intersection (corresponding to $\wedge$) and composition and the unary operation of converse.

A binary constraint on the variables x_i and x_j can be *extensionally* characterized as a (finite) union of the atomic relations, that is

$$B_1 \cup \ldots \cup B_n$$

in which $B_1, \ldots, B_n$ are $n \leq 13$ of the atomic relations, once it is stated if these relations are represented on rational or real numbers. But it is usually *intensionally* described in the following two ways, without committing to any particular representation over any set:

$B_1 \vee \ldots \vee B_n$ as a logic formula,
$\{B_1, \ldots, B_n\}$ in a set theoretic fashion.

We will stick to the second expression, feeling free to choose the other one when useful.

Now we are able to present the temporal constraint problems of Allen in a first order logic framework.

Definition 2.9.2 *A first order language with equality is given, let's call it L_a; it has twelve relation symbols corresponding to the basic Allen relations different from the equality relation; these are the only non logical symbols. We have:*

1. *a finite number of variables ranging over real or rational valued intervals;*

2. *a finite number of binary constraint relations.*

NP–completeness of IA

Checking consistency for IA constraint problems turned out to be NP–hard; to prove this, Vilain and Kautz[16] reduce the 3–clause satisfiabiliy problem to the problem of determining consistency in IA, constructing a "computationally trivial mapping between a formula in 3–SAT form and an equivalent encoding of the formula in the interval algebra".

[16] Cfr. [39].

Theorem 2.10 *Determining the consistency of a subset of IA is NP–hard.*

pf. For every literal A and its negation $\neg A$ in the formula, let's define a couple of intervals iA and $iNegA$. These intervals are then related to a "truth–determining" interval called $middle$: intervals that are before $middle$ correspond to false literals and those falling after correspond to true ones. The original formula can be so encoded, in polynomial time, in IA: for each clause $P \vee Q \vee R$, intervals are created so that at most two of them can be before $middle$ (which makes them false) and the other one\s can fall after $middle$ (which makes them true). Since the original formula has a model iff the interval encoding is satisfiable and 3–SAT is an NP–complete problem, the assertion follows.

2.2.2 Path–consistency and IA

In his paper [1], Allen introduces path–consistency to deal with IA constraint problems; he motivates his choice as follows.

> [...] [Path–consistency] is an attempt to characterize the inferences about time that appear to be made automatically or effortlessly during a dialogue, story comprehension, or simple problem–solving.

The path–consitency algorithm, as we will see later[17], "propagates" relations between intervals, by means of composition, in search of a minimal constraint problem equivalent to the given one; this means that, whenever new relations between two intervals are introduced in the problem, new constraints are added computing the composition of these new relations and so on. Intuitively, the process keeps on in the way described below ([1]).

> When a new interval relation is entered, all concequences are computed.[...] The new fact adds a constraint about how its two intervals could be related, which may in turn introduces new constraints between other intervals through the transitivity rules[18] governing the temporal relationships. For instance, if the fact that i *during* j is added, and j *before* k, then it is inferred that i *before* k. This new fact is added to the network[19] in an identical fashion, possibly introducing further constraints on the relationship between other intervals.

Allen's original path–consistency algorithm represents constraints as queues; we will not use this representation, as we will see on page 36.

[17] Cfr. table 1.2.
[18] E.g.: by computing the composition of these new relations.
[19] E.g.: the constraint problem.

Soundness of the path–consistency algorithm for IA problems

Arc–consistency is computationally cheap, unfortunately it is not a good approximation to consistency in the case of IA problems, as stated in [33]. This is indeed the case of path–consistency, which was used for the first time by Allen[20] to approximate the set of *all* solutions; moreover, it sufficient to guarantee consistency of problems on some particular important subalgebras of IA[21], but it is not enough to guarantee the consistency of IA problems[22].

However there are IA constraint problems which are not path–consistent, which means that path–consistency is a good candidate as a pruning technique in this framework: for instance, consider the constraint problem on three variables, x, y and z, so that the constraint on (x, z) is *before* $\vee$ *after*, it is *before* $\vee$ *meets* on (x, y) and it is *before* $\vee$ *meets* on (y, z); since

$$\begin{aligned}&(\mathit{before} \vee \mathit{meets}) \odot (\mathit{before} \vee \mathit{meets}) = \\ &(\mathit{before} \odot \mathit{before}) \vee (\mathit{before} \odot \mathit{meets}) \vee (\mathit{meets} \odot \mathit{before}) \vee (\mathit{meets} \odot \mathit{meets}) = \\ &\mathit{before}\end{aligned}$$

this constraint problem is not path–consistent (so it is not consistent).

As seen in this case, by means of the composition table and the distributivity of $\odot$ wrt $\cap$ $(\vee)$, one can easily compute the composition between any two of the 2^{13} Allen relations; however use of these distributive laws is very time–consuming and some techniques have been explored to speed up composition, as we will see later. Computing composition is the key step in the process of path–consistency reduction of a TCP, because of the subsequent statement.

Proposition 2.10.1 *A constraint problem is path–consistent iff the following condition holds: for any of its constraints B_{ij}, for any variable x_k, $B_{ij} \subseteq B_{ik} \odot B_{kj}$.*

pf. We saw that path–consistency is equivalent to 3–consistency[23]; that is any consistent instantiation of two variables x_i and x_j can be extended to a consistent instantiation of $\{x_i, x_k, x_j\}$, for any x_k of TCP. In our framework[24] this means exactly that for any $(r_i, r_j) \in B_{ij}$, for any k, there is r_k such that $(r_i, r_k) \in B_{ik}$ and $(r_k, r_j) \in B_{kj}$; by definition of composition, this amounts to saying that $B_{ij} \subseteq B_{ik} \odot B_{kj}$.

From the previous proposition we get immediately the following result.

[20] Cfr. [1].
[21] Cfr. p. 11.
[22] Cfr. [1].
[23] See proposition 1.0.1.
[24] Once we have chosen to represent the Allen relations either on rational or real numbers.

Corollary 2.11 *Searching a path–consistent CP amounts to the search for* the greatest fixed point *of the following set of equations:*

$$\begin{cases} X_{ij} \subseteq B_{ij} \\ X_{ij} = B_{ij} \cap \bigcap_{k \leq n} B_{ik} \odot B_{kj} \end{cases}$$

where B_{ij} is the constraint on (x_i, x_j), for every i, j.

In [36], Ladkin and Reinefeld show how to represent an IA problem as an $n \times n$–matrix; first they reduce a given TCP to an equivalent one such that:

a) for each couple of the n variables (x_i, x_j), there exists and is unique the constraint on them;

b) each one of the n constraints C_{ii} is a subset of the identity relation.

How do we get this result? For every (x_i, x_j) such that there is a constraint on it, let's intersect all the constraints on this couple; whenever $x_i = x_j$, let's intersect all these constraints with the equality relation too; this process leads to a TCP equivalent to the original one. If there are no constraints on (x_i, x_j), $x_i \neq x_j$, just choose any (a_i, a_j) in $U \times U$ and state it as a new constraint between these two variables: if it happens that $x_i = x_j$, let's choose $a_j = a_i$; this way we get a TCP equivalent to the original one and satisfying conditions a and b.

The constraint matrix M of a given TCP[25] is just the $n \times n$ matrix whose entry M_{ij} is the constraint on (x_i, x_j) of the equivalent TCP we get the way described above.

Proposition 2.11.1 *Searching a path–consistent CP amounts to the search for* the greatest fixed point *of the following set of equations:*

$$\begin{cases} X_{ij} \subseteq M_{ij} \\ X_{ij} = M_{ij} \cap \bigcap_{k \leq n} M_{ik} \odot M_{kj} \end{cases}$$

where M is the constraint matrix.

pf. It follows immediately from corollary 2.11.

The previous proposition proves the *soundness* of the algorithm given in table 1.2 to reduce the given TCP to an equivalent path–consistent one.

[25] Cfr. p. 12.

LRC(C)

```
function LRC (var M: matrix; i,j integer): boolean
   N ← M
   for each l_k ∈ M_ij ∩ ℰ do
       M_ij ← l_k
           if LRPC(M) then
               if M_ij is the last one or LRC(M, next(i), next(j)) then
                   return true
       M ← N
   return false
```

Table 2.4: LRC algorithm to check consistency

As Allen's original algorithm, LRPC takes a time in $O(n^3)$, because of compositions, that is the *relaxation* $M \leftarrow M^2$. Some improvements to speed up composition are presented in the following paragraph.

Further Ladkin and Reinefeld introduce an algorithm to check consistency of a TCP, which calls LRPC as a subroutine: see table 2.2.2. We propose to use any subset $\mathcal{E}$ of IA, from which the constraints l_k can be chosen, this way:

1. for $\mathcal{E}$, path–consistency is sufficient to guarantee consistency,
2. intersecting an element of $\mathcal{E}$ with anyone of the Allen relations yields $\emptyset$ or an element of $\mathcal{E}$: that is, if $B \in \mathcal{E}$ and R is anyone of the Allen relations, then $B \cap R \in \mathcal{E}$.

Proposition 2.11.2 *The algorithm in table 2.2.2 is sound whenever $\mathcal{E}$ is a subset of IA satisfying conditions* 1 *and* 2 *above.*

pf. Immediate because of conditions 1 and 2.

Remark 2.2 *Ladkin and Reinefeld suggest $\mathcal{E}$ be the set of atomic relations; the algorithm runs correctly because path–consistency is complete in this case*[26] *and the intersection of any atomic relation with anyone of the 2^{13} Allen relations is still an atom.*

How to speed path–consistency reduction

As already observed, the computation of composition is the main cause for the complexity of path–consistency algorithms. A CP with n variables has

[26] Cfr. [50].

$n \cdot (n-1)/2$ possible constraints; this means that the number of compositions and intersections to perform is $(n-2) \cdot n \cdot (n-1)/2$. Moreover, path–consistency is an iterative process, that usually requires more than one iteration to stabilize. That is why a certain number of techniques to speed up the *triangle operation*

$$M_{ij} \leftarrow M_{ij} \cap (M_{ik} \odot M_{kj})$$

have been developed.

How to compute compositions

To speed up composition, one can compute compositions of non atomic relations by looking up the composition table of atomic relations (Allen's method); if there is enough memory available, the full $2^{13} \times 2^{13}$ relation table can be stored and accessed efficiently; if this is not feasible, one could split it in four (Hogge's method, cfr. [35].) or two tables.

Avoiding useless operations

The path–consistency algorithm in table 1.2 recomputes all labels in every iteration; it would be enough to recompute only the constraints which changed in the previous iteration. Avoiding this step can be done implementing a hashing table which holds all previously computed compositions: before performing a new composition, one simply looks in this table and sees if this computation is available: in this case, the result is taken without performing any new computation; otherwise one can use one of the previous method to compute the new composition[27].

In [56] a series of heuristics to skip useless computations is presented: for instance, if M_{ik} or M_{kj} is the equality relation, the computation can be avoided; if two constraints include *before* and *after*, or *after* and *before* or *during* and *includes*, then the resulting composition is $=$; if the computation to perform would produce a larger constraint than the input one, it can be avoided because it would not constraint its couple of variables further.

2.2.3 Tractable subalgebras of IA

Allen's algebra IA contains $2^{13} = 8192$ relations, this means that there are 2^{8192} subsets in IA and so their *complete* classification is probably not feasible. So research has focused on identifying first *tractable* and recently *maximal tractable* subalgebras of IA, that is algebras which cannot be extended further by means of any relation without loosing tractability.

[27] Cfr. [34].

Some of the most important subalgebras of IA are obtained "translating" qualitative point relations or metric ones into Allen relations; this means that first we will have to introduce other languages to describe some set of qualitative or quantitative relations between points and then we will translate them in subalgebras of IA.

An exaustive search by computers is a key technique to prove the maximality of the algebras that up to now have been discovered; this machine case analysis was firstly introduced by B. Nebel and H.J. Bürckert[28]. A different approach to this problem, in a geometric and not a logic apparatus, is given in Ligozat's work ([38],[37]).

The point algebra

The point algebra (PA) has been one of the first important structures to be studied in the literature of Temporal Constraint Programming: it was introduced by Vilain and Kautz[29] and further research was carried on by van Beek.

PA constraint problems can be defined as follows:

1. variables range over the set of rational (real) numbers and they stand for time points;
2. constraints are disjunctions of binary relations belonging to the set

$$\{<, \leq, =, >, \geq, \neq, ?\}$$

   where $? := \{<, =, >\}$[30]; these are called the *basic* relations.

We will call L_{pa} the language individuated by these relations to distinguish it from L_a.

The operations of intersection, composition and converse between PA relations are computed by means of the basic ones[31].

A restricted set of Allen relations, namely SA, can be translated into PA relations and vice versa, without loss of information[32]. In SA constraint problems, the constraints between two intervals are only those which can be translated

[28] The utility `atry` is available by `ftp` at ....; it generates minimal extensions of subsets by adding a relation and computing the closure of the new set of relations under converse, intersection and composition.

[29] Cfr. [39].

[30] We use ? when we do not have any explicitly stated constraint between two variables.

[31] Cfr [52].

[32] Cfr. [52].

into conjunctions of PA relations among the endpoints of these two intervals. A lot of applications of IA problems in the literature actually use only SA relations: for instance, in representing temporal information in medical expert systems, Hamlet and Hunter [26] adopt only relations of SA except the disjointness relation. In fact the expressive power of SA is limited by the fact that the "disjointness" of intervals cannot be translated into PA constraints; take for instance the IA relation $\{before, after\}$, which requires disjunction of conjunctions of PA relations among endpoints.

However SA turns out to be expressive enough for many practical tasks and can be used to approximate solutions for IA constraint problems. Let's examine PA.

In [53], van Beek shows how to transform constraints involving only the relations $<, \leq, =, >, \geq$ into simple temporal metric constraints (STCs) this way:

$$\begin{array}{lcl} x_i = x_j & \rightarrow & 0 \leq x_j - x_i \leq 0 \\ x_i \leq x_j & \rightarrow & 0 \leq x_j - x_i < \infty \\ x_i < x_j & \rightarrow & -\epsilon \leq x_j - x_i < \infty \end{array}$$

Note that the positive real number ϵ allows to transform the PA problem in one using only $\leq, =, \geq$, ruling out $<$; for instance we could choose $\epsilon = (1/(n^2 + 1))$, where n is the number of variables of the constraint problem.

This transformation shows that: if the set of allowed relations is a subset of $\{\leq, =, \geq\}$, then there are not negative cycles,[33] therefore one can use Dijkstra's algorithm for each of the n variables to find a consistent instantiation of the constraint problem; if the set of allowed relations include also $<$ or $>$, then there might be negative cycles and so one can use Floyd–Warshall's algorithm which takes a time in $O(n^3)$ to produce a consistent instantiation.[34]

The constraint relation $x_i \neq x_j$ cannot be transformed into STCs, because we need disjunction to express it: in fact it is (logically) equivalent to $x_i < x_j \vee x_j < x_i$ and so it is transformed into $-\epsilon \leq x_j - x_i < \infty \vee -\epsilon \leq x_i - x_j < \infty$.

This is only one possible technique to solve PA constraints, going back to Dechter, Meiri and Pearl's paper [13]; another one is that given by Ladkin and Maddux[35] whose algorithm to find a consistent instantiation runs in $O(n^3)$ time, reducing the problem to a path–consistent one; in [53], van Beek proposes an algorithm, called CSPAN, which runs in $O(n^2)$ time.

This algorithm applies topological sort; this requires we rule out $\leq$, $=$, $\neq$ and $\geq$, dealing only with $<$ or $>$. The input of CSPAN is an adjacency matrix C, of which the elements C_{ij} are the constraint relations\labels $< i, j >$ of

[33] Cfr. [12].
[34] Cfr. [12].
[35] Cfr. [33].

CSPAN(C)

1. Tarjan's algorithm to get the strongly connected components $S_1, \ldots, S_m$
2. **for** $i, j \leftarrow 1, \ldots, m$
3. **do** $C_{S_i S_j} \leftarrow \{<, =, >\}$
4. **for** each $x \in S_i$, $y \in S_j$
5. **do** $C_{S_i S_j} \leftarrow C_{S_i S_j} \cap C_{xy}$
6. **if** $C_{S_i S_j} = \emptyset$
7. **then return** ("Inconsistent network")
8. Replace any remaining $\{<, =\}$ with $\{<\}$
9. Topological sort using only the constraints (edges) involving $\{<\}$

Table 2.5: CSPAN algorithm to find a consistent instantiation for PA

the associated d–graph; first he "condences" the given PA into an equivalent one substituting the verteces x_i with the classes S_i of all vertices which are "constrained" to be equal[36] and the new constraints $C_{S_i S_j}$ are the intersections $\bigcap_{v \in S_i\ w \in S_j} C_{v\,w}$. While creating these equivalence classes, the algorithm detects inconsistencies, if any; first replacing $\leq$ with $<$ and $\geq$ with $>$ (this process yelds to an equivalent TCP because $=$ and $\emptyset$ have already been removed), then performing topological sort, he gets a solution iff the original TCP had one. The relation $\neq$ is handled implicilty: because of the previous steps of the algorithm, there are only distinct time points. These facts yeld the following result; for a more detailed proof of soundness, see [53].

Proposition 2.11.3 *The algorithm in table 2.2.3 is sound and complete: that is, given as input a PA constraint problem which is consistent, it produces a solution; otherwise it detects inconsistency. It takes a time in $O(n^2)$.*

pf. Because Tarjan's algorithm runs in $O(n^2)$ time and so do the tasks of condensing the constraints and of topoligical sort, the given algorithm takes a time in $O(n^2)$.

Further van Beek gives an algorithm, called FEASIBLE (cfr. table 2.2.3), able to find all solutions computing the minimal constraint problem equivalent to the input one, which runs in $O(max(mn^2, n^3))$, where n is the number of vertices of the graph and m is the number of pairs of points which are in the relation $\neq$. So first this algorithm prunes the search space by means of path–consistency[37] and then it looks for what van Beek calls "the forbidden subgraph" (cfr. [52], [53]).

[36] Producing these equivalence classes is proved to be equivalent to indentifying the strongly connected components of a graph, as van Beek shows in [52]; one may use Tarjan's algorithm to perform the last task because this algorithm takes a time in $O(n^2)$.

[37] We can use the algorithm given in table 1.2, now relations being PA ones.

FEASIBLE(C)
1. PATH–CONSISTENCY(C)
2. FIND–SUBGRAPHS(C)

FIND–SUBGRAPHS(C)

1. **for** each (x, y) st $y \in adj_{\neq}(x)$ **do**
2. $\quad S \leftarrow (adj_{\geq}(x) \cap adj_{\geq}(y))$
3. $\quad T \leftarrow (adj_{\leq}(x) \cap adj_{\leq}(y))$
4. $\quad$ **for** each $s \in S$, $t \in T$ **do**
5. $\qquad C_{st} \leftarrow \{<\}$
6. $\qquad C_{ts} \leftarrow \{>\}$

Table 2.6: FEASIBLE algorithm to find *all* consistent instantiations of PA

Remark 2.3 *As we have previously observed, path–consistency is complete for constraints of atomic relations*[38]*; further, it has been proved that it is also complete for PA constraints problems without $\neq$, since they can be translated into STCPs. The importance of the algorithm FEASIBLE lies in this fact: path–consistency is* not *sufficient for finding the minimal constraint problem if PA constraints involve $\neq$*[39].

As van Beek suggests in [53], one can solve an SA problem translating it into PA ones, solving them and then taking the union of solutions; the algorithm performs well if the number of points said to be unequal is minimal.

The NB algebra

In [43], Nebel and Bürckert have singled out the maximal subalgebra of IA containing all Allen basic relations, called NB, of which the satisfiability can be decided in polynomial time.

A *time interval* is defined to be a formula as Bx_ix_j, where B stays for one of the basic Allen relations; as usual, we will write it as x_iBx_j. A formula (positive clause) of the form $x_iB_1x_j \vee \ldots \vee x_iB_nx_j$ is called *interval formula*; we will briefly write it as $x_i\{B_1, \ldots, B_n\}x_j$.

In any *interval–interpretation* I, briefly I–interpretation, the B_j are always interpreted as Allen relations, while $I(x_i)$ and $I(x_j)$ are interpreted as real intervals.

[38] Cfr. 2.2.
[39] [54].

Another first–order language with equality, namely the language $\mathcal{L}_m := <=, \leq, r_i >_{r \in \mathbf{R}}$, has to be introduced to define NB.

An R–interpretation of the formulae of this language interprets r_i as a real number and it always assigns to $\leq$ the usual linear order relation.

The *point form* of an interval formula $\varphi := x_i\{B_1 \ldots B_n\}x_j$ is the set of clauses θ in $\mathcal{L}_m$ so that any I–model of φ can be translated into an R–model of this set θ, by translating the Allen relations into end–point relations involving $=, \leq$ and their negation, and vice versa. For instance $x_i\, begins\, x_j$ gets $\{b_{x_i} = b_{x_j}, e_{x_i} \leq e_{x_j}, e_{x_i} \neq e_{x_j}\}$, where b_x represents the beginning point and e_x the ending point of x; that is, if $\mathcal{I}(x) = a$, then the corresponding R–interpretation will have to interpret b_x as the beginning point of a and e_x as the ending one.

In the following we are only concerned with clauses whose literals do not allow the negation of $\leq$; the *ORD–point form* of an interval formula $\varphi := x_i\{B_1 \ldots B_n\}x_j$, written as $\pi(\varphi)$, is the point form of φ so that its compound clauses are so restricted. For every interval formula we can find its ORD–point form, since we can equivalently[40] reduce $r_i \not\leq r_j$ to $\{r_j \leq r_i, r_j \neq r_i\}$.

Proposition 2.11.4 *Let Θ be a set of interval formulae; Θ is I–satisfiable iff $\pi(\Theta)$ is R–satisfiable.*

pf. Immediate by definition.

We select a subclass of these closed clauses to define NB: we consider the subset NB of Allen relations st $\pi(\text{NB})$ has only ORD–point form sets whose clauses contain *at most* one positive literal; so these clauses are called *ORD Horn clauses.*

Not all of the Allen relations can be translated this way; for instance

$$x_1\{overlaps,\ overlapped\_by,\ meets,\ met\_by\}x_2$$

is translated as

$$\begin{aligned}&\{(b_{x_1} \leq b_{x_2} \vee b_{x_2} \leq b_{x_1}),\\ &(e_{x_1} \leq e_{x_2} \vee e_{x_2} \leq e_{x_1}),\\ &(b_{x_1} \neq b_{x_2}),\\ &(e_{x_1} \neq e_{x_2}),\\ &(e_{x_1} \neq b_{x_2}),\\ &(e_{x_2} \neq b_{x_1})\}\end{aligned}$$

[40] Equivalently with respect to R–interpretations.

Let's now consider the theory ORD which axiomatizes $=$ as a congruence relation with respect to $\leq$ and $\leq$ as a partial order on the equivalence classes, that is:

$$\begin{array}{l}\forall x\, x = x \\ \forall x\, y\, (x = y \,\wedge\, y = x \;\rightarrow\; x = y) \\ \forall x\, y\, z\, (x = y \,\wedge\, y = z \;\rightarrow\; x = z) \\ \forall x\, y\, (x = y \;\rightarrow\; x \leq y) \\ \forall x\, y\, (x = y \;\rightarrow\; y \leq x) \\ \forall x\, x \leq x \\ \forall x\, y\, (x \leq y \,\wedge\, y \leq x \;\rightarrow\; x = y) \\ \forall x\, y\, z\, (x \leq y \,\wedge\, y \leq z \;\rightarrow\; x \leq z)\end{array}$$

Proposition 2.11.5 *A finite set Γ of ORD–Horn clauses is R–satisfiable iff $ORD \cup \Gamma$ is satisfiable.*

pf. If Γ is R–satisfiable, then $R := <\mathbf{R}, \leq>$ is a model of Γ and so of $ORD \cup \Gamma$.
Let's suppose that $U \models ORD \cup \Gamma$, which implies that $=$ is a congruence relation and so that $U/= \,\models \Gamma$; since $U/=$ is logically equivalent to U, $U/=$ is a model of $ORD \cup \Gamma$; but every partially ordered set can be extended to a linearly ordered one which is equivalent to it and can be embedded into R, it follows that R satisfies Γ.

Let's denote with ORD_Γ the skolemization of ORD obtained by means of the endpoints occurring in Γ[41]; as a corollary of Herbrand theorem we get the following proposition.

Proposition 2.11.6 *$ORD \cup \Gamma$ is satisfiable iff $ORD_\Gamma \cup \Gamma$ is satisfiable.*

Now we can give a sketch of the proof of the completeness of path–consistency for determining the consistency of NB constraint problems[42].

Lemma 2.11.1 *Let Θ be a path–consistent set of interval formulae whose relations are in NB; Θ is I–satisfiable iff the empty relation is not among those occurring in Θ.*

pf. A case analysis of the possible non–unit clauses in $\pi(\Theta) \cup ORD_{\pi(\Theta)}$ shows that no new units can be derived by positive unit resolution, because of path–consistency. By refutation completeness of positive unit resolution we get our result.

[41] If Γ is finite, then ORD_Γ is finite, because there are not function symbols.
[42] For the full proof, see [43].

1. **Input:** a set N of NB relations.
   Output: a solution if the problem is consistent; `nil` otherwise.

2. Reduce the given problem to a path–consistent one; if it is not consistent, then **return** `nil`; otherwise, let M be the reduced path–consistent set of NB relations.

3. Execute CSPAN (cfr. table 2.2.3) on a set of PA–constraints in $\pi_1(M) \cup D$, where $\pi_1(M)$ is the set of unary clauses in $\pi(M)$, $\pi_2(M)$ that of binary ones (the soundness of this algorithm also relies on the fact $\pi(M) = \pi_1(M) \cup \pi_2(M)$) and D is a set of $\neq$–relations consisting of a $\neq$–disjunct for each clause in $\pi_2(M)$. Let s be the solution computed by CSPAN.

4. Assigning to each interval endpoint of M a number consistent with s, a solution for M (therefore for N) is assembled.

Table 2.7: Procedure to get a solution for NB–problems

Lemma 2.11.2 *NB is a subalgebra of IA.*

pf. The only difficulty lies in showing the closure wrt composition.

From the two previous lemmas the result below immediatly follows.

Theorem 2.12 *The satisfiability of a set of NB relations can be decided by means of path–consistency.*

Corollary 2.13 *The satisfiability of a (finite) subset of NB relations can be decided in polynomial time; the same claim holds for any subalgebra of NB.*

pf. As path–consistency algorithms run in $O(n^3)$ and theorem 2.12, our claim trivially follows.

Remark 2.4 *Since SA is a subalgebra of IA and is a subset of NB, the previous result applies to SA constraint problems as well.*

A method for finding a solution for NB constraint problems is given in [22]; if the given constraint problem is already path–consistent, it takes $O(n^2)$ time, where n is the number of variables in the problem; otherwise it takes a a time in $O(n^3)$; this procedure is given in table 2.7.

So far, we have seen that NB is indeed a subalgebra of IA, but we have still to see that it is the maximal one including all Allen atoms; an important tool to prove the maximality of NB is given by the concept of "closure in IA".

Definition 2.13.1 *Let S be a subset of relations of IA; $C_{IA}(S)$ is the minimal subalgebra of IA containing S; it is called the* IA–closure *of S.*[43]

Closure can be computed using the utility `aclose`. The pivotal role played by the closure operator is made clear by this result.

Proposition 2.13.1 *Let S be a subset of relations of IA; checking the satisfiability of $C_{IA}(S)$ is either a polynomial or an NP–complete problem iff respectively that of S is.*

pf. Cfr. [43].

Theorem 2.14 *NB is the maximal subalgebra of NB containing all of the Allen atoms and whose consistency can be decided in polynomial time.*

pf. NB is a subalgebra of IA[44]. Let $S \subseteq IA$ strictly contain all NB relations; running the utility `aclose`, it turns out that its closure includes at least one of two relations for which checking satisfiability is NP–complete[45]; by proposition 2.13.1, it follows that the satisfiability prolem for S is NP–complete as well.

New maximal subalgebras

A line of research has been open in finding other subalgebras of IA, incomparable with NB, but whose consistency can be still decided in polynomial time.

Intractable subsets

The main subsets of Allen relations used to prove intractability of some subalgebras of IA are those given in the following definition.

Definition 2.14.1 *Let's call A the set given by the following relations:*

$\{before, includes, overlaps, meets, finished - by\}$
$\{before, during, overlaps, meets, starts\}$

We can now define the following sets of Allen relations.

[43] This means that it is the least subset of IA containing S and that is closed under converse, intersection and composition.
[44] Cfr. lemma 2.11.2.
[45] Cfr. [43].

$\mathcal{N}_1 := A \cup \{during, includes, overlapped - by, started - by, finishes\}$
$\mathcal{N}_2 := A \cup \{includes, overlaps, overlapped - by, started - by, finished - by\}$
$\mathcal{N}_3 := \{before, after\} \cup \{overlaps, overlapped - by\}$
$\mathcal{N}_4 := \{before, after\} \cup \{overlaps, overlapped - by, meets, met - by\}$
$\mathcal{N}_5 := \{meets, met - by\} \cup \{before, after, starts, started - by, finishes, finished - by\}$

The fact that they are intractable is proved in [16].

Tractable subalgebras

Let b one of Allen atoms except *meets* and *met* $-$ *by*. Let r be one of the following compound relations.

$\{before,\ includes\ , overlaps, meets, starts, finished - by\}$
$\{before, during, overlaps, meets, started - by, finished - by\}$
$\{before, during, overlaps, meets, starts, finishes\}$
$\{before, during, overlaps, meets, starts, finished - by\}$

Definition 2.14.2 *$A(r, b)$ is the set of all Allen relations r' which satisfy the following conditions.*

$\{b, b^{-1}\} \subseteq r'$
$\{b\} \subseteq r' \subseteq \{=\} \cup r$
$\{b^{-1}\} \subseteq r' \subseteq \{=\} \cup r^{-1}$
$r' \subseteq \{=\}$

Proposition 2.14.1 *There are* 20 *distinct $A(r, b)$ sets*[46] *and all they are algebras*[47] *containing* 2178 *elements; furthermore each of them contains exactly* 3 *basic relations, namely b, b^{-1} and $=$.*

pf. See [17].

Among these algebras there are four, namely $A(r,$ *before* $)$, which contain the relations $=$, *before* , { *before* , $=$ }, *after* , { *after* , $=$ }, { *before* , *after* }; these relations are needed to express the notion of *sequentiality*, useful to argue about actions. The NB algebra does not contain the relation

{ *before* , *after* }

and so it cannot express the notion of sequentiality.

[46] Since $A(r^{-1}, b^{-1}) = A(r, b)$.
[47] It is easily verified running the utility `aclose`.

SAT($A(r,b)$)
1. Redirect the arcs of $\mathcal{C}$ to get relations in $A_1(b) \cup A_2(r,b) \cup A_3(r,b)$
2. Remove arcs whose labels are not in $A_2(r,b) \cup A_3(r,b)$; call $\mathcal{C}'$ the result
3. Find the strong componentsSC of $\mathcal{C}'$[48]
4. **for** every arc e in $\mathcal{C}'$ whose relation does not contain $=$ **do**
5. **if** e connects two nodes in some SC reject
6. accept

Table 2.8: FEASIBLE algorithm to find *all* consistent instantiations of PA

An algorithm to solve satisfiability for each $A(r,b)$ is given in table 2.2.3; the input constraint problem $\mathcal{C}$ is represented as a directed labeled graph where the label on the arc (x,y) is r iff the constraint on (x,y) is the relation r. It runs in time $O(n^2)$, where n is the number of interval variables: cfr. [17].

We are not simply interested in tractable subalgebras of IA but in the maximal ones.

Proposition 2.14.2 *Let b be either $finishes$ or $starts$; then $A(r,b)$ are maximal tractable algebras. All the other $(20-8)$ ones are not tractable.*

pf. Running the utility `atry`, the minimal extensions of $A(r,b)$ are generated adding a relation and computing the closure in IA of the new set; since the closure of every set so generated is IA itself and deciding the satisfiability of IA is an NP–complete problem, by proposition 2.13.1 the first result follows. The extensions of the other 12 algebras will be given further on.

There is another maximal tractable subalgebra of IA, namely $A_=$, defined below.

Definition 2.14.3 *$A_=$ is the algebra that contains every relations which contains $=$ and the empty one.*

That $A_=$ is an algebra can be easily verified by hand; it contains 4097 elements. An easy algorithm to check its satisfiability is the following: if some relation contains the empty one, then inconsistency; else it is satisfiable. Arguing as with the other 8 maximal algebra, by means of the utility `atry` we get the following result.

Proposition 2.14.3 *$A_=$ is a maximal tractable algebra.*

Notwithstanding its cardinality and the fact that it is a maximal tractable subalgebra of IA, the expressive power of $A_=$ is clearly too weak.

One of the most adopted techniques to prove consistencies of IA problems is to split compound relations into relations from some algebra for which the path–consistency algorithm is complete; it turns out that the path–consistency algorithm is complete for all $A(r,b)$ algebras and for $A_=$ too.

Theorem 2.15 *Let $A(r,b)$ one of the algebras in definition 2.14.2; the path–consistency algorithm decides satisfiability for $A(r,b)$ and $A_=$.*

pf. Cfr. [17].

Tractable subalgebras via metric constraints In their paper [15], Drakengren and Jonsson identify more tractable subalgebras of IA via metric constraints in the form of Horn disjunctive linear relations (DRLs), whose expressiveness subsume that of the NB algebra.

In order to define these new algebras, first we need to introduce a new first order language: it is an extension of L_m[49] by means of a binary function symbol $+$, whose intended interpretation is that of sum over rational\real numbers.

Definition 2.15.1 *A* linear relation *over a finite set of real–valued variables, let's say the set $\{x_1,\ldots,x_n\}$, is an expression of the form $(a_1x_i+a_0)\,r\,(b_1x_i+b_0)$, where a_1,a_0,b_1,b_0 are constant symbols (they stay for real numbers) and r is either $<$, $\leq$, $=$, $\neq$, $\geq$, $>$. A* disjunctive linear relation *(DLR) over $\{x_1,\ldots,x_n\}$ is a disjunction of one or more linear relations. A DLR is said to be Horn iff "at most" one of its disjuncts is not of the form $(a_1x_i+a_0)\neq(b_1x_i+b_0)$.*

Proposition 2.15.1 *There is a polynomial–time algorithm to decide the satisfiability of Horn DLRs.*

pf. Cfr. [15] .

Remark 2.5 *Nevertheless this, the eight maximal algebras presented in [17] are not expressible as Horn DLRs.*

Now let's give the explicit transformation from interval relations to ending–point relations as given in [15].

Definition 2.15.2 *Let's suppose that b is one of the Allen basic relations, I and J interval variables; then we can define a* binary relation *$sprel(r)$* between the starting points *of I and J in the following way.*

[49] Cfr. section 2.1.

$sprel(=) := \text{“}=\text{''}$
$sprel(before) := \text{“}<\text{''}$
$sprel(during) := \text{“}>\text{''}$
$sprel(overlaps) := \text{“}<\text{''}$
$sprel(meets) := \text{“}<\text{''}$
$sprel(starts) := \text{“}=\text{''}$
$sprel(finishes) := \text{“}>\text{''}$
$sprel(r^{-1}) := (sprel(r))^{-1}$

Similarly we can define a binary relation $eprel(r)$ between the ending points *of* I *and* J*. In the case of a compound relation* $r := b_1 \vee \ldots \vee b_n$*, then* $sprel(r) := sprel(b_1) \vee \ldots \vee sprel(b_n)$ *and* $eprel(r) := eprel(b_1) \vee \ldots \vee eprel(b_n)$*. Furthermore* $sprel^+(r) := sprel(r \cap \{=, \mathit{finishes}, \mathit{finished} - by\})$ *and symmetrically* $sprel^+(r) := sprel(r \cap \{=, starts, started - by\})$*.*

In [15], Drakengren and Jonsson use this transformation to transfer information from interval relations to point relations and vice versa, maintaining satisfiability: they define two new classes of subalgebras of IA, calling them *starting point* and *ending point* algebra; then they present an algorithm which, using the procedures for checking the satisfiability of Horn and PA DLRs, is able to decide if these algebras are satisfiable or not in polynomial time.

They also identify 8 new subalgebras of IA, given in the definition below.

Definition 2.15.3 *Let's define the following two relations.*[50]

$r_s := \{\, after, during, overlapped - by, met - by, \mathit{finishes}\}$
$r_e := \{before, during, overlaps, meets, starts\}$

If b *is one of the relations after, during, overlapped* $-$ *by, then let's define* $S(b)$ *as the set of relations* r *such that either one of the following holds.*

$\{b, b^{-1}\} \subseteq r$
$\{b\} \subseteq r \subseteq r_s \cup \{=, starts, started - by\}$
$\{b^{-1}\} \subseteq r \subseteq r_s^{-1} \cup \{=, starts, started - by\}$
$r \subseteq r_s\{=, starts, started - by\}$

Simmetrically, $E(b)$ *is defined as the set of relations* r *satisfying the following conditions.*

$\{b, b^{-1}\} \subseteq r$
$\{b\} \subseteq r \subseteq r_e \cup \{= \ \mathit{finishes}, \mathit{finished} - by\}$
$\{b^{-1}\} \subseteq r \subseteq r_e^{-1} \cup \{=, \mathit{finishes}, \mathit{finished} - by\}$
$r \subseteq r_e\{=, \mathit{finishes}, \mathit{finished} - by\}$

[50] Observe that r_s contains all basic relations b such that whenever IbJ for interval variables I and J, then $I^- > J^-$ has to hold in any model and, symmetrically, r_e is equivalent to $I^+ < J^+$ holding in any model.

Let's now define the set S^* as made up of relations r satisfying the following conditions.

$\{=, finishes, finished - by\} \subseteq r$
$\{finishes, finished - by\} \subseteq r \subseteq r_s \cup r_s^{-1}$
$\{=, finishes\} \subseteq r \subseteq r_s \cup \{starts, started - by\}$
$\{=, finished - by\} \subseteq r \subseteq r_s^{-1} \cup \{starts, started - by\}$
$\{finishes\} \subseteq r \subseteq r_s$
$\{finished - by\} \subseteq r \subseteq r_s^{-1}$
$\{=\} \subseteq r \subseteq \{=, starts, started - by\}$
$r = \emptyset$

Simmetrically, replacing finishes by starts and so their inverses, $\{=, starts, started - by\}$ *by* $\{=, finishes, finished - by\}$ *and* r_s *by* r_e *we get the subset* E^*.

The main results concerning these eight new sets are collected in this proposition; for a proof, cfr. [15].

Proposition 2.15.2 *$S(b)$ and S^* are starting point algebras; $E(b)$ and E^* are ending point algebras.*
The six algebras $S(b)$ and $E(b)$ contain 2312 elements, while S^* and E^* contain 1445 each; the basic relations contained in $S(b)$ are $=$, r, r^{-1}, starts and started $-$ by, while those contained in $E(b)$ are $=$, r, r^{-1}, finishes and finished $-$ by.
In all these algebras there are relations which are not expressible as HORN DLRs.

The last fact is easily proved observing that the point relations induced by Allen relations

{ *before* , *after* },
{ *during* , *includes* },
{ *overlaps* , *overlapped_by* , }
{ *after* , *finished_by* },
{ *before* , *starts* },

are not Horn DLRs.

The first result allows using their algorithm in order to check satisfiability for these eight new subalgebras of IA, so they are tractable algebras.

By running the utility `atry`, we see that these eight algebras are maximal tractable subalgebras of IA.

Proposition 2.15.3 *The algebras $S(b)$, $E(b)$, S^* and E^* are maximal tractable.*

The classification In [16], Drakengren and Jonsson present a more general classification of maximal tractable subclasses of Allen's algebra.

Let $\mathcal{R}$ a set equipped with an operator $C_{\mathcal{R}} : \mathcal{P}(\mathcal{R}) \to \mathcal{P}(\mathcal{R})$; let's suppose that for each $R \subseteq \mathcal{R}$ a problem of satisfiability $SAT(R;\mathcal{R})$ can be defined satisfying the following:

1. if $SAT(C(\mathcal{R});\mathcal{R})$ is NP–complete, then $SAT(R;\mathcal{R})$ is NP–complete;
2. if $SAT(R;\mathcal{R})$ is NP–complete, then $SAT(S;\mathcal{R})$ is NP–complete for any S containing R;
3. if $SAT(R;\mathcal{R})$ is polynomial, then $SAT(R;\mathcal{R})$ is polynomial for all $S \subseteq R$.

Let $\mathcal{R}_P$, $\mathcal{R}_{NP}$ subsets of $\mathcal{P}(\mathcal{R})$ and $\mathcal{B} \subseteq \mathcal{R}$ such that $SAT(S;\mathcal{R})$ is polynomial for each $S \in \mathcal{R}_P$ and NP–complete for each $S \in \mathcal{R}_{NP}$; furthermore $\mathcal{B} \subseteq S$, for each $S \in \mathcal{R}_P$.

Theorem 2.16 *If each subset T of $\mathcal{R}$ of cardinality less than that of $\mathcal{R}_P$ satisfies*

either that $T \subseteq S$ for some $S \in \mathcal{R}_P$,

or that $S \subseteq C_{\mathcal{R}}(T \cup \mathcal{B})$, for some $S \in \mathcal{R}_{NP}$,

then, for any S containing $\mathcal{B}$, $SAT(S;\mathcal{R})$ is polynomial iff S is a subset of some set in $\mathcal{R}_P$, otherwise it is NP–complete.

pf. Cfr. [16], pg. 1468.

Allen's algebra satisfies the hypotheses of this theorem. Since the satisfiability problem for Allen's algebra is NP–hard and the set B of basic relations is in $C_{IA}(\{meets\})$[51], we get the following result.

Proposition 2.16.1 *If A is a subset of IA and meets is one of its relations, then either A is a subset of the ORD–Horn algebra or its satisfiability problem is NP–complete.*

As applications of theorem 2.16, we get the following three propositions.

[51] Running the utility `aclose`.

Proposition 2.16.2 *If A is a subset of IA and $before$ is one of its relations, then either A is a subset of the NB algebra, $A \subseteq S(before)$, $A \subseteq E(before)$ or its satisfiability problem is NP–complete.*

This result reduces the number of basic relations allowed in a maximal tractable subalgebra of IA not included in the NB algebra or in any $S(b)$ or $E(b)$, $b \in \{before, during, overlaps\}$, to at most 9; let's call T this class of algebras.

Proposition 2.16.3 *If A is a subset of IA, during and overlaps or starts and $finishes$ are among its basic relations, then either A is a subset of the NB algebra or its satisfiability problem is NP–complete.*

This result and the remaining proposition reduce the number of basic relations allowed in T to at most 3.

Proposition 2.16.4 *If A is a subset of IA, $starts$ and $during$ are among its basic relations, then either A is a subset of the NB algebra, A is a subset of $S(during)$ or its satisfiability problem is NP–complete.*
If A is a subset of IA, $starts$ and $overlaps$ are among its basic relations, then either A is a subset of the NB algebra, A is a subset of $S(overlaps)$ or its satisfiability problem is NP–complete.
The same result holds replacing $finishes$ by $starts$, using E instead of S.

In synthesis: the maximal tractable subalgebras of IA which are included neither in the NB algebra, in $S(b)$ nor in $E(b)$ for $b \in \{before, during, overlaps\}$, can only contain at most 3 among the basic relations $=$, *during*, *overlaps*, *starts*, *finishes*.

Finally, if $\{before,\ after\}$ is among the relations of a subalgebra we get this last classification result.

Proposition 2.16.5 *If A is a subset of IA, $\{before,\ after\}$ is among its relations, then either $A \subseteq S(before)$ or its satisfiability problem is NP–complete.*

2.2.4 The main techniques to find a solution to the general problem

Path–consistency

Path–consistency can be used by itself as a heuristic test for consistency or in a backtracking search for consistencies in which it can be applied as a preprocessing algorithm or interleaved with the other techniques. We have already discussed about it in 2.2.2, since it was the first technique proposed to approximate solutions in IA.

Backtracking algorithms

Backtracking for finding a solution proceeds by instantiating one variable per time; if no consistent instantiation is found for the variable under examination, the search backs up. The order in which variables are instantiated and values chosen in the domains turns out to be important for speeding up backtracking algorithms.

The idea behind *variable ordering heuristics* is to instantiate variables first that will constraint the instantiation of the other variables the most; this way it is more likely that a possible backtracking search is executed at the beginning and is not delayed.

Value ordering heuristics aim at ordering those values which constraint the choices for other variables the least, because such values are the most likely to be part of a possible solution.

Subalgebras of IA

One can also restrict the search to tractable subproblems of IA to find a solution\all solutions for the given IA problem: first splitting the IA problem in some\all tractable ones we have chosen to deal with (either PA or NB etc.) and trying to find a consistent instantiation for them; then "assembling" a solution\all solutions for the general IA problem.

In particular, choosing PA (NB as well), we can follow this procedure[52].

One) First one can look for a subproblem SA of the given IA, selecting from each of its constraint $\{b_1, \ldots, b_n\}$[53] a subset of allowed relations to get an SA problem; then one can either translate the SA problem into a PA one and check consistency applying CSPAN(PA) till the seventh line or apply a path–consistency algorithm; finally, if not already done before, one should translate the SA problem into a PA one and pass it to the whole CSPAN algorithm to find a consistent instantiation[54]; see table 2.2.4.

All) The idea behind this algorithm is similar to the previous one, namely splitting the IA problem in "all" the possible consistent SA ones and then

[52]Cfr. [53].
[53]$b_1, \ldots, b_n$ are some of the Allen atoms.
[54]Cfr. table 2.2.3.

CSIA(C)

1. Find a consistent SA constraint subproblem, S, of C; use backtrack search every time an inconsistent S is detected; if no such an SA problem is found, return "inconsistent";
2. translate S into a PA constraint problem P;
3. run CSPAN(P).

applying to each of them FEASIBLE; the solution of the IA problem is the union of all these solutions.

The first procedure turns out to be useful; it can be improved interleaving it with backtracking algorithms: for instance, after a solution for a particular subclass is found, one can apply a variable ordering heuristic to speed up the search. The second procedure is practical only for small instances of the problem.

Since NB is path–consistent, in [42], Nebel himself modifies Ladkin and Reinefeld's algorithm to gain a path–consistent problem so that it works with every subalgebra of IA for which path–consistency is equivalent to consistency: first they prune the search space reducing the given CP to a path–consistent one; then they choose an unprocessed constraint and select from it a subset of the Allen relations which belongs to the subalgebra chosen (for instance NB) and re–run the path–consistency algorithm; they instantiate the chosen constraint with each one of this relation, every time running the path–consistency algorithm on the new CP so obtained.

2.3 Relations among the qualitative and the quantitative approaches

2.3.1 Metric constraints between points and qualitative constraints between intervals

In [28], there is the first attempt to mix the qualitative interval–based approach and the quantitative point–based one in a logical framework able to capture the expressive power of both languages; the complexity results are not optimal, but the general framework has been introductory to further developments.

A first–order two–sorted language, L, is introduced to formalize Allen's relation algebra and STCPs of Detcher, Meiri and Pearl[55] enriched to deal with

[55] Cfr. [13].

strict inequalities too[56]; there are two types, one for *points* and one for *intervals*.

Definition 2.16.1 *A two–sorted first order language is introduced this way:*

two distinct types of variables:

- *point variables, corresponding to rational numbers or* ∞*:* x, y, ...
- *interval variables, corresponding to rational valued intervals:* i, j, ...

function and constant symbols:

- L, R *from the set of interval variables to that of point variables*[57]*;*
- $-$*, which formalizes subtraction between rational numbers, is a function from the set of couples of point variables to the set of point variables;*
- *point functions to construct rational numbers;*
- *a constant symbol* ∞ *of type point;*

two classes of relation symbols:

- *two binary relation symbols,* $<$ *and* $\leq$*, such that* $t_1 < t_2$ *or* $t_1 \leq t_3$ *iff* t_1, t_2 *and* t_3 *are point terms and* t_3 *can be* ∞*;*
- 13 *relation symbols corresponding to the Allen relations, including* $=$*, holding between interval terms.*

Formulas of the form

$$(i\,B_1\,j) \vee \ldots \vee (i\,B_n\,j),$$

where $B_1, \ldots, B_n$ are atoms of Allens' algebra, are called *simple Allen constraints* and individuate a sublanguage called $\mathcal{L}_A$.

Formulas of the form

$$(F(i) - G(j)) \leq n \wedge (G(j) - F(i)) \leq m,$$

where F and G are L or R, $\leq$ may be replaced by $<$, n and m are numerals or ∞, are called *simple metric constraints* and individuate a sublanguage called $\mathcal{L}_M$.

[56]Cfr. p. 23.

[57]Intutitively, $L(i)$ stays for the left endpoint of i and $R(i)$ for the right one.

Further Kautz and Ladkin present a series of axioms to get the intended model of time:

- arithmetic axioms for $-$, $<$, $\leq$ and for numerals; these last ones include $\forall x\, x < \infty$
- $\forall i\, L(i) < R(i)$
- axioms for each one of the Allen atoms:
  1. $\forall i, j\, i = j \leftrightarrow L(i) - L(j) \leq 0 \wedge L(j) - L(i) \leq 0 \wedge R(i) - R(j) \leq 0 \wedge R(j) - R(i) \leq 0$
  2. $\forall i, j\, i$ *before* $j \leftrightarrow R(i) - L(j) < 0$
  3. $\forall i, j\, i$ *meets* $j \leftrightarrow R(i) - L(j) \leq 0 \wedge L(j) - R(i) \leq 0$
  4. $\forall i, j\, i$ *overlaps* $j \leftrightarrow L(i) - L(j) < 0 \wedge L(j) - R(i) < 0 \wedge R(i) - R(j) < 0$
  5. $\forall i, j\, i$ *starts* $j \leftrightarrow L(i) - L(j) \leq 0 \wedge L(j) - L(i) \leq 0 \wedge R(i) - R(j) < 0$
  6. $\forall i, j\, i$ *during* $j \leftrightarrow L(j) - L(i) < 0 \wedge R(i) - R(j) < 0$
  7. $\forall i, j\, i$ *finishes* $j \leftrightarrow L(j) - L(i) < 0 \wedge R(i) - R(j) \leq 0 \wedge R(j) - R(i) \leq 0$

To compute the minimal equivalent constraint problem of a simple temporal constraint problem with strict inequalities we use the procedure presented on page 23, adding the constraints from $\mathcal{L}$ which state that the left point of an interval is before its right one, this for every interval variable i in the problem; that is we add an arc $(L(i), R(i))$ with label $(0, 0)$ for each i.

Kautz and Ladkin present a method to compute the minimal constraint problem representation in the case of simple temporal constraints with strict inequalities; they use the approximation algorithms for constraints in $\mathcal{L}_A$.

Combining these two procedures they get a constraint satisfaction algorithm for $\mathcal{L}_A \cup \mathcal{L}_M$: given Allen constraints A and metric ones M, $m(A)$ and $m(M)$ are separatately computed; new Allen constraints are derived from the metric ones and added to $m(A)$; new metric constraints are derived from the new Allen constraints and so on till no new constraints can be derived. This procedure is sound but not optimal: it runs in $O(n^2(e + n^3))$ where n is the number of intervals that appear in $M \cup A$ and e is the time required to compute $m(A)$.

2.3.2 Qualitative constraints between points and intervals, quantitative constraints between points

In [40], Meiri combines qualitative constraints between intervals (II), between points (PP), quantitative ones between points, mixed ones between points and intervals (PI) or intervals and points (IP).

A CP can be so stated:

- the two–sorted first order language in definition 2.16.1 is augmented with new binary relation symbols for IP and PI constraint relations[58];
- *external constraints* are unary (only the metric ones) or binary and they can be qualitative (between two intervals or two points) or quantitative;
- *internal constraints* relate each interval variable i to its end points and they are[59]:
    1. $\forall i\, x((L(i)\ \mathit{starts}\ i \wedge R(i)\ \mathit{finishes}\ i) \wedge (x\ \mathit{starts}\ i \rightarrow x = L(i)) \wedge (x\ \mathit{finishes}\ i \rightarrow x = R(i)))$;
    2. $\forall i\, (L(i) < R(i))$.

A relation of order between constraints of the same type is estabilished as usual; this relation is extended to CPs saying that $CP1 \subseteq CP2$ if whenever C_s is a constraint in $CP1$ on the set of variables s, for any $C_{s'}$ constraint in $CP2$ on s, $C_s \subseteq C_{s'}$. Since equivalent CPs are closed under intersection, there exists and is unique the *minimal CP* equivalent to a given one.

The constraint tecniques adopted serve different purposes:

1. some of them aim at finding a solution to the given CP decomposing it into singleton constraint subproblems which are solvable in polynomial time; sometimes backtracking algorithms are used to improve the search;
2. path–consistency tecniques can be introduced to prune the search space or to compute an approximation to the minimal constraint problem.

Meiri has identified two classes of tractable problems solvable in polynomial time:

i. the first class consists of CPs composed of *qualitative constraints between points (PA)* and of *unary quantitative constraints between points*;

ii. the second class consists of CPs for which *path–consistency algorithms are exact.*

[58] Cfr. [40].
[59] Assuming that these variable are of different type.

Qualitative constraints

A qualitative constraint between two points (PP), two intervals (II) or a point and an interval (PI or IP) is a disjunction like $x_i R_1 x_j \vee \ldots \vee x_i R_n x_j$, where the R_i are basic relations of three possible types: II (the Allen ones), PP ($\{<, =, >\}$), PI, IP (cfr. fg. 1 and table 1 on p. 346 of [40]).

The *qualitative algebra* QA is the relational algebra whose elements are the 2^{13} Allen relations, the 2^3 PP relations, the 2^5 PI ones and the remaining 2^5 IP ones. The internal operations are those of intersection, denoted by $\cap$, and of composition, denoted by $\odot$; they are given in tables 3–5 on pp. 347–348 of Meiri's paper; $\emptyset$ corresponds to illegal combinations.

Quantitative constraints

These constraints relate points and they can both be unary or binary; in both cases a constraint is represented by a set of intervals $\{I_1, \ldots, I_k\}$, open or closed in either sides, that is the relation between two variables can be either $<$ ($>$) or $\leq$ ($\geq$).

Relations between the qualitative and the quantitative constraints

The existence of a constraint C between two points, let's say x_1 and x_2, of a certain type can imply the existence of a constraint of the other type between these two points; writing $Quant(C)$ if the given constraint C is quantitative, that is a set of intervals $\{I_1, \ldots, I_k\}$, and $Qual(C)$ otherwise, that is a subset involving $\{<, =, >\}$, we get the following implications:

1. $Quant(C) \to Qual(C)$:

   if $0 \in Quant(C)$, then "$=$" $\in Qual(C)$;
   if there is $r > 0$ st $r \in Quant(C)$, then "$<$" $\in Qual(C)$;
   if there is $r < 0$ st $r \in Quant(C)$, then "$>$" $\in Qual(C)$;

2. $Qual(C) \to Quant(C)$:

   if "$=$" $\in Qual(C)$, then $[0] \in Quant(C)$;
   if "$<$" $\in Qual(C)$, then $(0, \infty) \in Quant(C)$;
   if "$>$" $\in Qual(C)$, then $(-\infty, 0) \in Quant(C)$.

The operations of intersection and composition are extended to constraints C_1 of quantitative type and C_2 of qualitative one this way:

- $C_1 \cap C_2$, of qualitative type, is $C_1 \cap Quant(C_2)$;
- if C_2 is of type PP, then $C_1 \odot C_2$ is of quantitative type and is $C_1 \odot Quant(C_2)$; if C_2 is of type PI, then $C_1 \odot C_2$ is of qualitative type and is $Qual(C_1) \odot C_2$.

The hierarchy of qualitative constraint problems

If all constraints are II, we have an IA constraint problem; if all constraints are PP relations, then the constraint is a PA one, in particular, if the relations do not involve $\neq$, then we have a constraint subproblem of PA which is called *convex* PA, briefly CPA; if all constraints are PI and IP relations, then the CP is called an *interval–point algebra* CP, briefly an IPA constraint problem.

Let $net(S)$ be the set of qualitative constraints that can be represented as a CP of type S; for instance $i_1\{starts\}i_2$ can be represented as a PA constraint problem by $L(i_1) = L(i_2)$ and $R(i_1) < R(i_2)$, but it is not itself a PA constraint problem.

Proposition 2.16.6 *Let QCP the set of all qualitative constraints; then we have the following hierarchy:*

$$net(CPA) \subset net(PA) \subset net(IPA) \subset net(IA) = QCP$$

We already know that checking the satisfiability of IA constraint problems is an NP–hard problem; furthermore we have the following result, of which the proof requires the use of the internal constraints.

Theorem 2.17 *Deciding the consistency of an IPA constraint problem is NP–hard.*

pf. By reduction to the *betweenness problem* which is so stated: given a non empty set A and a set of ordered triplets (a, b, c) of elements of A, the question is if there is a one–to–one function from A in itself such that, for each triplet of distinct elements (a, b, c), we have either $f(a) < f(b) < f(c)$ or $f(c) < f(b) < f(c)$.

The first new class of tractable problems: augmented qualitative constraint problems on points with quantitative constraints

CPA or PA constraint problems augmented with unary quantitative constraints are considered, that is constraints on the domains of the following kinds.

1. Discrete and finite domains: $\{[d^1], \ldots, [d^k]\}$; in this case, if we only deal with CPA constraint problems, arc–consistency is enough to ensure consistency; deciding the consistency of PA constraint problems over discrete and finite domains is NP–hard.

2. Single–interval domains: $\{[d_1, \ldots, d_n]\}$, from which we can exclude a finite number of values (in this case we deal with "almost convex" single–interval domains); a nonempty arc–consistent *acyclic* PA constraint problem over almost convex single interval domains is consistent and its reduced constraint problem[60] is minimal (the algorithm given runs in time $O(e(k+n))$); a nonempty arc–consistent and path–consistent PA constraint problem over almost convex single–interval domains is consistent, its reduced is minimal and the algorithm runs in $O(n^4)$ time for single domains, in $O(n^4k^2)$ for almost convex single–interval domains.

3. Multiple–intervals domains: $\{[d_1^1, \ldots, d_n^k], \ldots, [d_1^k, \ldots, d_n^k]\}$; a nonempty arc–consistent *acyclic* CPA constraint problem is consistent and minimal (Meiri gives an algorithm which runs in $O(e\log(k))$ time), while a *cyclic* one requires path–consistency too in order to ensure consistency and minimality (the algorithm takes a time in $O(n^4k^2)$).

The second class of tractable problems: general constraint problems

Two kinds of algorithms are proposed: exact but exponential ones; path–consistency ones but incomplete.

The idea adopted is, as usual, splitting the original CSP in ones whose constraints are of the basic types and solvable in polynomial time; then one has to combine these partial solutions to get the whole one.

[60]The *reduce constraint problem* of a given one is obtained substituting each closed domain $[a, b]$ with (a, b); if the input CP was arc–consistent, its reduced one is arc–consistent too.

Chapter 3

Conclusions

Writing this thesis meant, first of all, reading a great amount of literature about Temporal Reasoning and Constraint Programming, dealing with the different formalizations of Temporal Constraint Programming and, above all, trying to get a homogeneous work.

We could have divided the literature about Temporal Constraint Programming into two main streams: in the first one, we would have classified the research on algorithms for solving TCPs; the other one is characterized by the study of algebraic or logical properties of the underlying temporal constraint frameworks.

However, this classification turned out to be too simplistic, since these two fields are not so clearly separate; as we have seen, most of the investigation on Allen's subalgebras, which are the main tools for reasoning in the qualitative approach to Temporal Constraint Programming, has been motivated by the fact that the problem of satisfiability for this Allen's algebra (IA) turned out to be NP–hard[1]. Therefore, we have preferred to classify the different approaches to Temporal Constraint Programming into three main branches, as we did in chapter 2.

The main two approaches we have identified are the *quantitative* (*metric*), first introduced by Dechter, Meiri and Pearl, and the *qualitative* based on Allen's interval algebra.

The metric characterization shows its efficiency when dealing with temporal problems involving metric information, like "Paulo leaves home at $9:10\,a.m.$" or "Eyal goes to the park either at $5:00$ or at $5:15\,p.m.$". Allen's approach is instead useful if the temporal information focuses on relations between intervals,

[1] Cfr. theorem 2.10.

like "Alessandra wakes up either *before* or *after* Raffaella does"; for instance, the relation of *disjointness* between intervals (the example just proposed) cannot be characterized in the metric approach with binary constraints.

In section 2.1 we have discussed the seminal work of Dechter, Meiri and Pearl on Temporal Constraint Programming with metric information; since the satisfiability problem for their general framework is NP–hard[2], authors have researched on what we have called simple temporal problems. We have studied the literature concerning this approach and presented it as a useful tool to enhance the search for solutions of general temporal constraint problems or to approximate them.

At the end of this section (cfr. p. 27), we tried to summarize the main techniques one can use when dealing with this kind of constraint problems.

In section 2.2, we have presented the qualitative approach, focusing on Allen's interval algebra. As the satisfiability problem for constraint problems based on IA is NP–hard, we directed our attention to subalgebras of IA which are tractable and expressive enough for many applications; for instance, we introduced the PA algebra and its properties as a tool for studying IA problems. This idea has guided us for the rest of our exposition; whenever we introduced metric temporal problems in this section, we used them to study interesting subalgebras of IA[3]. In fact, one of the most promising direction of research aims at classifying the maximal tractable subalgebras of IA with different expressive power; for instance, we have seen that the algebra of Nebel and Bürckert is the maximal tractable subalgebra of IA that contains all of the Allen atoms, but it cannot express the relation of sequentiality, that is given by $\{\mathit{before}, \mathit{after}\}$; however, this relation can be expressed by the 8 maximal tractable subalgebras $A(r, b)$ of Drakengren and Jonsson[4].

As in the metric approach, at the end of section 2.2 we have reviewed and summarized the main techniques that Constraint Programming offers to solve IA problems; the main one is given by path–consistency, for historical reason and because it is enough to guarantee consistency of the tractable subalgebras we have presented.

At the end, in section 2.3, we have briefly introduced a new line of investigation, which we called "the mixed approach". Research is open in this field; local consistency procedures and new classes of constraint relations are currently under investigation[5].

Temporal Constraint Programming could grow further, taking into consid-

---

[2] Cfr. theorem 2.1.
[3] Cfr. p. 49.
[4] Cfr. definition 2.14.2.
[5] Cfr. [40].

eration event calculus[6] and the systems adopted to reason about actions and changes. In our frameworks, time is the only ontological object, as events, fluents, states or actions are identified with their period of occurrence; this choice provides a computational and logical system easy to manage and efficient. However, the creation of a new constraint–based logical system, in which events, fluents or actions are considered as new objects of different type, represents an attractive challenge[7].

Aknowledgements

A warm thank is due to my Amsterdam friends who enriched my days and patiently listened to all my not little doubts; Paulo, with whom I shared a working summer discussing of temporal constraints and events (the most important ones concerning Brazilian music); my "little Italy" in Amsterdam, Alessandra, Riccardo, Raffaella, Caterina, Marco; not to forget Carlos, Nikos, Eyal, Renata, Rosalie and the many others I have been lucky to meet during this long year.

I thank Krzysztof Apt, Hans Rott, Johan van Benthem and all ILLC members who helped me with my work.

[6] For an introduction and a discussion about it, based on Shanahan's work, cfr. [44].
[7] A first work in this direction has been already proposed, [47].

Bibliography

[1] James F. Allen. Mantaining knowledge about temporal intervals. *Comm. ACM*, 26:832–843, 1983.

[2] James F. Allen. Towards a general theory of action and time. *AI*, 23:123–154, 1984.

[3] James F. Allen and George Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–579, October 1994.

[4] James F. Allen and Patrick J. Hayes. Moments and points in an interval-based temporal logic. *Computational Intelligence*, 5:225–238, 1989.

[5] K. R. Apt. From chaotic iteration to constraint propagation. In *ICALP–97*, pages 36–55. Springer–Verlag Lecture Notes in Computer Science, 1997.

[6] K.R. Apt. *From Logic Programming to Prolog.* Prentice Hall, 1997.

[7] K.R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 1998. In press.

[8] K.R. Apt. A proof theoretic view of constraint programming. *Fundamenta Informaticae*, 1998. In press.

[9] K.R. Apt and E.R. Olderog. *Verification of Sequential and Concurrent Programs.* Springer-Verlag, 1997.

[10] K.R. Apt and F. Turini. *Meta-logics and Logic Programming.* The MIT press, 1995.

[11] Fahiem Bacchus and Peter van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the 15th National Conference on Artificial Intelligence*, Madison, Wisconsin, July 1998.

[12] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Algorithms.* MIT Press and Mc–GrawHill, 1997 edition, 1990.

[13] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *AI*, 49:61–95, 1991.

[14] Rina Dechter and Peter van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173:283–308, 1997.

[15] Thomas Drakengren and Peter Jonsson. Eight maximal subclasses of allen's algebra with metric time. *Journal of Artificial Intelligence Research*, 7:25–45, 1997.

[16] Thomas Drakengren and Peter Jonsson. Towards a complete classification of tractability in allen's algebra. In *IJCAI–97*, 1997.

[17] Thomas Drakengren and Peter Jonsson. Twenty–one large tractable subclasses of allen's algebra. *AI*, 93:297–319, 1997.

[18] Eugene Freuder. Partial constraint satisfaction. In *IJCAI-89: Proceedings 11th International Joint Conference on Artificial Intelligence*, pages 278–283, Detroit, 1989.

[19] Eugene Freuder and Richard Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[20] Antony Galton. A critical examination of allen's theory of action and time. *Artificial Intelligence, March 1990(2-3)*, 42:159–188, 1990.

[21] Alfonso Gerevini and Matteo Cristani. On temporal constraint networks with inequations. Technical Report RT 199611–3, Dipartimento di Elettronica per l'Automazione, Universit di Brescia, November 1996.

[22] Alfonso Gerevini and Matteo Cristani. On finding a solution in temporal constraint satisfaction problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97*, pages 1460–1465, Nagoya, Japan, August 1997.

[23] Alfonso Gerevini and Lenhart K. Schubert. Efficient temporal reasoning through timegraphs. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 648–654, Chambéry, Savoie, France, 1993. Morgan-Kaufmann.

[24] Alfonso Gerevini and Lenhart K. Schubert. Efficient algorithms for qualitative reasoning about time. *AI*, 74(2):207–248, 1995.

[25] Alfonso Gerevini and Lenhart K. Schubert. On computing the minimal labels in time point algebra networks. *Computational Intelligence*, 11(3):443–448, 1995.

[26] I. Hamlet and J. Hunter. A representation of time for medical expert systems. In *Proceedings of European Conference on Artificial Intelligence in Medicine*, pages 112–119, 1987.

[27] Joxan Jaffar and Michael Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[28] Henry A. Kautz and Peter B. Ladkin. Integrating metric and qualitative temporal reasoning. In *Proceedings of AAAI*, 1991.

[29] Grzegorz Kondrak and Peter van Beek. A theoretical evaluation of selected backtracking algorithms. *AI magazine*, 89:365–387, 1997.

[30] Manolis Koubarakis. Dense time and temporal constraints with $\neq$. In *KR–92*, pages 24–35, 1992.

[31] Manolis Koubarakis. From local to global consistency in temporal constraint networks. *TCS 173*, 1:89–112, 1997.

[32] Peter Ladkin. *The Logic of Time Representation*. PhD thesis, University of California at Berkeley, 1987.

[33] Peter B. Ladkin and Roger D. Maddux. On binary constraint networks. Technical Report Tech. Rept. KES.U.88., Kestrel Institute, 1988.

[34] Peter B. Ladkin and Roger D. Maddux. On binary constraint problems. *Journal of the ACM*, 41(3), 1994.

[35] Peter B. Ladkin and Alexander Reinefeld. Fast algebraic methods for interval constraint problems. *to appear in Annal of Mathematics and Artificial Intelligence*.

[36] Peter B. Ladkin and Alexander Reinefeld. Effective solution of qualitative interval constraint problems. *AI*, 57:105–124, 1992.

[37] Gérard Ligozat. Corner relations in allen's algebra. In *Actes AAAI-96 Workshop on Spatial and Temporal Reasoning*, Portland, Oregon, 1996.

[38] Gérard Ligozat. A new proof of tractability for ord-horn relations. In *AAAI–96*, Portland, Oregon, 1996.

[39] Henry A. Kautz Marc B. Vilain. Constraint propagation algorithms for temporal reasoning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 377–382, 1986.

[40] Itay Meiri. Combining qualitative and quantitative constraints in temporal reasoning. In *AAAI*, volume 1, pages 260–267, 1991.

[41] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(2):95–132, 1974.

[42] Bernhard Nebel. Solving hard qualitative temporal reasoning problems: Evaluating the efficiency of using the ord–horn class. *CONSTRAINTS*, 1(3):175–190, 1997.

[43] Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen's interval algebr. *Journal of the ACM*, 42(1):43–66, 1995.

[44] Paulo Eduardo Santos. Formalizing common sense in a mobile robot. Master's thesis, ILLC, Amsterdam, The Netherlands, 1998.

[45] Eddie Schwalb and Rina Dechter. Coping with disjunctions in temporal constraint satisfaction problems. In *AAAI*, pages 127–132, 1993.

[46] Eddie Schwalb and Rina Dechter. Processing disjunctions in temporal constraint networks. *AI*, 93:29–61, 1997.

[47] Eddie Schwalb, Kalev Kask, and Rina Dechter. Temporal reasoning with constraints on fluents and events. In *AAAI*, volume 2, pages 1067–1072, 1994.

[48] Murray Shanahan. *Solving the Frame Problem.* The MIT Press, 1997.

[49] Oliviero Stock. *Spatial and Temporal Reasoning.* Kluwer Academic Publishers, 1997.

[50] Raùl E. Valdés-Pérez. The satisfiability of temporal constraint networks. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 256–260, 1987.

[51] Peter van Beek. Approximation algorithms for temporal reasoning. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1291–1296, Detroit, Michigan, August 1989.

[52] Peter van Beek. *Exact and approximate reasoning about qualitative temporal relations.* PhD thesis, University of Waterloo, 1990.

[53] Peter van Beek. Reasoning about qualitative temporal information. *AI*, 58:297–326, 1992.

[54] Peter van beek and Robin Cohen. Exact and approssimate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

[55] Peter van Beek and Rina Dechter. Constraint tightness and looseness versus local and global consistency. *Journal of the ACM*, 44:549–556, 1997.

[56] Peter van Beek and Dennis W. Manchak. The design and experimental analysis of algorithms for temporal reasoning. *Journal of AI Research*, 4:1–18, 1996.

[57] Joahn F.A.K. van Benthem. *The Logic of Time.* Reidel, 1983.

[58] Luìs Vila. A survey on temporal reasoning in artificial intelligence. *AI Communications*, 7(1):4–28, March 1994.