Handling Contingency in Temporal Constraint Networks: from Consistency to Controllabilities

Thierry VIDAL
LGP / ENIT
47 avenue d'Azereix
F-65016 Tarbes cedex
e-mail: thierry@enit.fr
phone: +33.5.62442762

Hélène FARGIER
IRIT / UPS
118, rte de Narbonne
F-31062 Toulouse, France
e-mail: fargier@irit.fr
phone: +33.5.61558297

Abstract

Temporal Constraint Networks (TCN) allow to express minimal and maximal durations between time-points. Though being used in many research areas, this model disregards the contingent *nature of some constraints, whose effective duration cannot be decided by the system but is provided by the external world. We propose an extension of TCN based on the definition of the* Simple Temporal Problem under Uncertainty (STPU) *in which the classical network consistency property must be redefined in terms of* controllability*: intuitively, we would like to say that a network is controllable iff it is consistent in any* situation *(i.e. any assignment of the whole set of contingent intervals) that may arise in the external world. Three levels of controllability must be distinguished, namely the* Strong*, the* Weak *and the* Dynamic *ones. This paper provides a full characterization of those properties and their usefulness in practice, and proposes algorithms for checking them. Complexity issues and tractable equivalence classes are only partially tackled, since it is still the topic of on-going work. All the same, hardness is discussed and argued, giving evidence for the general intractability of* Dynamic controllability*, which is the most commonly required property in domains such as planning or scheduling.*

1 Background and overview

A large number of research areas in Artificial Intelligence have to tackle the necessity of handling time in an explicit and highly expressive manner. This is especially true when one has to reason on the "activities" performed by some "agent" interacting with a so-called "external world". This defines in rough terms all the possible target areas of this paper, though we will most of the time preferably use terms and examples inspired by the planning community.

Apart from studies dealing with logics of time, researches have been carried out on temporal algebras, i.e. formalisms solely capturing the time entities (points or intervals) and the relations between them. Constraint Satisfaction Problems (Kumar, 1992) model them in terms of variables and binary constraints between them, and the main reasoning issue, on which this paper will focus, is then to check the *consistency* of the whole (i.e. check that one can find a complete assignment of the variables, called a *solution*, that satisfies the constraints). We should distinguish here between *symbolic* constraint algebras (Allen, 1983; Vilain et al., 1989), models dealing with *numerical* constraints (Dechter et al., 1991), and some advanced proposals

combining both (Meiri, 1991; Ghallab and Vidal, 1995; Drakengren and Jonsson, 1997).

Such CSP-based approaches have proven to be useful in such domains as scheduling (Dubois et al., 1993; Dorn, 1994), supervision (Dousson et al., 1993), diagnosis and temporal databases (Brusoni et al., 1994), multimedia authoring environments (Jourdan et al., 1997), or planning (Drabble and Kirby, 1991; Rutten and Hertzberg, 1993; Ghallab and Vidal, 1995; Cervoni et al., 1994). Different formalisms are used however, for instance the well-known *Time Map Manager* (Dean and McDermott, 1987) serves as an underlying representation and reasoning scheme in (Rutten and Hertzberg, 1993). We will tell a few words in the conclusion about how our approach can be compared to that line of research, but we are more particularly interested in this paper in the *Temporal Constraint Network (TCN)* (Dechter et al., 1991) graph-based formalism, on which we will focus. TCNs express numerical constraints between time-points as continuous intervals of possible values, which interestingly enough encompass symbolic constraints as well. A *duration* will be used in the following to designate one such value "assigned" to a constraint (within its bounds), encompassing as well the notion of *date* of a time-point, which is nothing but the duration between the origin of time (noted 0) and this time-point (Meiri, 1991) .

A large number of approaches in the domains we are interested in (see e.g. (Cervoni et al., 1994; Ghallab and Vidal, 1995; Brusoni et al., 1994)) have chosen to use the expressive power and practical efficiency of the polynomial restriction of TCN (namely STP) to check the temporal consistency of the plan [resp. schedule, diagnosis, etc]. But they disregard the inherent uncertain nature of durations of some tasks in realistic applications. One should indeed distinguish between *contingent* constraints (whose effective duration will only be observed at execution time. e.g. the duration of a task) and *free* ones (whose instantiation is controlled by the agent, e.g. a delay between starting times of tasks).

This leads to the redefinition of the concept of consistency in terms of *controllabilities*: intuitively a network is controllable if it is consistent in the classical sense in any *situation* that may arise in the external world, i.e. no matter how "Nature" will decide to assign the contingent intervals. Inspired by the thorough study carried out in the framework of discrete CSPs (Fargier et al., 1996) on the same kind of distinction, we exhibit three levels of controllability, namely the *Strong* one (i.e. existence of one "universal" solution), the *Weak* one (i.e. existence of a solution matching each complete "situation" likely to arise in the external world) and the *Dynamic* one, that best suits dynamic applications, where the effective task durations are only observed as far as execution is getting on.

Section 2 will give the formal definition of our new temporal problem under uncertainty, then section 3 will define the three levels of controllability. It is only in section 4 that we will discuss those three properties with respect to the constraint satisfaction area, to argue about their relative hardness. Then section 5 will provide algorithms for checking them, before section 6 suggests some first steps in the design of tractable classes.

2 Representation issues

2.1 Quantitative Temporal Constraint Networks

We first recall the basics of TCNs (Dechter et al., 1991) to model the TCSP (the former acronym refers to the graph representation while the latter refers to the problem in terms of CSP). In the last published work (Schwalb and Dechter, 1997) on that topic, a distinction is made between the merely Quantitative TCN and more general (quantitative and interval-based qualitative) TCNs. It should be clearly stated that although we are only dealing here with Quantitative TCNs, we will use the acronym TCN for the sake of simplicity and by analogy with the original model (Dechter et al., 1991).

At the qualitative level, we rely on the time-point continuous algebra (Vilain et al., 1989), where time-points are related by relations, which can be be represented by a graph where nodes are time-points and edges correspond to precedence relations (Ghallab and Mounir-Alaoui, 1989). It might be interesting to distinguish between a strict precedence, noted $\prec$, and a non-strict one, noted $\preceq$ (i.e. simultaneity of both time-points is allowed). We can use the same time-point graph to represent quantitative constraints as well, thanks to the TCN formalism (Schwalb and Dechter, 1997). Here continuous binary constraints restrict the possible durations between the occurrences of two time-points by means of temporal intervals. A basic constraint between x_i and x_j is $l_{ij} \leq (x_j - x_i) \leq u_{ij}$ equally expressed as $c_{ij} = [l_{ij}, u_{ij}]$ in the TCN. TCSPs allow disjunctions of such expressions, which results in disjunctions of intervals in the TCN. The *STP* (Simple Temporal Problem) restriction applies when disjunctions are not permitted. A TCSP is said to be consistent if one can CHOOSE for each time point a value such that all the constraints are satisfied, the resulting instantiation being one *solution* of the TCSP. Consistency checking of a general TCSP is NP-complete, but for some restrictions such as the STP, polynomial-time algorithms are complete (e.g. the path-consistency algorithm PC-2 (Mackworth and Freuder, 1985), used *incrementally* in planning or scheduling for instance, i.e. at each addition of a new constraint).

2.2 The STPU model

Considering domains in which an "agent" has to perform some "activities", then the TCSP and STP models suit well the cases in which effective durations are under the control of the agent. But this may not arise in realistic applications in which some durations are only "observed", and hence can be considered as being provided by the "external world". Then the problem has to be redefined as follows in this subsection.

Let us start with some notations:

- The N time-points of the constraint network will be noted either b_i or e_i, according to their type (see below), with $i \in \{1, 2, \ldots N\}$. When a time-point might be of any type, we will use the notation x_i.
- A constraint relating two time-points x_i and x_j will henceforth be noted either c_{ij} or g_{ij}, according to its type (see below), and r_{ij} will be used when it can be of any type.

Definition 2.1 (Types of constraints/variables)

The activated *time-points b_i are those whose date is assigned by the agent.*

The received *time-points e_i are those whose unpredictable date is assigned by the external world.*

A free constraint c_{ij} *(referred as* Free *in the following) is a numerical constraint of type $(x_j - x_i) \in [l_{ij}, u_{ij}]$, whose effective duration will be assigned* by the agent *out of the domain $[l_{ij}, u_{ij}]$.*

A contingent constraint g_{ij} *(referred as* Ctg *in the following) is a numerical constraint of type $(e_j - b_i) \in [l_{ij}, u_{ij}]$, whose effective duration will be assigned* by the external world *out of the domain $[l_{ij}, u_{ij}]$, and that is further constrained to be such that $l_{ij} > 0$.*

So a Ctg encompasses the notion of some activity whose (non-null) duration is contingent. In planning for instance, it might be a task of the agent whose precise duration depends on unpredictable conditions, only known at execution time, or the date of some expected event which is not under the control of the agent. Hence the "observation" of the effective duration must be associated to the reception of some "ending" event. This is reflected in the definition since a Ctg always relates two strictly ordered time-points, the second one being necessarily a *received* one (e_j). Notice as well that the first one is constrained to be an *activated* time-point (b_i), though this last restriction is not mandatory, since we might wish to connect two tasks one just after the other, and hence the starting point of the second one should be a received time-point (the end of the first one). But we preferred to keep the intuitive notion that a task is always activated at some point (even though one has no real choice on this activation). Not only does this allow us to

lighten the definitions that will be given further, but also it does not restrict the expressiveness at all: the case given above is easily represented introducing a Free betwen the two tasks whose duration is constrained to equal 0.

One should notice that the definition enforces the basic expected property that two activated time-points can only be related by a Free.

The definition also means that the domain of a *Free* can be reduced by propagation (removing values that are inconsistent with other contraints (Dechter et al., 1991)) while the domain of a *Ctg* should NOT (as it would remove possibly occurring values). Anyway, considering restriction of contingent domains as a failure (as in (Dorn, 1994), where similar distinctions are made with respect to Allen's algebra (Allen, 1983)) is not sufficient as a satisfiability checking process, as it will be illustrated in next section.

The previous distinctions allow us to define a new model called *Simple Temporal Problem under Uncertainty (STPU)*. We should once again distinguish between this new kind of problem and its associated graph-based representation, that we wish to call *Contingent TCN*. Anyway, we will in the following for the sake of simplicity only refer to the term STPU, even when focusing on purely graph-based notions.

Definition 2.2 (STPU)
$\mathcal{N} = (X_b, X_e, R_g, R_c)$ *represents a* STPU *with*
$X_b = \{b_{i_1}, \ldots, b_{i_B}\}$*: set of the* B *activated time-points,*
$X_e = \{e_{i_1}, \ldots, e_{i_E}\}$*: set of the* E *received time-points,*
$N = B + E$,
$R_c = \{c_{i_1 j_1}, \ldots, c_{i_C j_C}\}$*: set of the* C *Frees,*
$R_g = \{g_{i_1 j_1}, \ldots, g_{i_G j_G}\}$*: set of the* G *Ctgs, whith* $\forall g_{ij} = [l_{ij}, u_{ij}], l_{ij} > 0$.

Please notice that we will simplify the notations $r_{i_k j_k}$ to either r_{ij} or r_k, anytime it will be suitable and it might be done unambiguously.

The value assigned to an activated time-point will be noted $\delta(b_i)$, and called a *decision*, while the term *observation* will be used to refer to the effective duration of a Ctg, and noted ω_{ij}. One should notice that observing a ω_{ij} amounts to observing the effective date of a received time-point e_j.

3 Consistency revisited: 3 levels of controllability

3.1 Preliminary definitions

In view of what has been introduced so far, we should first define a "situation" in terms of a tuple of observations and a "control sequence" in terms of a tuple of decisions.

Definition 3.1 (Complete/partial assignments)
A control sequence δ *of the STPU is an assignment of the sole activated time-points:* $\delta = \{\delta(b_{i_1}), \ldots, \delta(b_{i_{B'}})\}, B' \leq B$ *represents the tuple* $\{b_{i_1} := \delta(b_{i_1}), \ldots, b_{i_{B'}} := \delta(b_{i_{B'}})\}$. *A control sequence is* complete *if it instantiates all the activated time-points (i.e.* $B' = B$*). Otherwise it is* partial *and constitutes a subset of a complete control sequence.*

Definition 3.2 (Complete/partial situations)
Given that for all $k = 1 \ldots G$, $g_k = [l_k, u_k]$,
$\Omega = [l_1, u_1] \times \ldots \times [l_G, u_G]$ *will be called the* space of complete situations *of the STPU.*
The tuple $\omega = \{\omega_1 \in [l_1, u_1], \ldots, \omega_G \in [l_G, u_G]\} \in \Omega$ *will be called a* complete situation *of the STPU. Any subtuple of a complete situation is called a* partial situation.

In other words, a *complete situation* (often called for short *situation* henceforth) represents one possible configuration of the whole list of Ctgs. As we will argue further, our problem can be interpreted as a "game against Nature", in which case a situation corresponds to one possible strategy carried out by Nature. For any complete situation ω the STPU amounts to a classical STP $\mathcal{N}_\omega$. A solution of this STP is a complete control sequence that satisfies the Frees in this particular situation.

Definition 3.3 (Projection)
For all $\omega \in \Omega$, $\mathcal{N}_\omega$ *is called the* projection *of the STPU* $\mathcal{N}$ *in the situation* ω*, and is constructed by replacing every* Ctg $g_k = [l_k, u_k]$ *in* $\mathcal{N}$ *by the singleton* $g_k = \{\omega_k\}$*, with* $\omega_k \in \omega$.

Property 3.1 *The projection* $\mathcal{N}_\omega$ *of a complete situation* ω *is a classical STP, while the projection of a partial situation remains a STPU.*

The proof is immediate as a constraint in a *projection* is either a *Free* (i.e. the classical form of a STP constraint) or a singleton.

3.2 Strong controllability

Definition 3.4 (Strong controllability)
$\mathcal{N}$ *is* Strongly controllable *iff*
there exists a control sequence $\delta = \{b_{i_1}, ..., b_{i_B}\}$ *such that for all* $\omega \in \Omega, \delta$ *is a solution of* $\mathcal{N}_\omega$.

This first definition of STPU properties tells that the STPU is *Strongly controllable* iff there exists at least one "universal" solution that fits any situation (and that hence, as far as dynamic applications are concerned, might be computed off-line beforehand). We will illustrate this property and the next ones through some small toy examples that may arise in everyday life common-sense planning.

Examples

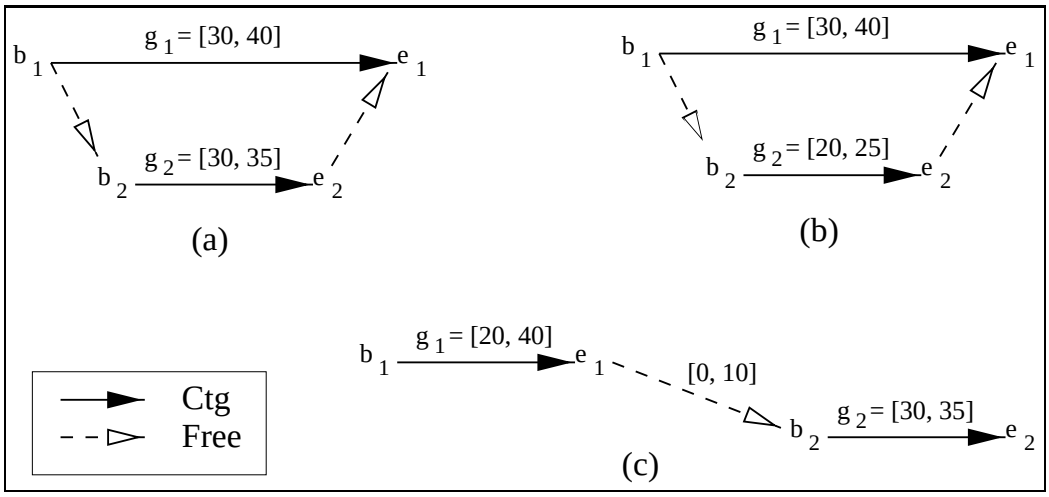

Figure 1: Strong controllability in practice

The two first drawings in Figure 1 show a *during*-like relation (Allen, 1983) between two Ctgs g_1 and g_2. Imagine for instance that you need some food to go and buy downtown (task g_2), knowing that the supermarkets will be closing in 30 up to 40 minutes (contingent interval g_1). Going shopping in supermarket a will take you between 30 and 35 minutes, but if you go to supermarket b it will in any case not take you more than 25 minutes. Then it should not be so hard for you to deduce that going to supermarket b is the only way to be sure to make it on time. In other words, the first case (a) is not Strongly controllable, since there exists at least one situation $\{\omega_1$=30, ω_2=35$\}$ that violates the *during* relation. Notice that nevertheless

no contingent domain has been restricted, which means, as suggested in the previous section, that forbidding to restrict contingent domains is not sufficient as a checking mechanism. The second case (b) is Strongly controllable since deciding for instance to have $b_2 = b_1$ (i.e leaving now) leads to a valid solution whatever values are taken by ω_1 and ω_2.

The third case in Figure 1(c) exhibits a simple *before*-like relation (Allen, 1983) between two Ctgs in which the delay between the reception of e_1 and the activation of b_2 is constrained. Consider for instance that g_1 is the task Cooking and g_2 is the task Having-dinner, and you don't want to eat your dinner cold. The example would be rejected by a *Strong controllability* checking algorithm, since one cannot fix beforehand one unique date for b_2 that will be valid in any situation: the instantiation of b_2 depends on ω_1. Which means that you cannot decide in advance at what precise time you will sit down to table, since it depends on the time you will need to cook: if it takes you only 20 minutes, then you'll necessarily start the dinner within 30 minutes after b_1. But if g_1 takes 40 minutes, then dinner will only begin at least 40 minutes after b_1. But one should notice that this example looks controllable from a planning point of view since one will have no problems deciding *en route* when to activate the second task g_2 once the first one g_1 is achieved. Hence, Strong controllability appears to be a too demanding property, calling for a "weaker" one ...

Anyway, Strong controllabilty may be relevant in specific applications where the situation is not observeable at all or where the complete control sequence must be known beforehands (e.g. when other activities processed by other agents depend on it, which for instance may arise in the production planning area).

3.3 Weak controllability

Definition 3.5 (Weak controllability)
$\mathcal{N}$ *is* Weakly controllable *iff*
for all $\omega \in \Omega$, *there exists* $\delta = \{b_{i_1}, \ldots, b_{i_B}\}$ *such that* δ *is a solution of* $\mathcal{N}_\omega$.

In other words, the STPU will remain *Weakly controllable* as far as in any given situation, there exists at least one solution (which holds in the case of Figure 1(c)). Hence, as soon as one knows the situation, one can pick out and apply the control sequence that matches the situation. But this property proves to be not so useful in classical planning for instance, as we will show it hereafter.

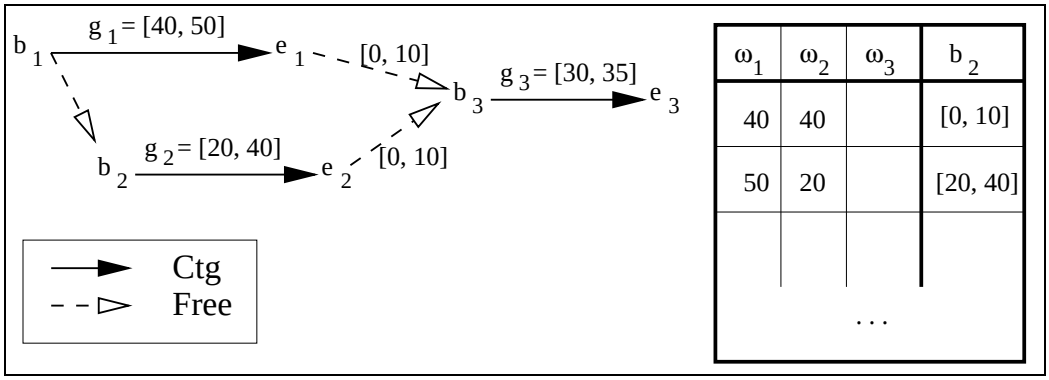

Figure 2: Drawbacks of Weak controllability

Example

Figure 2 exhibits a possible 3-Ctgs network. Consider that you are still faced with your dinner cooking problem, where now g_2 is the task Cooking and g_3 is the task Having-dinner, and there is still the same constraint between them. But you also need to take into account the fact that your children will be back from their football play in about 45 minutes (this is represented by g_1 lasting between 40 and 50 minutes). Not only should you wait for them, but you also know they will be starving, so you shouldn't let them waiting more than 10 minutes before sitting down to table. All durations of the three "tasks" being contingent, the question is: can you be sure to serve a warm dish without delay to your children ?

Let's assume for the sake of simplicity that $b_1 = 0$. To satisfy the two constraints on the activation of b_3, it is enough to ensure that the delay between e_1 and e_2 (however ordered they are) is lower than 10, which can be represented by a constraint [-10, 10]. Then the problem is merely to find a value for b_2 satisfying this constraint. It is then easy to prove that the example is Weakly controllable: to any couple of values assigned to ω_1 and ω_2 (the problem does not depend on ω_3) can be associated at least one value for b_2 such that there is less than 10 minutes between e_1 and e_2. But if we consider it more carefully, we can exhibit two specific cases that appear in the table next to the figure. Those two cases (ω_1=40, ω_2=40) and (ω_1=50, ω_2=20) compel us to choose the value of b_2 within two corresponding ranges of values, that can be obtained through a simple propagation process. Those two ranges [0, 10] and [20, 40] have no common value, which means there is no universal solution and the choice is conditional on the situation (i.e. the problem is not Strongly controllable). The trouble is that the decision to be taken depends upon the observation of ω_1 (i.e. the arrival time of the children)

and ω_2 (i.e. the duration of the dish making), which are necessarily unknown when b_2 is activated. So the "right" decision cannot be taken for sure (i.e. you have to choose between serving a warm dish or serving it quickly, but you cannot be sure of meeting both requirements ...) Therefore in dynamic domains we would like to conclude that the network is "not controllable".

Hence the property that we seek in most dynamic applications is not Weak controllability either. But again, this property may be relevant in specific applications (e.g. in large-scale warehouse scheduling), where the actual situation will be totally "observable", i.e. the duration of all the contingent tasks will be precisely known, before (possibly JUST BEFORE) the execution starts, but one wants to know in advance that whatever the situation appears to be, there will always be at least one feasible solution matching this situation .

3.4 Dynamic controllability

As illustrated in last paragraph, in dynamic application domains such as planning, the solution is built in the process of time, as far as one "observes" the situation at hand. Which means that decisions are to be taken although the situation remains partially unknown. Figure 3 illustrates this phenomenon, stating that at each point in time t the situation splits into the *past situation* that is known (represented by $\omega_{\prec t}$) and the *situation to come* that remains unknown (and is still to be picked out from a partial space of possible situations to come $\Omega_{\succ t} \subseteq \Omega$).

What has just been stated informally will now be defined precisely in order to eventually provide the definition of a third level of controllability, taking advantage of the notions of partial situations and partial control sequences introduced earlier.

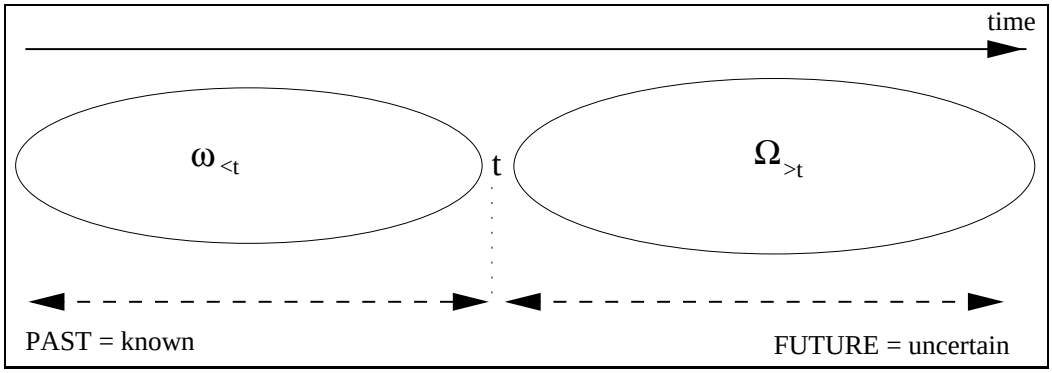

Figure 3: Dynamic observation of the situation

Definition 3.6 (Executed control sequence)

The executed control sequence at t *of the complete sequence δ is the partial sequence $\delta_{\prec t,\delta} \subseteq \delta$ executed so far at date t, defined by:* $\delta_{\prec t,\delta} = \{\delta(b_i)$ *s.t.* $\delta(b_i) \leq t\}$.

Now consider we are at date t where the partial sequence $\delta_{\prec t,\delta}$ has been executed: then obviously only observations related to already executed tasks are known.

Definition 3.7 (Observed situation)

The observed situation at t *associated with $\delta_{\prec t,\delta}$ and ω is the partial situation $\omega_{\prec t,\delta,\omega} \subseteq \omega$ such that:*

$\omega_{\prec t,\delta,\omega} = \{\omega_{ij} \in \omega \ s.t. \ \delta_{\prec t,\delta}(b_i) + \omega_{ij} \leq t\}$

Suppose now that at date t, $\delta_{\prec t,\delta}$ has been executed and $\omega_{\prec t,\delta,\omega}$ has been observed. The space of possible complete situations is then restricted to those that are *compatible* with this "past" observed situation:

Definition 3.8 (Compatible complete situation)

The subspace of complete situations compatible *with a partial situation ω_p is* $\Omega_{\omega_p} = \{\omega' \in \Omega \ s.t. \ \omega_p \subseteq \omega'\}$

Those preliminary definitions being given, we can now propose a definition of *Dynamic controllability*.

Definition 3.9 (Dynamic controllability)

$\mathcal{N}$ *is* Dynamically controllable *iff*

for all $\omega \in \Omega$, there exists $\delta = \{\overline{\delta(b_{i_1})}, \ldots, \delta(b_{i_B})\}$ such that

for all $t \in \delta$, for all $\omega' \in \Omega_{\omega_{\prec t,\delta,\omega}}$, $\exists \delta'$ such that $\delta_{\prec t,\delta} \subseteq \delta'$ and δ' is solution of $\mathcal{N}_{\omega'}$.

In other words, the network is dynamically controllable if and only if we can take successive decisions such that at any point in time, the partial sequence executed so far is ensured to extend to a complete solution whatever durations remain to be observed. This level of controllability suits well dynamic application domains such as planning or scheduling, for which the solution is built in the process of time, as far as one "observes" what occurs: here decisions are taken while the situation is *partially* unknown as opposed (see previous subsection) to Strong controllability that suits cases where the situation is *totally unknown* and to the Weak controllability where the situation is *totally known* when decisions are taken.

Planning is not the only domain where this applies: in multimedia authoring environments research (Jourdan et al., 1997), one needs to make the same distinction betwen contingent and free constraints to model the temporal structure and dynamic presentation of a document, and it appears that

Dynamic controllability is the property that one would like to check (Fargier et al., 1998).

3.5 The basic property relating the three levels

Property 3.2 (Implication rule)
Strong controllability $\Rightarrow$ *Dynamic controllability* $\Rightarrow$ *Weak controllability.*

Proof: The first implication is straightforward: if there exists a "universal" control sequence δ, then one will choose to develop this solution whatever the situation ω is, and hence δ will satisfy the definition of the Dynamic controllability, since at any point in time, the current solution extends to δ itself in any possible situation to come.
The second implication is even more trivial: for any situation ω, if at any point in time a current control sequence can be extended to any compatible situation to come, then it means there exists a solution matching ω itself. $\square$

In other words, when trying to satisfy the Dynamic controllability requirement, checking the Strong one will be a complete but unsound process, while Weak controllability checking will issue sound but uncomplete answers.

4 What makes Dynamic controllability difficult ?

Our intention here is not to make a complexity analysis of the problem, which still remains to be done thoroughly. We rather aim at putting hardness issues into perspective, characterizing our three-fold problem with respect to the area of constraint processing, so as to describe more precisely the nature of each problem and hence argue why Dynamic controllability is expected to be a really hard problem.

4.1 Basic principles of constraint satisfaction search

This part is clearly not intended as an exhaustive survey, but we wish to exhibit features of complementary approaches that will act as standpoints for analysing our problem. We hence assume that the reader is somehow already familiar with the concepts recalled hereafter (some introductory references will be given anyway).

Reasoning with constraints is a concern that emerges in such fields as Artificial Intelligence, Operational Research or Logic Programming. *Constraint*

Programming is a unified term for referring to languages and systems that incorporate *constraint solvers*. They rely either on built-in algorithms for search (see below), like ILOG Solver for instance, or are built on top of a logic programming language, hence using its specific search engine: there one will use the term *Constraint Logic Programming* (CLP) (Frühwirth et al., 1992). Constraint Programming is generally well-suited either for *modelling complex systems* or for *solving combinatorial problems* (van Hentenryck, 1995), the latter being the framework in which our problem lies, and on which we will hence focus. We are interested here in the task of the constraint solver, relying on *constraint satisfaction* (Kumar, 1992) techniques.

As far as resolution is concerned, one has to distinguish between 3 types of problems to solve:

1. Proving the existence of a solution, referred to as *satisfiability* or *global consistency*;
2. Finding one (possibly optimal) solution;
3. Finding all solutions.

We will not consider optimization problems in this paper. Point (1), i.e global consistency (or consistency for short), can in some cases be solved by simple propagation techniques, for instance in the STP model, by the succesful computation of the minimal network (see 2.1), that constitutes a so-called *solution form*. In most cases where consistency checking is not as straightforward, it remains an NP-complete problem that can be solved through point (2), i.e. finding at least one solution using some *enumeration* process which interleaves

- a non-deterministic *backtrack* search process: the search space is a tree, with choice points as nodes, on which a heuristic search is run, developping the tree through successive variable assignments, backtracking upon failure, until a consistent leaf (i.e. a solution) is reached, otherwise inconsistency is concluded;
- and a deterministic constraint propagation (or *filtering*) process that allows to prune this search space and then converge more quickly to a solution. There the *incremental* behaviour of the constraint propagation process becomes a key requirement.

Hence the classical *generate and test* process in Logical Programming is replaced by a *constrain and generate* one. Finding all solutions (3) is a more complex process since backtrack is performed even after success in order to find the next solution, until all the search space has been exhausted.

But enumeration assumes the existence of some kind of disjunctions in the description of the variables [resp. constraints], so as to account for choice

points. It is therefore particularly well-suited to discrete constraint domains like integers, or Finite Domain in CHIP. In the temporal reasoning area, the time line is basically a continuous domain, but constraints exhibit disjunctions providing straightforward choice points. For instance $\{\leq, \geq\}$ in time-point algebra (Vilain et al., 1989), $\{\mathtt{m}, \mathtt{m}_i\}$ in interval algebra (Allen, 1983), [10,20]∪[30,40] in Quantitative TCNs (it can be noticed here that the search tree is such that each leaf is a STP whose satisfiability is then solved by local consistency enforcement) (Schwalb and Dechter, 1997).

Unfortunately this is not the case for our Dynamic controllability problem in STPUs for which we are still faced with continuous constraint domains with no possible discretized choice points. We are going to develop this statement in the following subsections, reviewing the three controllability properties one by one.

But before that let us make a last consideration, whose development is out of the scope of this paper, but which will guide the following discussion. There is a well known relationship between constraint satisfaction and the Satisfiability problem (SAT) in Boolean logic (see for instance (Mackworth, 1992)). The definitions given in the last section give evidence of the interleaving of existential and universal quantifiers in the controllability properties, which a priori suggests a correspondance with the upper class of Quantifier Boolean Formulas (QBF), or at least QBF 3SAT (see (Garey and Johnson, 1979) for details), which are shown to be PSPACE-complete problems ...

4.2 Why is Strong controllability tractable ?

We will give in section 5 a complete algorithm for checking Strong controllability, that will be proven to be polynomial. Our intention here is merely to show that tractability could have been expected from the definition of Strong controllability itself.

If we go back to the simple example of Figure 1(b), it can be expressed in a logical form in the following way:

$$\exists b_1, b_2 : P(b_1, b_2) \wedge Q(b_1, b_2), \text{ where}$$
$$P(b_1, b_2) \equiv b_1 \leq b_2$$
$$Q(b_1, b_2) \equiv \forall \omega_1 \in [30, 40], \forall \omega_2 \in [20, 25], b_1 + \omega_1 \geq b_2 + \omega_2$$

It is easy to see in this example that the second logical expression can be simplified in the following way: the inequation is true for any values of ω_i if and only if it is true in the worst possible case, i.e. when $\omega_1 = 30$ and $\omega_2 = 25$. Then we get the formula $b_1 + 5 \geq b_2$, hence getting rid of the

universal quantifier and giving back a simple interval constraint: combining P and Q produces $0 \leq b_2 - b_1 \leq 5$.

This process can be generalised, as we will see it in section 5, and results in expressing any set of relations between two Ctgs g_1 and g_2 by a simple relation involving only b_1, b_2, ω_1 and ω_2, that we will note here $R_{1,2}(b_1, b_2, \omega_1, \omega_2)$.

Then the definition of Strong controllability can be rewritten in the following informal way:

$$\exists b_1, \ldots, b_B : \forall \omega_1, \ldots \omega_G, R_{1,2}(b_1, b_2, \omega_1, \omega_2) \wedge R_{1,3}(b_1, b_3, \omega_1, \omega_3), \ldots$$

Since we have only conjunctions of relations $R_{i,j}$, we can distribute the universally quantified ω_i on those relations, only keeping for each $R_{i,j}$ those ω_i and ω_j involved in the relation, which gives:

$$\exists b_1, \ldots, b_B : (\forall \omega_1, \omega_2, R_{1,2}(b_1, b_2, \omega_1, \omega_2)) \wedge (\forall \omega_1, \omega_3, R_{1,3}(b_1, b_3, \omega_1, \omega_3)) \wedge \ldots$$

This amounts to the conjunction of local satisfiability checks, solved as in the example above. We will get into more details in section 5, but one can simply see here that from the definition of Strong controllability, *since existential quantifiers remains on the left side of the logical expression, and universal quantifiers on the right side*, this latter can then be recomputed in the shape of simple constraints, and the problem made equivalent to a simple TCN consistency problem, solvable through a no-search propagation process.

4.3 Why is Weak controllability (not so) difficult ?

Conjecture 4.1 (Complexity of Weak controllability)
Checking the Weak controllability is Co-NP-complete.

Proof: The Co-problem of checking Weak controllability is: is there a situation $\omega \in \Omega$ such that $\mathcal{N}_\omega$ is an inconsistent STP ? Since checking that a STP is inconsistent is a polynomial problem, this co-problem belongs to NP. Hence, Weak controllability belongs to Co-NP. The difficulty of the problem (Co-NP-*complete*) remains to be proven.

Why should it be "only" Co-NP-complete and not PSPACE-complete as interleaving of existential and universal quantifiers would suggest it ? The following properties will allow us to answer this question.

Property 4.1 *if $\mathcal{N}_{\{\omega_1,\ldots,l_k,\ldots,\omega_G\}}$ and $\mathcal{N}_{\{\omega_1,\ldots,u_k,\ldots,\omega_G\}}$ are consistent STPs, then for any $v_k \in [l_k, u_k]$, $\mathcal{N}_{\{\omega_1,\ldots,v_k,\ldots,\omega_G\}}$ is a consistent STP.*

Proof: Let us consider the partial projection $\mathcal{N}_{\omega \setminus \omega_k}$ where all Ctgs but g_k have been instantiated. Then let us assume that the two completed projections with $\omega_k = l_k$ and $\omega_k = u_k$ are consistent STPs and that the completed projection with $\omega_k = v_k$ is an inconsistent STP. Then it means the constraint $l_k \leq \omega_k \leq u_k$ was necessarily inconsistent in $\mathcal{N}_{\omega \setminus \omega_k}$, which in turn means that either $l_k \leq \omega_k$ or $\omega_k \leq u_k$ is inconsistent, and hence either $\omega_k = l_k$ or $\omega_k = u_k$ must be inconsistent, which contradicts the assumptions. □

Property 4.2 (Weak controllability on bounds) *A STPU is Weakly controllable iff for any* $\omega^{bnd} \in \{l_1, u_1\} \times \ldots \times \{l_G, u_G\}$, $\mathcal{N}_{\omega^{bnd}}$ *is a consistent STP.*

Proof: This follows directly from the former proposition: Weak controllability is equivalent to "all the possible projections of $\mathcal{N}$ are consistent", and the given proposition informs us that it is enough to consider projections on the bounds to be sure that all intermediate projections will be consistent as well. □

What lessons can be learned from this ? Originally we have an expression to solve where *universal quantifiers are in the left side and existential ones in the right side*. The problem here is that universal quantifiers apply to continuous constraints, which we overcome thanks to the property that allows us to restrict ourselves to the bounds. This corresponds to a *discretization*, which introduces disjunctions and hence choice points. Since universal quantifiers are confined on the left side of the Weak controllability expression, this discretization provides as leaves in the search tree expressions in which only existential quantifiers remain, which corresponds to STP consistency problems. This suggests a classical enumerative process (see 4.1), together with look-ahead techniques, which will be presented in section 5. Since one needs to exhaust the search so as to look for all "bound" situations ω^{bnd}, this process should be related to *finding all solutions*. Alhough this is exponential in the worst case, it constitutes anyway a nicer result than the PSPACE-completeness expected from QBF problems.

Illustrating this through the example of Figure 1(c) gives the following steps, first removing ω_2 that is not involved, then restricting the condition to the bounds of ω_1:

$$\begin{aligned}
&\forall \omega_1 \in [20, 40], \forall \omega_2 \in [30, 35], \exists b_1, b_2 : 0 \leq b_2 - e_1 \leq 10 \\
&\models \forall \omega_1 \in [20, 40], \exists b_1, b_2 : 0 \leq b_2 - b_1 - \omega_1 \leq 10 \\
&\models \forall \omega_1 \in \{20, 40\}, \exists b_1, b_2 : 0 \leq b_2 - b_1 - \omega_1 \leq 10 \\
&\models ((\omega_1 = 20) \wedge (20 \leq b_2 - b_1 \leq 30)) \vee \\
&\quad ((\omega_1 = 40) \wedge (40 \leq b_2 - b_1 \leq 50))
\end{aligned}$$

$\models \mathtt{T}$

4.4 Why is Dynamic controllability really hard ?

We argued in last section that Dynamic controllability was expected to be of high theoretical complexity in the general case. Although this remains to be thoroughly proven, we will show here why we shouldn't be surprised by this result.

If we consider the example of Figure 2 (which is Weakly but not Dynamically controllable), and still assuming that $b_1 = 0$, we get

$\exists b_2 : \forall \omega_1 \in [40, 50], \forall \omega_2 \in [20, 40], \exists b_3 : 0 \leq (b_3 - b_2 - \omega_2) \leq 10$ and $0 \leq (b_3 - \omega_1) \leq 10$

Here we have

- *interleaving of universal and existential quantifiers*, which makes it tricky to translate the problem into a classical CSP and use simple propagation, as for Strong controllability: even if the variables are still clear (the activated time-points b_i), what simple constraints could be designed between them ? This feature can be interpreted as a kind of non-linearity of the problem, which prevents us as well to rely on Linear Programming techniques, though those appear to be very powerful in non-contingent temporal reasoning (Jonsson and Bäckström, 1996).
- On the other hand, there is no obvious way of *discretizing* the problem to yield choice points and hence apply classical branching search techniques, as for Weak controllability. This makes a difference with previous problems and explains why this one is really difficult, and is expected to lie in a class higher than NP (the comparison with QBF suggesting at most PSPACE-Completeness).

There is actually one way of discretizing the problem, which is simply through a discretization of time, with some suitable granularity (for instance restricting the constraint domains to the set of integers, each time unit possibly representing a second, a minute, etc). Moreover the interleaving of observations and activations suggests to rely on game-based search, the problem being a problem of *game against Nature* for a player being for instance the agent executing a plan or a schedule. This approach is going to be presented in section 5.

Before that, as a matter of conclusion for this section, it may be worth mentioning the usual theoretical complexity classification that divides constraint satisfiability into (1) *no-search* processes (complete propagation) lying

in the class P, (2) *constrain and generate* processes in the class NP and (3) *game search* or *theorem proving* in higher complexity classes. The contribution of the study carried out in this section was to give evidence of how this three-fold distinction suits well our three controllability problems, giving hence a clear picture of their severely distinct natures.

5 Algorithmic issues

5.1 Checking Strong controllability

An algorithmic method has been proposed in a previous work (Vidal and Ghallab, 1996) for ensuring Strong controllability of a STPU involving only Allen-like qualitative constraints as Frees between tasks (actually simple precedence of the form $x_i - x_j \geq 0$). This approach can be extended to any kind of STPU: our algorithm relies on the fact that the definition of Strong controllability can be rewritten in terms of satisfaction of the Frees, and then one can always reverse the universal quantifiers relating to the situations and to the Frees:

Property 5.1 $\mathcal{N}$ *is* Strongly controllable
iff $\exists \delta$ *such that* $\forall \omega \in \Omega$, $\forall c_k \in R_c$, (δ, ω) *satisfies* c_k
iff $\exists \delta$ *such that* $\forall c_k \in R_c$, $\forall \omega \in \Omega$, (δ, ω) *satisfies* c_k.

Then we can merge all the possible situations into a more compact description of the constraints. Let us first notice that a Free $(x_i - x_j) \in [a,\ b]$ can be equivalently represented by two Frees $(x_i - x_j) \geq a$ and $(x_i - x_j) \geq -b$. Up to this transformation, there are four types of Frees, depending on what kind (*begin/activated* or *end/received*) of points they relate. Introducing explicitly the contingent durations, the STPU can be rewritten as a system of disequations involving only activated time-points and durations (we will note here for the sake of simplicity ω_i the duration of the Ctg g_i activated at time-point b_i):

- Free of type 1 (*begin-begin*): $b_i - b_j \geq a$;
- Free of type 2 (*begin-end*): $b_i + \omega_i - b_j \geq a$;
- Free of type 3 (*end-begin*): $b_i - b_j - \omega_j \geq a$;
- Free of type 4 (*end-end*): $b_i + \omega_i - b_j - \omega_j \geq a$;
- Ctg: $u_i \geq \omega_i \geq l_i$.

Now, since a constraint has to be satisfied whatever the situation is, it has to be satisfied in the worst case:

- A constraint of type 1, i.e. $b_i - b_j \geq a$ does not depend on the situation;

- A constraint of type 2, i.e. $b_i + \omega_i - b_j \geq a$ must be satisfied whatever ω_i is. It is easy to see that $b_i + \omega_i - b_j \geq a$ is satisfied whatever $\omega_i \in [l_i, u_i]$ is iff it is satisfied when $\omega_i = l_i$;
- A constraint of type 3, i.e. $b_i - b_j - \omega_j \geq a$ is satisfied whatever $\omega_j \in [l_j, u_j]$ is iff it is satisfied when $\omega_j = u_j$;
- A constraint of type 4, i.e. $b_i + \omega_i - b_j - \omega_j \geq a$ is satisfied whatever $\omega_i \in [l_i, u_i]$ and $\omega_j \in [l_j, u_j]$ are iff it is satisfied when $\omega_j = u_j$ and $\omega_i = l_i$.

Hence, the STPU is strongly consistent iff $\exists \delta = \{\delta(b_{i_1}), \ldots, \delta(b_{i_B})\}$ that satisfies the following sets of constraints:

- The original set of constraints of type 1;
- A set of constraints corresponding to the original constraints of type 2 and of the form: $b_i - b_j \geq a - l_i$;
- A set of constraints corresponding to the original constraints of type 3 and of the form: $b_i - b_j \geq a + u_j$;
- A set of constraints corresponding to the original constraints of type 4 and of the form: $b_i - b_j \geq a - l_i + u_j$.

This system of disequations involves activated time points and defines a classical STP, which is consistent iff the STPU is strongly controllable (see previous work). Moreover, any solution of the STP is a control sequence of the STPU. The algorithm `StrongControllability(STPU)` (Vidal and Ghallab, 1996) is hence straightforward and consists only in (1) a linear time translation of the original constraints into constraints between mere activated time-points, and (2) the application of a classical polynomial-time propagation algorithm (e.g. Floyd-Warshall or PC-2) to the resulting STP. It obviously follows that:

Property 5.2 *Checking Strong controllability is polynomial.*

5.2 Checking Weak controllability

We have seen in previous section that Weak controllability was expected to be co-NP-Complete, and we have given the fundamental property allowing us to design some enumerative algorithm, which was only suggested. This section is aimed at providing the complete construction of such an algorithm.

This will be a classical tree search process, where choice points are assignments of one bound to a Ctg. As usual, one wants to prune this search space through incomplete propagation steps run at each choice point. For instance, the STPU resulting from the choices of bounds made so far can be considered as a classical STP, and if a simple propagation fails, this implies

that the corresponding STPU has no chance to be weakly controllable, and it is not worth developping the tree further: backtrack can then be processed. The following `SimplePropagation` procedure can be designed for that purpose, and then used by the next `WeakControllability` checking recursive algorithm:

```
SimplePropagation(STPU: in out)
    Apply PC2 to STPU and stop it as soon as a Ctg is modified
    if any Ctg of STPU has changed then return false
    else return true.

WeakControllability(STPU, R_g:  in)
    if not SimplePropagation(STPU) then return false
    else if STPU is strongly controllable then return true
        else if R_g = {} then return false
            else
                choose and remove a g_i from R_g
                replace g_i by {l_i}
                if not WeakControllability(STPU, R_g) then return false
                else
                    replace g_i by {u_i}
                    return WeakControllability(STPU, R_g)
```

Notice that when all the Ctgs have been instantiated, the STPU becomes a classical STP and Weak controllability is equivalent to classical consistency. In this sketch of algorithm, `SimplePropagation` is called from scratch. Obviously, incremental calls taking into account the last modifications of the STPU will be used in practice.

In classical CSPs, necessary conditions of consistency are defined using the notion of k-consistency (Freuder, 1978). In short, a CSP is k-consistent if and only if any sub-CSP that can be defined from it and that contains k variables is consistent. In the same way, we can define a property of k-Weak controllability, that is a necessary condition of Weak controllability:

Definition 5.1 (k-Weak controllability)

A k-subproblem of a STPU is a STPU involving k of the Ctgs of the original problem and all the Frees that relate the activated and received time-points of these Ctgs.

A STPU is k-Weakly controllable iff all its k-subproblems are Weakly controllable.

Classical CSPs use constraint propagation algorithms for ensuring k-

consistency. In the same way, we can design a polynomial propagation algorithm for k-Weak controllability. For instance, 3-Weak controllability can be ensured as follows, the provided algorithm being a simple version; a more efficient one, handling in a more clever way the list Q as in PC2 can be easily designed but will not be reported here for the sake of clarity. The new (still incomplete) `3WeakClbPropagation` algorithm replaces the previous `SimplePropagation` one, with the advantage that a higher degree of "local controllability" is ensured, still low enough (k=2 or 3 are always used in practice (Tsang, 1993)) to account for a cheap cost, hence providing the best compromise between propagation and search (see 4.1).

`3WeakClbPropagation(STPU)`
SimplePropagation($STPU$)
$Q := \{(i,j,k)$ such that g_i, g_j and g_k are Ctgs of $R_g\}$
$weak :=$ true
while Q is not empty and $weak$ **do**
choose and remove a (i,j,k) from Q
$STPU'$ is the 3-subproblem of $STPU$ involving g_i, g_j and g_k
for all $(\omega_i, \omega_j, \omega_k) \in \{l_i, u_j\} \times \{l_j, u_j\} \times \{l_k, u_k\}$ **do**
$STP_{(\omega_i,\omega_j,\omega_k)} := STPU'_{(\omega_i,\omega_j,\omega_k)}$
ensure the 3 Consistency of $STP_{(\omega_i,\omega_j,\omega_k)}$
if any of the 6 $STP_{(\omega_i,\omega_j,\omega_k)}$ is inconsistent **then** $weak :=$ false
else
for any Free C_h of $STPU'$ **do**
$l_h := min_{STP_{(\omega_i,\omega_j,\omega_k)}} l_{h_{(\omega_i,\omega_j,\omega_k)}}$
$u_h := max_{STP_{(\omega_i,\omega_j,\omega_k)}} u_{h_{(\omega_i,\omega_j,\omega_k)}}$
$weak :=$ SimplePropagation(STPU)
add to Q all the (i',j',k') such that some Frees relating Ctgs
$g_{i'}$, $g_{j'}$ and $g_{k'}$ have been modified.
return $weak$

where $C_{h(\omega_i,\omega_j,\omega_k)} = [l_{h_{(\omega_i,\omega_j,\omega_k)}}, u_{h_{(\omega_i,\omega_j,\omega_k)}}]$ is the Free of $STP_{(\omega_i,\omega_j,\omega_k)}$ that relates the same points as C_h.

5.3 Checking the Dynamic controllability

At first sight, checking the Dynamic controllability of a STPU appears to be an intractable task: implicitly, the Dynamic controllability property involves an alternance of existential and universal quantifiers that suggests a complexity level in the polynomial hierarchy (Garey and Johnson, 1979), namely

the PSPACE class of problems. As a matter of fact, the problem can be understood as a "game against Nature" where the first player (the "agent") gives the begin times of tasks while the second one (Nature) provides the end of those activated tasks.

This section will give the very first algorithm for Dynamic controllability checking, based on a classical alpha-beta scheme for two-player games (see e.g. (Pearl, 1984)). Let us call $b_0 \in V_b$ the time-point corresponding to the origin of time, and $c_{i,0}$ the Free between b_0 and b_i, i.e. the date of b_i. The set of time-points to activate is the set of beginning points of Ctgs. The set of time points to receive, initially empty, is updated as soon as the beginning of a Ctg is activated.

```
Play(STPU:in)
    toActivate := {};
    for all g_i in R_g do
        add c_{i,0} : b_i − b_0 ≥ 0 to R_c
        add (b_i, t_i := 0) to toActivate;
    toReceive := {};
    α := 0;      β := 1;
    if AgentPlays(STPU, toReceive, toActivate, α, β) > 0 then success;
```

When the agent has to play, it computes the set of points that must be activated (N) and the set of points he can activate (Π). Similarly, Nature computes the set of points that must be received (N) and the set of points that can be received (Π). In both cases, a choice contains all the points of N and a subset of the time points of Π (one should notice that this implicitly enforces the possible simultaneity of events, which obviously must be permitted in the application domains we are interested in). After each choice, the network is updated (modification of the Frees relating the chosen time-point to b_0 when the agent plays, modification of the chosen Ctgs when Nature plays). The next player then "filters" the network using Strong and Weak controllability propagation procedures (see previous section): if the network is not weakly controllable, then it is not dynamically controllable, i.e. the agent necessarily loses in the current situation. On the other hand, if the network is strongly controllable, then it is dynamically controllable, and the agent necessarily wins in the current situation. Notice that an empty choice (denoted $\emptyset$) may be possible: this represents for instance a case where the agent waits for an observation before starting a task. Anyway, when no observation is waited for (i.e. when *toReceive* is empty), the agent must play as soon as he can (the time counters are accelerated until this date).

```
GamePropagation(STPU: in out; end, win:  out)
    weak := 3WeakClbPropagation(STPU)
    if weak then
        win := StrongControllability(STPU)
        end := win
    else end := true

AgentPlays(STPU, toReceive, toActivate, α, β:  in)
    GamePropagation(STPU, end, win);
    if end then      if win then α := 1 else β := 0;
    else
        if toReceive := {} then      — — acceleration of the counters
            δ := min_i(l_{i,0} − t_i);
            for all (b_i, t_i) ∈ toActivate do t_i := t_i + δ;
        Π := {(b_i, t_i) ∈ toActivate such that l_{i0} ≤ t_i ≤ u_{i0}}
        N := {(b_i, t_i) ∈ toReceive such that t_i = u_{i0}}
        A := {E such that E = N ∪ P and P ⊆ Π}
        if toReceive := {} then remove ∅ from A
        While A ≠ {} and α < β do
            STPU' := STPU;
            toReceive' := toReceive;
            toActivate' := {};
            choose and remove a choice C from A;
            for all (b_k, t_k) in toActivate do
                if (b_k, t_k) ∈ C
                    then c'_{k0} := [t_k, t_k]
                    else add (b_k, t_k + 1) in toActivate'
            α := max(α, naturePlays(STPU', toReceive', toActivate', α, β)
    return α
```

```
NaturePlays(STPU, toReceive, toActivate, α, β:  in)
    GamePropagation(STPU, end, win);
    if end then      if not win then β := 0 else β := 1;
    else
        Π := {(e_i, t_i) ∈ toReceive such that l_i ≤ t_i ≤ u_i}
        N := {(e_i, t_i) ∈ toReceive such that t_i = u_i}
        A := {E such that E = N ∪ P and P ⊆ Π}
        While A ≠ {} and α < β do
            choose and remove a choice C from A;
            STPU' := STPU;
            toActivate' := toActivate;
            toReceive' := {};
            for all (g_i, t_i) in toReceive do
                if (g_i, t_i) ∈ C
                    then g_i := [t_i, t_i]
                    else add (e_i, t_i + 1) to toReceive'
            β := min(β, agentPlays(STPU', toReceive', toActivate', α, β))
    return β;
```

This algorithm can be enhanced in several ways. First, the propagation procedures must obviously be called in an incremental way (propagation starts from the knowledge of the last modifications of the STPU). Even more important is the need to define a function that estimates the interest of the current situation for the agent: as in classical games, it will be called as soon as a given level of recursive calls has been reached. We will then obtain some substantial gain in efficiency. This will of course be counter-balanced by a loss: the resulting algorithm will be incomplete, but how could one avoid it ? Dynamic controllability is indeed an untractable problem, as any game search problem, traditionally lying in the class of PSPACE-Complete problems. It is hence more realistic to manage to find incomplete but tractable solutions, still providing a good estimate of the overall feasability, together with a secure complete Dynamic controllability checking in the short-term, that can be reevaluated during execution: our algorithm can obviously be tailored according to those requirements.

Finally, one of the most important drawback of this algorithm is that Time is explicitly discretized. This leads to develop branches that are very similar (and in particular branches where the two players do nothing). The idea is to replace the explicit representation of a discretized time by the notion of sequence of decisions / observation: the actual decision of the agent will be to wait or not to wait an observation before starting a task. This can easily be made using the underlying symbolic structure of the tempo-

ral constraint network, through the notion of *successors* of a time-point (see (Ghallab and Mounir-Alaoui, 1989; Ghallab and Vidal, 1995) for a description of this structure).

6 Tractable equivalence classes

Assuming that Weak and Dynamic controllabilities are more than polynomial, and that the conjecture NP $\neq$ P is true, then it is always useful to track maximal restricted representations in which the problem becomes polynomial again (Nebel and Bürckert, 1995; Bessière et al., 1996; Drakengren and Jonsson, 1997). In our framework, it amounts to finding the maximal equivalence classes between Weak [resp. Dynamic] and Strong controllabilities.

We will only give in this section some first hints on how to build such classes, since this study is still the subject of on-going work.

6.1 Weak / Strong equivalence class

We have mainly focused on the Weak $\equiv$ Strong equivalence classes, still relying on our definition of a precedence constraint between time-points as being $\preceq$ (before or equals) (Vilain et al., 1989).

Definition 6.1 (Upper / Lower Ctgs)
A Ctg $g_i = [l_i, u_i]$ *is said to be* lower *iff* $\forall \delta$ *solution of* $\mathcal{N}_{\{\omega_1,\ldots,l_i,\ldots,\omega_G\}}$, δ *is also a solution of* $\mathcal{N}_{\{\omega_1,\ldots,u_i,\ldots,\omega_G\}}$.
A Ctg $g_i = [l_i, u_i]$ *is said to be* upper *iff* $\forall \delta$ *solution of* $\mathcal{N}_{\{\omega_1,\ldots,u_i,\ldots,\omega_G\}}$, δ *is also a solution of* $\mathcal{N}_{\{\omega_1,\ldots,l_i,\ldots,\omega_G\}}$.

Recall (section 2) that a Ctg always relates an activated time-point ("begin") to a received one ("end"), and that hence two activated [resp. received] time-points will always be related by a Free. A Ctg is always of the form:

$e_i - b_i = \omega_i$ with $\omega_i \in [l_i, u_i]$

And we only have 3 kinds of Frees:

$c_k : b_j - b_i \in [l_k, u_k]$
$c_k : b_j - e_i \in [l_k, u_k]$ i.e. $b_j - b_i - \omega_i \in [l_k, u_k]$
$c_k : e_j - e_i \in [l_k, u_k]$ i.e. $b_j + \omega_j - b_i - \omega_i \in [l_k, u_k]$

A Ctg g_i is typically *lower* when any Free related to it has one of the following forms:

$c_k : b_j - b_i \in [l_k, u_k]$
$c_k : b_j - e_i \in (-\infty, u_k]$
$c_k : e_j - e_i \in (-\infty, u_k]$

And g_i is typically *upper* when any Free related to it has one of the following forms:

$$c_k : b_j - b_i \in [l_k, u_k]$$
$$c_k : b_j - e_i \in [l_k, +\infty)$$
$$c_k : e_j - e_i \in [l_k, +\infty)$$

This allows us to exhibit the following characterization of the polynomial classes $PWeak_{\preceq}$ entailed by an equivalence relation with Strong controllability.

Property 6.1 (PWeak$_{\preceq}$ equivalence class)
If all the Ctgs g_i are lower *[resp.* upper*],*
then Strong controllability is equivalent to Weak controllability.

Proof: *Strong controllability* $\Rightarrow$ *Weak controllability* has already been proven in the general case. Conversely, suppose that the STPU is Weakly controllable. Consider the situation ω^* where $\omega_i = l_i$ for any lower g_i and $\omega_j = u_j$ for any upper g_j (for Ctgs which are both upper and lower, one can equally choose l_i or u_i). Since the STPU is Weakly controllable, there exists a solution δ of the STP $\mathcal{N}_{\omega^*}$. It is also a solution of any $\mathcal{N}_\omega$ such that $\omega \in \Omega$, by definition of upper and lower cgts. Hence, the STPU is Strongly controllable.

We finally obtain two distinct classes, that are proven to be maximal. Anyway, this is not going to be reported here any further but is to be developped in a forthcoming paper more devoted on theoretical complexity issues. One should just be aware that expressiveness is much restricted in those classes since for instance the *during* relation must be excluded in both, and in the larger one even the *before* relation has to be discarded.

On another hand, this notion of lower and upper Ctgs is useful to enhance our Weak controllability sketch of algorithm: before enumerating, one can transform the STPU into an equivalent one for Weak controllability replacing any lower *Ctg* g_i [resp. any upper *Ctg*] by the singleton $g_i = \{l_i\}$ [resp. $g_i = \{u_i\}$]. This might also be done during the enumeration process, since some *Ctgs* can become upper or lower once others have been instantiated.

6.2 On-going work

We have begun to investigate the design of maximal tractable subclasses as well for Dynamic controllability, trying to exhibit equivalence classes between it and Strong controllability, but this part of the work is not mature enough to be reported here.

One interesting track of research that we have been starting to look upon consists in redefining the precedence relation between time-points as being now $\prec$ (strictly before) (Vilain et al., 1989). Not only this defines an algebra closer to Allen's one (Allen, 1983), but it seems to permit the design of more interesting classes both for the Weak and the Dynamic controllability, that one could call PWeak$_\prec$ and PDynamic$_\prec$.

7 Conclusion and Future work

This paper deals with notions of controllabilities that replace the classical notion of constraint network consistency when one has to deal with contingent durations. The main contribution is to exhibit three distinct properties, of increasing hardness that have been discussed and studied. The last one, Dynamic controllability, is of particular interest in such application domains as planning, scheduling, supervision or multimedia document management, in which an agent has to perform activities in a dynamic and only partially predictable environment. We have provided complete algorithms for checking Weak and Dynamic controllabilities. It still remains to prove the theoretical complexities of those.

7.1 Discussion about related work

As promised in the introduction, we ought to tell a few words about how our TCN-based work should be compared with another important track of research for dealing with Time, especially in planning (Rutten and Hertzberg, 1993) and scheduling, namely the *Time Map Managers* (Dean and McDermott, 1987). The temporal representation in TMMs is a point-based one close to the STP. But those time-points are defined as *begin* and *end* points of *tokens*, which are basically temporal intervals onto which semantic interpretations are mapped in the shape of propositions that are either true or false along the token. This makes it possible to handle in a unique system temporal, persistence and causal reasoning, while TCNs are designed to merely deal with the temporal constraints of a problem, disregarding its semantics. Hence one can say that TCNs manage partially ordered events (in the sense of an instantaneous discrete event) while TMMs manage the truth of propositions over time. The former will hence be preferred for supervision purpose for instance (Dousson et al., 1993), the latter being well suited for querying temporal databases. Both can be used in planning and scheduling, but our approach deals with the feasibility of the execution of a given plan, while TMMs are used for the whole planning search procedure.

If one considers in more details the work made on the management of ambiguity and possible/necessary truth of facts in β-TMM (Schrag et al., 1992), similarities with our work could be guessed: notions such as *observation*, or *weak* and *strong* properties, are defined in similar ways, but do not refer to the same things. While an observation in (Schrag et al., 1992) refers to the undefeasible knowledge that a proposition holds over some interval, an observation in our case is the time-stamped occurrence of an event. In both cases they are inputs to the system, but in a TMM it is a semantic and static input given beforehand, while in our approach it corresponds to a merely temporal but dynamically acquired knowledge. Then the notion of *possible worlds* in (Schrag et al., 1992) is similar to our *situations* (as far as one considers only the temporal uncertainty aspects): both correspond roughly speaking to distinct possible positioning and total ordering of the contingent temporal entities.

So if the notion of uncertainty (or contingency) is defined similarly, what is definitely different is what one wants to check. In β-TMM, one wants to assess if a proposition *holds* at some given point. Ambiguity from temporal uncertainty and persistence clipping compels to distinguish what *necessarily holds* and what *possibly holds* (such as in the temporal database work made in (van Beek, 1991), in which one distinguishes between what MUST be true and what MAY be true). Answering such queries is NP-complete, hence two tractable approximations are given: the *strong holds* is a sound and incomplete approximation of the *necessarily holds* and the *weak holds* is a complete but unsound approximation of the *possibly holds*. Those definitions are given in a rather different context and are hence distinct from ours, although they sound similar for obvious reasons.

Our problem context is much more related to the one addressed in the *mixed CSPs* (Fargier et al., 1996), in which one has to build a compound decision, taking into account both contingent and decisional variables, and the notions of weak and strong consistency that appear there are very close to and actually inspired ours. Weak consistency is somehow related to "possible" decisions: if one checks the situation (that is called *possible world* here again) THEN there is always a *conditional decision* fitting this situation. Strong consistency somehow corresponds to "necessary" decisions: there must exist one *universal decision* that will fit any situation, so the situation might be observed only AFTER having taken the decision.

So both previous approaches (Schrag et al., 1992; Fargier et al., 1996) have already introduced the concept of contingency. The former is richer than our approach in that it takes into account persistence and causal ambiguity as well, hence contributing to a more complete reasoning system, but it tackles query processes instead of decision problems, hence it deals with contingency

but not with decisional variables. The latter is more expressive from another standpoint since it deals with general constraints and hence general decision problems, not only temporal ones.

But neither of them considered the problem as a dynamic phenomenon: temporal uncertainty should indeed in our case not only be interpreted in terms of incomplete knowledge of the *state* of the world, but also in terms of incomplete knowledge of the *evolution* of the world. This constitutes the main novelty in our problem specifications, and compelled us to introduce this new Dynamic controllability property that lies between the Weak and Strong ones, since here the situation is neither known before nor after the decision is taken, but *while* the decision is gradually taken.

7.2 Future work

The next step will be to apply our methods to specific applications, for instance in multimedia authoring environments (Jourdan et al., 1997), where building a document with a temporal structure is nothing but planning, and browsing such a document is nothing but plan execution (see (Fargier et al., 1998) for a preliminary statement of the problem). The interest of this area is that the degree of parallelism of the temporal networks is most of the time bounded. So we can reasonably expect to get acceptable processing times in spite of the inherent theoretical complexity of the task.

Other on-going work regards the suggested improvements (see previous section) of the game algorithm. One should also keep in mind that when dealing with untractable problems, it is always useful to look for tractable subclasses, which we have started to carry on for Weak and Dynamic controllabilities. This study is also promising to be useful for improving the checking algorithms in the general case.

Lastly, we have begun investigating another research track relying on continuous games in the shape of timed automata (Alur and Dill, 1994), which seems to be better suited for dealing with the usual *incremental* aspect of the plan building process (see (Cervoni et al., 1994; Rutten and Hertzberg, 1993)), i.e. when Dynamic controllability must be checked each time a new set of constraints is added to the network.

References

Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):509–521.

Alur, R. and Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science*, 126:183–235.

Bessière, C., Isli, A., and Ligozat, G. (1996). Global consistency in interval algebra networks: tractable subclasses. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 3–7, Budapest (Hungary).

Brusoni, V., Console, L., Pernici, B., and Terenziani, P. (1994). LaTeR: a general purpose manager of temporal information. In *Methodologies for Intelligent Systems 8, Lecture Notes in Computer Science*, volume 869, pages 255–264. Springer Verlag.

Cervoni, R., Cesta, A., and Oddi, A. (1994). Managing dynamical temporal constraint networks. In *Proceedings of the 2nd International Conference on AI Planning Systems (AIPS-94)*, pages 13–17, Chicago (Il, USA).

Dean, T. and McDermott, D. (1987). Temporal data-base management. *Artificial Intelligence*, 32:1–55.

Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49:1–95.

Dorn, J. (1994). Hybrid temporal reasoning. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94)*, pages 625–629, Amsterdam (Netherlands).

Dousson, C., Gaborit, P., and Ghallab, M. (1993). Situation recognition: representation and algorithms. In *Proceedings of the 13th International Joint Conference on A.I. (IJCAI-93)*, Chambéry (France).

Drabble, B. and Kirby, R. (1991). Associating A.I. planner entities with an underlying time point network. In *Proceedings of the 1st European WorkShop on Planning (EWSP-91)*, pages 27–38, Sankt Augustin (Germany).

Drakengren, T. and Jonsson, P. (1997). Eight maximal subclasses of Allen's interval algebra with metric time. *Journal of Artificial Intelligence Research (JAIR)*, 7:25–45.

Dubois, D., Fargier, H., and Prade, H. (1993). The use of fuzzy constraints in job-shop scheduling. In *IJCAI-93 Workshop on Knowledge-Based Planning, Scheduling and Control*, Chambéry (France).

Fargier, H., Jourdan, M., Layaïda, N., and Vidal, T. (1998). Using temporal constraint networks to manage temporal scenario of multimedia documents. In *ECAI-98 workshop on Spatio-Temporal Reasoning*, Brighton (UK).

Fargier, H., Lang, J., and Schiex, T. (1996). Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In *Proceedings of the 12th National Conference on A.I. (AAAI-96)*, pages 175–180, Portland (Oregon, USA).

Freuder, E. (1978). Synthetizing constraint expressions. *Communications of the ACM*, 21(11):958–966.

Frühwirth, T., Herold, A., Küchenhoff, V., Provost, T. L., Lim, P., Monfroy, E., and Wallace, M. (1992). Constraint Logic Programming — an informal introduction. In L., S., editor, *Notes of the Logic Programming Summer School*, Zürich (Switzerland).

Garey, M. and Johnson, D. (1979). *Computers and intractability — a guide to the theory of NP-Completeness*. W.H. Freeman and cie, San Francisco (USA).

Ghallab, M. and Mounir-Alaoui, A. (1989). Managing efficiently temporal relations through indexed spanning trees. In *Proceedings of the 11th International Joint Conference on A.I. (IJCAI-89)*, pages 1297–1303, Detroit (USA).

Ghallab, M. and Vidal, T. (1995). Focusing on a sub-graph for managing efficiently numerical temporal constraints. In *FLorida A.I. Research Symposium (FLAIRS-95)*, Melbourne Beach (FL, USA).

Jonsson, P. and Bäckström, C. (1996). A linear programming approach to temporal reasoning. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 1235–1240, Portland (OR, USA).

Jourdan, M., Layaïda, N., and Sabry-Ismail, L. (1997). Time representation and management in MADEUS: an authoring environment for multimedia documents. In Freeman, M., Jardetzky, P., and Vin, H., editors, *Multimedia Computing and Networking*, pages 68–79.

Kumar, V. (1992). Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13(1):32–44.

Mackworth, A. (1992). The logic of constraint satisfaction. *Artificial Intelligence*, 58:3–20.

Mackworth, A. and Freuder, E. (1985). The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65–74.

Meiri, I. (1991). Combining qualitative and quantitative constraints in temporal reasoning. In *Proceedings of the 9th National Conference on A.I. (AAAI-91)*, pages 260–267, Anaheim (CA, USA).

Nebel, B. and Bürckert, H. (1995). Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66.

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley, Reading (Mass, USA).

Rutten, E. and Hertzberg, J. (1993). Temporal planner = nonlinear planner + Time Map Manager. *AI Communications*, 6(1):18–26.

Schrag, R., Boddy, M., and Carciofini, J. (1992). Managing disjunction for practical temporal reasoning. In Nebel, B., Rich, C., and Swartout, W., editors, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, pages 36–46. Morgan Kaufmann.

Schwalb, E. and Dechter, R. (1997). Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, 93(1-2):29–61.

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London (UK).

van Beek, P. (1991). Temporal query processing with indefinite information. *Artificial Intelligence in Medicine*, 3:325–339.

van Hentenryck, P. (1995). Constraint solving for combinatorial problems: a tutorial. In *1st International Conference on Principles and Practice of Constraint Programming (CP-95)*, pages 564–587, Cassis (France).

Vidal, T. and Ghallab, M. (1996). Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 48–52, Budapest (Hungary).

Vilain, M., Kautz, H., and van Beek, P. (1989). Constraint propagation algorithms: a revised report. In *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufman.