Rapporto di ricerca
Research report
December 2023

1/2023

Some results and challenges Extending Dynamic Controllability to Agile Controllability in Simple Temporal Networks with Uncertainties.

Roberto Posenato
Marco Franceschetti
Carlo Combi
Johann Eder

Abstract

Simple Temporal Networks with Uncertainty (STNU) are an expressive means to represent temporal constraints, requirements, or obligations. They feature contingent timepoints, which are set by the environment with a specified interval. Dynamic controllability is the current most relaxed notion for checking that the constraints are not in conflict. It requires that a timepoint may only depend on earlier timepoints. *Agile controllability* extends dynamic controllability by taking into account that a later timepoint might already be known earlier and allowing a timepoint to depend on all timepoints whose value is known before. In this report, we formally introduce the notion of an STNU with oracle timepoints, formally define the notion of agile controllability, and discuss approaches for checking agile controllability.

Keywords: simple temporal networks with uncertainty, dynamic controllability, agile controllability

1 Introduction

Time-constrained process models are subject to requirements stating constraints on the durations and times of execution of their activities [7]. Constraints on the duration may state minimum or maximum allowed activity durations and whether an agent may control an activity duration or only observe it to take a given time within a specified interval [13]. Constraints on the time of execution restrict the occurrence of timepoints–start and end times of activities, temporal parameters [6]– with respect to other timepoints.

For time-constrained process models, it is desirable to check whether they are temporally correct, i.e., they admit executions that do not violate any temporal requirements [1]. For processes that are entirely under the control of an agent, the satisfiability of temporal constraints is an adequate notion for temporal correctness [4]. However, for processes in which some activity durations are not controllable by the agent (known as *contingent* durations), more sophisticated notions for temporal correctness, such as dynamic controllability, are required [11, 14]. They consider that a contingent duration is only known when it completes; for instance, the duration of an uncontrollable chemical reaction is only observed when the reaction ends. The dynamic controllability of a process model featuring contingent durations can be effectively checked via mapping of the process model into a Simple Temporal Network with Uncertainty (STNU), a data structure that encodes the timepoints and temporal requirements in terms of nodes and edges [3, 9].

Contingent durations effectively model the characteristics of several real-world process activities. However, the assumption that contingent durations may only be revealed at the time of completion of the corresponding activities is too restrictive for specific scenarios. There exist situations in which an activity duration cannot be controlled, but this duration is revealed *before* the activity completion. In such cases, knowing in advance the contingent duration allows for more flexible process scheduling since the agent does not need to consider the worst-case scenario (e.g., the maximum possible duration) anymore to schedule the execution of other process activities, avoiding constraint violations. An example of such a case is the shipment of an order: typically, the shipping times are given within an uncontrollable range at the time of order placement, say 6 to 10 days. However, after the order has been placed and processed, the information that the shipment will take precisely seven days may be revealed. With this information, a buyer may be able to schedule any further activities that depend on the shipment with the certainty of the delivery time.

Currently, state-of-the-art procedures for checking the dynamic controllability of processes based on the STNU ignore the possibility of knowing in advance contingent durations. Hence, they produce potentially overly restrictive results for process models where contingent durations may be revealed in advance. More specifically, these procedures label these models as not dynamically controllable while they can be scheduled, avoiding constraint violations thanks to such information.

Here, we propose a conservative extension of the STNU to overcome the limitation of not being able to represent information on contingent durations in advance. We introduce the concept of *contingent oracle* node to represent the point in time in which a contingent duration is revealed before its completion. Then, we discuss possible algorithmic solutions to check the dynamic controllability of an STNU with oracle nodes, along with respective limitations.

In Section 2, we recall the formalization of the STNU. We introduce a motivating example in Section 3. In Section 4, we introduce the extension of the STNU with contingent oracles and propose algorithms for checking its dynamic controllability. In Section 5, we discuss the proposed algorithms and their respective limitations. Section 6 concludes this paper.

2 Backgrounds: Simple Temporal Networks with Uncertainty

The Simple Temporal Network with Uncertainty (STNU) is a data structure that models temporal problems in which the execution of some events cannot be controlled. The STNU is composed of a set of timepoints and a set of temporal constraints. The timepoint set is partitioned into controllable (executable) timepoints and uncontrollable (contingent) ones; the constraint set is partitioned into regular and contingent ones.

The following is a formal definition of the STNU adapted from [8]:

Definition 1 (STNU)**.** An STNU is a triple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$, where:

- $\mathcal{T}$ is a finite, non-empty set of real-valued variables called timepoints. $\mathcal{T}$ is partitioned into $\mathcal{T}_X$, the set of executable timepoints, and $\mathcal{T}_C$, the set of contingent timepoints.

- $\mathcal{C}$ is a set of binary (ordinary) constraints, each of the form $Y - X \leq \delta$ for some $X, Y \in \mathcal{T}$ and $\delta \in \mathbf{R}$.

- $\mathcal{L}$ is a set of contingent links, each of the form (A, x, y, C), where $A \in \mathcal{T}_X, C \in \mathcal{T}_C$ and $0 < x < y < \infty$. A is called the *activation* timepoint; C *contingent* timepoint. If (A_1, x_1, y_1, C_1) and (A_2, x_2, y_2, C_2) are distinct contingent links, then $C_1 \neq C_2$.

The tuple $(\mathcal{T}, \mathcal{C})$ forms a Simple Temporal Network (STN), a data structure proposed by Dechter et al. in [4] to study the Simple Temporal Problem, i.e., the satisfiability of a set of (controllable) temporal constraints. An STN is *satisfiable* if it is possible to determine an assignment (schedule) to timepoints such that all the constraints are satisfied. We say that a controller *executes an STN* when it schedules its timepoints.

The STNU model extends the STN one by adding contingent timepoints and links. The contingent link bounds cannot be modified, and the schedule of contingent timepoints is decided by *nature*, who determines the *duration* of each contingent link once the relative activation timepoint is scheduled.

An important property of the STNU is the *dynamic controllability.* To define it, we need to define some concepts we recall from [8] formally.

Definition 2 (Situation)**.** If $(A_1, x_1, y_1, C_1), \dots (A_K, x_K, y_K, C_K)$ are the K contingent links in an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, then the corresponding space of *situations* for $\mathcal{S}$ is $\Omega = [x_1, y_1] \times \cdots [x_K, y_K]$. Each *situation* $\omega = (\omega_1, \dots, \omega_K) \in \Omega$ represents one possible complete set of values for the duration of the contingent links of $\mathcal{S}$ (chosen by *nature*).

Definition 3 (Schedule)**.** A *schedule* for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is a mapping $\xi : \mathcal{T} \to \mathbf{R}$. Ξ denotes the set of all schedules for an STNU. For historical reasons, we represent $\xi(X)$ as $[X]_\xi$.

Definition 4 (Execution Strategy)**.** An *execution strategy* for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is a mapping $S : \Omega \to \Xi$.

Definition 5 (Viable Execution Strategy)**.** An execution strategy for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is *viable* if for each situation $\omega \in \Omega$ the schedule $S(\omega)$ is a solution for $\mathcal{S}$, i.e., an assignment that satisfies all the constraints in the network.

Definition 6 (Dynamic Execution Strategy)**.** An execution strategy for an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is *dynamic* if, for any two situations ω', ω'' and any executable timepoint $X \in \mathcal{T}$, it holds that:

$$\textit{if } [X]_{S(\omega')} = k \textit{ and } S(\omega')^{\leq k} = S(\omega'')^{\leq k}\textit{, then } [X]_{S(\omega'')} = k,$$

where $S(\omega')^{\leq k}$ is the set of contingent link durations observed up to and including time k, called *history until* k. Since history also considers contingent durations observed at instant k, we say that the dynamic execution strategy implements the *instantaneous reaction* semantics.

An STNU is *dynamically controllable* if there exists a viable dynamic execution strategy for it, i.e., an execution strategy that assigns the executable timepoints with the guarantee that all constraints will be satisfied, irrespectively from the values (within the specified bounds) the contingent timepoints will be revealed to take [8].

Each STN $\mathcal{S} = (\mathcal{T}, \mathcal{C})$ has a corresponding graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, where the timepoints in $\mathcal{T}$ serve as the graph's nodes and the constraints in $\mathcal{C}$ correspond to labeled, directed edges in $\mathcal{E}$. In particular: $\mathcal{E} = \{X \xrightarrow{\delta} Y \mid (Y - X \leq \delta) \in \mathcal{C}\}$. For convenience, $X \xrightarrow{\delta} Y$ may be notated as (X, δ, Y).

Constraint propagation algorithms based on applying constraint propagation rules on the corresponding graph have been proposed to check whether an STNU is dynamically controllable. A constraint propagation algorithm applies constraint propagation rules to derive implicit constraints from the existing ones in the STNU. The algorithm terminates when either reaching network quiescence, i.e., no new constraints can be derived (the network is dynamically controllable), or a negative loop is found (the network is not dynamically controllable). The original rules proposed by Morris and Muscettola in [11] to check the dynamic controllability (DC) property of an STNU are depicted in Table 1. Such rules assume that an STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ has a corresponding graph $\mathcal{G} = (\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}_{\text{lc}} \cup \mathcal{E}_{\text{uc}})$, where $(\mathcal{T}, \mathcal{E}_{\text{o}})$ is the graph for the STN $(\mathcal{T}, \mathcal{C})$, and $\mathcal{E}_{\text{lc}}$ and $\mathcal{E}_{\text{uc}}$ contain labeled, directed edges derived from the contingent links in $\mathcal{L}$. In particular: $\mathcal{E}_{\text{lc}} = \{A \xrightarrow{c:x} C \mid (A, x, y, C) \in \mathcal{L}\}$, and $\mathcal{E}_{\text{uc}} = \{C \xrightarrow{C:-y} A \mid (A, x, y, C) \in \mathcal{L}\}$. The so-called *lower-case* (LC) edge $A \xrightarrow{c:x} C$ represents the uncontrollable possibility that the duration $C - A$ might take on its minimum value x, while the so-called *upper-case* (UC) edge $C \xrightarrow{C:-y} A$ represents the uncontrollable possibility that $C - A$ might take on its maximum value y. Such edges may be respectively notated as $(A, c{:}x, C)$ and $(C, C{:}{-y}, A)$, while constraints in $\mathcal{C}$ and edges in $\mathcal{E}_{\text{o}}$ may be called *ordinary* constraints and edges, respectively, to distinguish them from the LC and UC edges.

Rule	**Conditions**	**Pre-existing and generated edges**
No Case (NC):		$\begin{tikzcd} W \arrow[r, "v"] \arrow[rr, bend right, "u+v"'] & Y \arrow[r, "u"] & X \end{tikzcd}$
Upper Case (UC):		$\begin{tikzcd} X \arrow[r, "u"] \arrow[rr, bend right, "C{:}u+v"'] & Y \arrow[r, "C{:}v"] & A \end{tikzcd}$
Lower Case (LC):	$v < 0$. Also, $v \leq 0$ is fine but not necessary.	$\begin{tikzcd} X & C \arrow[l, "v"'] & A \arrow[l, "c{:}u"'] \arrow[ll, bend left, "u+v"] \end{tikzcd}$
Cross Case (CC):	$D \not\equiv C$ and $v < 0$	$\begin{tikzcd} X & C \arrow[l, "D{:}v"'] & A \arrow[l, "c{:}u"'] \arrow[ll, bend left, "D{:}u+v"] \end{tikzcd}$
Label Removal (LR):	$v \geq -x$	$\begin{tikzcd} C & A \arrow[l, "c{:}x"'] & X \arrow[l, "C{:}v"', "v"] \end{tikzcd}$

Table 1: Morris-Muscettola rules for DC-checking STNUs

These rules were designed assuming the following:

- instantaneous reaction semantics, meaning that *"when the nature decides the duration of a contingent link, the agent can react at the same time executing one or more other timepoints"*.
- early-execution strategy, meaning that timepoints are assigned with the smallest possible value. The rules do not determine an upper bound for executing the timepoints.

The Morris-Muscettola DC-checking algorithm applies the rules from Table 1 in at most n^2 rounds, with a cost of $O(n^3)$ each round, where n is the number of network nodes. Afterward, it computes the *AllMax* STN, which is the STN projection in which each contingent link is set to its maximum duration. The *AllMax* STN is computed from the *fully propagated* STNU by:

1. removing all lower-case edges; and
2. removing the upper-case letters from all uppercase edges (original or generated).

If the *AllMax* STN is consistent, the STNU is declared dynamically controllable.

3 Motivating Example

Figure 1 represents a motivating scenario, which we will briefly discuss here.

The motivating example refers to managing patients' activities before surgical intervention. It is becoming widely acknowledged that patients reaching some planned intervention in an (as much as possible) good health state have a better recovery [5].

The process in the figure starts with the usual information to patients, who must be aware of the following activities. Then, different threads of activities are initiated. Two threads are related to the Physical and Psychological parts, respectively. The proposed physical exercises depend on and can be refined with respect to the results of different questionnaires the patient has to answer. After the end of these activities, the intervention is performed. Focusing on the process fragment related to the physical exercise activity and the related monitoring, these activities are connected through some temporal constraints, representing the allowed delays between their start and endpoints, respectively. Moreover, activities are enriched with their allowed time duration. In the figure, only the temporal duration and the temporal constraints relevant to

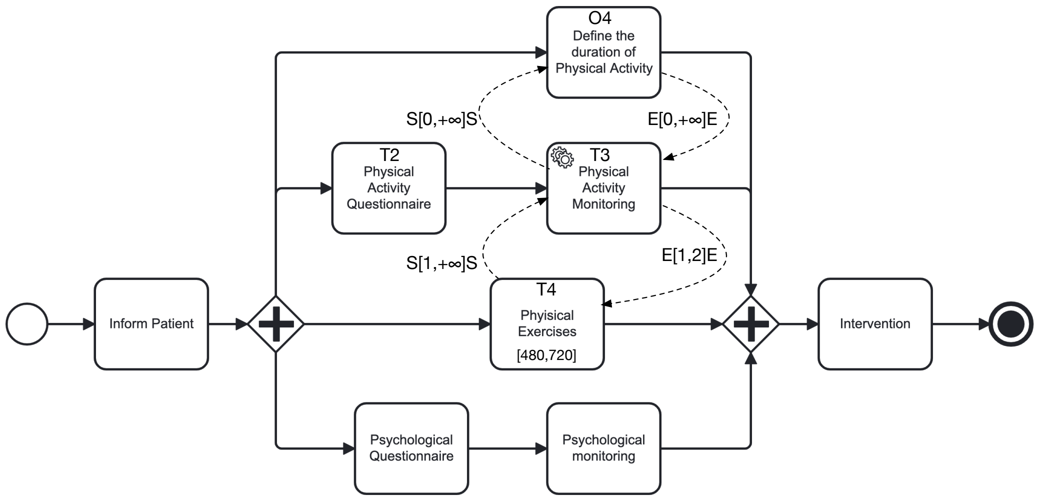

O4
Define the duration of Physical Activity
S[0,+∞]S
E[0,+∞]E
T2
Physical Activity Questionnaire
T3
Physical Activity Monitoring
S[1,+∞]S
E[1,2]E
T4
Physical Exercises
[480,720]
Inform Patient
Intervention
Psychological Questionnaire
Psychological monitoring

Figure 1: A BPMN representation of the process for the patient pre-intervention period.

our discussion are reported. The notation for task durations and for inter-task constraints is relatively standard in the literature [2]. Tasks have a duration attribute represented as a range [x,y], with $0 < x \leq y < \infty$, where x/y is the minimum/maximum allowed time span for an activity to go from state "started" to "completed" [10]. Inter-task constraints limit the time distance between the starting/ending instants of two tasks and have the form $I_S[u, v]I_F$, where I_S is the starting (S)/ending (E) instant of the first task, while I_F is the starting/ending instant of the second one [2].

Task T4 - Physical Exercises (PE) has to be performed for a period from 20 to 30 days, i.e., from 480 to 720 hours. Task T3 - Physical Activity Monitoring has to start (at least) 1 hour after the start of physical exercises and end 1 to 2 hours before the exercises end to avoid noisy information during the initial and final phases of physical activities. As the T3 - Physical Activity Monitoring (PAM) is a task, it is evident that the constraints between PAM and PE cannot be satisfied. Indeed, as the contingent duration of PE is not determined by the system, it can only be observed when the task ends, and thus, PAM cannot be set to end according to the required time distance before PE. It is worth noting that such observation holds both if we consider PAM a contingent task, having a duration only observable by the system, and if we consider it a 'controllable' task, for which the system can set the duration. Indeed, in any case, the ending instant of PAM, even if controllable, should be set with respect to the ending instant of PE, which has to occur afterward. Thus, the PAM ending instant cannot be adequately set unless there is some kind of early specification/acquisition/knowledge about the (contingent) duration of PE.

Let us now explicitly specify that the duration of Physical Exercises is set by the activity O4 - ‘Define the duration of Physical Activity’ (DDPA). Let us also assume, for the sake of simplicity, that the system can decide when the PAM has to end. From this perspective, how to derive when it is required to execute DDPA to make the overall process model controllable? It is straightforward to verify that if the system knows the overall duration of PE at least two hours before the effective end of PE, then the end of PAM can be set, satisfying the given constraints.

To the best of our knowledge, the early specification of contingent task durations cannot be modeled in the current temporal business process models and/or in the related temporal constraint networks. Representing and reasoning on such temporal aspects in the context of STNUs, on which temporal business process models are mapped, will be the focus of this paper. More specifically, we will consider the following issues:

- how to extend STNUs to represent contingent temporal constraints having their duration acquired/decided/known possibly before their occurrences?

- How to derive when such extended STNUs are controllable, i.e., how early do we need to acquire/decide/know the duration of some contingent link?

So far, all DC-checking algorithms and the respective constraint propagation rules assume that the value of a contingent duration is given only at the time of occurrence of the associated contingent timepoint. In general, execution strategies for dynamic controllability require that a timepoint may be assigned a value based on the knowledge of the values of all prior occurred timepoints and prior revealed contingent durations, but not future ones.

However, in real-world applications, the duration of a contingent link may be known before the contingent timepoint occurs because, for example, of communication with the environment, data received, or the result of an activity execution. In these cases, it is helpful to distinguish between the value of a contingent duration d associated with a contingent timepoint C and the point in time in which this duration value is revealed, $i(C)$. This allows for more flexible execution strategies for dynamic controllability. In particular, it is sufficient to require that the execution of a timepoint X may depend on C if d is known at the latest at the execution of X, i.e. if $i(C) \leq X$.

So far, the STNU does not allow decoupling the value of a contingent duration and the time of occurrence of the associated timepoint. Thus, dynamic controllability is defined, assuming that the two coincide. To take into account the possibility that the duration of a contingent link is revealed before the contingent timepoint occurs, the STNU and the above definitions must be extended. In the following, we introduce a conservative extension of the STNU that admits decoupling the time in which a contingent timepoint occurs from the time in which the duration of its contingent link is revealed.

4 STNU with Contingent Oracles

We introduce a new kind of timepoint called *contingent oracle*. A contingent oracle timepoint O_C is a timepoint associated bi-univocally with a contingent link (A, C). When O_C is executed, it reveals the duration of the associated contingent link. In other words, O_C can reveal the duration of the contingent link before the occurrence of the contingent timepoint C.

Such an oracle timepoint is important because it allows the management of networks containing timepoints that must be executed within a restricted temporal distance from a contingent timepoint. Such a kind of network is not DC without oracles.

For example, let us assume there is a network where a node B must be executed exactly four units of time *before* the contingent node C. In a classical STNU, such a network is not dynamically controllable because it is impossible to know in advance when C occurs. If there exists the C oracle O_C that occurs at least four units of time *before* C, then it is possible to determine when C occurs and, consequently when B must occur.

Now, let us consider the more structured case of Figure 1. Using some transformation rules like the ones presented in [12], it is possible to represent the process as an STNU instance. Figure 2 represents an excerpt of the STNU, focused only on three activities of the process: "Physical activity Questionnaire", "Physical Activity Monitoring", "Define duration of Physical Activity", and "Physical Exercise". Each activity is represented by two timepoints, one representing the starting instant, the other the ending one, and one or two constraints on its duration. According to the kind of duration (controllable or not), the duration constraints can be ordinary or contingent. Let us consider each activity.
The activity "Physical Activity Questionnaire" is a controllable activity. The execution agent can fix its duration. Therefore, it is represented by the two timepoints A_2 and B_2 and by two ordinary constraints between A_2 and B_2 that impose that the duration must be in the range $[1, 2]$. (Without loss of generality, we assume that the time granularity is *hours* and that all constraint values are integers.)
The activity "Physical Exercise" is contingent since it is possible to fix when

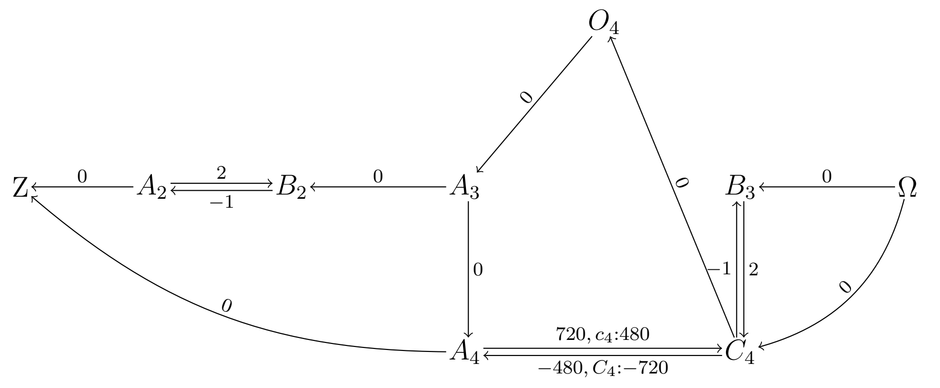
it must start, but given a possible duration, the end of the activity can only be observed when it occurs. Therefore, it is represented as two timepoints A_4, and C_4, and by two constraints representing the upper-case and lower-case of the contingent link associated to the activity. The duration of such an activity must be in the range of 20 to 30 days, i.e., $[480, 720]$ using the time unit of hours.
The activity "Physical Activity Monitoring" is just an activity that depends on the "Physical Exercise" one. Therefore, its timepoints are A_3 and B_3, which are constrained to be just after A_4 and one day C_4 at maximum, respectively.
Last, the activity "Define the duration of Physical Activity" is represented just by one timepoint, O_4. We assume that such activity is instantaneous since it represents just the instant in which the duration of "Physical Exercise" is decided. In the original BPMN process, such activity is constrained to be after the start of the monitoring activity, which must start after the "Physical Exercise" one. We will see below how it is possible to exploit the information associated with this activity to manage the execution of the process.

The STNU excerpt in Figure 2 is not dynamically controllable. Indeed, it is not possible to find a schedule that assigns the right execution time to B_3, which must occur one day max before C_4, without knowing in advance the instant in which C_4 is executed. On the other hand, if O_4 is considered the oracle for the activity (A_4, C_4), it is possible to constraint when O_4 must occur before C_4 to guarantee that B_3 can be scheduled correctly. In particular, in the example, thanks to the instantaneous reaction assumption, it is sufficient that O_4 would be executed just before C_4 (even on the same day, but before) to schedule B_3 correctly.

In the following, we extend the definitions of oracle dependency, execution strategy, viable execution strategy, and dynamic execution strategy when oracle timepoints are present.

Definition 7 (Oracle Dependency)**.** Let $\mathcal{S}_\mathcal{O} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU with contingent oracles; let a timepoint X constrained to be before a contingent timepoint C within a strict range; more formally, let $X \in \mathcal{T}_X, C \in \mathcal{T}_C$ with $(A, x, y, C) \in \mathcal{L}$. Let $X - C \leq -a,\ C - X \leq b$, with $0 \leq a < \infty$ or $a \leq b < \infty$, be constraints in $\mathcal{C}$ such that $b - a < y - x$. Then, we say that X depends on the oracle O_C of C.

The oracle dependency definition expresses the requirement that if a timepoint X is constrained to be executed with respect to a contingent timepoint C within a time interval that is smaller than the contingent interval, then the duration of the contingent link should be revealed in advance (i.e., by the oracle) in order to schedule X to satisfy the constraints.



A_2, B_2 represent the starting/ending event of 'Physical Activity Questionnaire'
A_3, B_3 represent the starting/ending event of 'Physical Activity Monitoring'
A_4, C_4 represent the starting/ending event of 'Physical Exercises'
O_4 represents the end of 'Define the duration of Physical Activity'

Figure 2: STNU distance graph representing a possible conversion of a part of the BPMN model in Figure 1

Definition 8 (Execution Strategy with Contingent Oracles)**.** Let $\mathcal{S}_{\mathcal{O}} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU with contingent oracles. An execution strategy with contingent oracles for $\mathcal{S}_{\mathcal{O}}$ is a mapping $S_O : \Omega \to \Xi$ where for each timepoint X that depends on oracle O_C for some contingent timepoint C, O_C is executed no later than X, i.e., $[O_C]_{S_O(\omega)} \leq [X]_{S_O(\omega)}$.

Definition 9 (Viable Execution Strategy with Contingent Oracles)**.** Let $\mathcal{S}_{\mathcal{O}} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU with contingent oracles. An execution strategy with contingent oracles S_O for $\mathcal{S}_{\mathcal{O}}$ is *viable* if for each situation $\omega \in \Omega$ the schedule $S_O(\omega)$ is a solution for $\mathcal{S}_{\mathcal{O}}$.

Definition 10 (Dynamic Execution Strategy with Contingent Oracles)**.** Let $\mathcal{S}_{\mathcal{O}} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ be an STNU with contingent oracles. An execution strategy with contingent oracles S_O for $\mathcal{S}_{\mathcal{O}}$ is *dynamic* if, for any two situations ω', ω'' and any executable timepoint $X \in \mathcal{T}$, it holds that:

if $[X]_{S_O(\omega')} = k$ *and* $S_O(\omega')^{\leq k} = S_O(\omega'')^{\leq k}$, *then* $[X]_{S_O(\omega'')} = k$,

where $S_O(\omega')^{\leq k}$ is the set of contingent link durations observed up to and including time k and contingent link durations revealed up to and including time k (i.e., with associated oracle executed up to and including time k), called *observed and revealed history until* k. Since history also considers contingent durations observed and revealed at instant k, we say that the dynamic execution strategy implements the *instantaneous reaction* semantics.

Definition 11 (Agile Controllability)**.** An STNU with contingent oracles is agile controllable if it admits a viable dynamic execution strategy with contingent oracles. We refer to agile controllability (AC) as the property of being agile controllable.

In the following, we discuss how to check the agile controllability of an STNU with contingent oracles.

4.1 Checking Agile Controllability

To consider the new kind of oracle timepoint, it is necessary to update the Morris-Muscettola rules to consider the oracle role.

The affected rules are the LC and UC rules. In general, the idea is that when a non-contingent node X has to be "strictly" scheduled with respect to the occurrence of a contingent C, then the presence of the oracle O_C is necessary, and such an oracle node must be scheduled before C for allowing the proper scheduling of X.

In detail, a node X has to be "strictly" scheduled with respect to a contingent C (and not w.r.t. A) when:

- X must occur before C (there is a negative edge between C and X),
- the distance range between X and C is smaller than the contingent allowed duration. In other words, if there is a configuration like

$$\begin{tikzcd} X \arrow[r, shift left, "v"] & C \arrow[l, shift left, "-u"] \arrow[r, shift left, "C:-y"] \arrow[d, "0"] & A \arrow[l, shift left, "c:x"] \\ & O_C & \end{tikzcd}$$

  where $v - u < y - x$.

In such a case, the LC and UC rules must not be applied because they would determine a false negative loop. On the contrary, the information given by the oracle O_C must be considered to check whether X can be scheduled correctly.

A possible rules update for implementing the above strategy is given in Table 2.

The nLC rule replaces LC. It must be applied only when the contingent node C does not have its oracle node O_C, or the span $v - u$ is greater than the span $y - x$ because, in this case, X can be scheduled w.r.t. A.

nUC and UC* rules replace UC. In the case of the first propagation of upper-case labeled value (first pattern), rules nUC must be considered, and it must be applied only when the contingent node C does not have its oracle node O_C, or the span $v - u$ is greater than the span $y - x$. For all other cases, UC* must be considered to propagate the wait (Y, A).

Rule ORC is new. When $v - u < y - x$ and there is the oracle node O_C, then it is necessary to require that X must occur after O_C for verifying whether it is possible to schedule it correctly w.r.t. C. The constraints

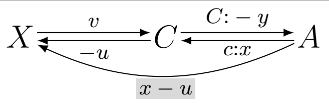
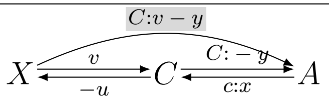
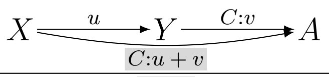
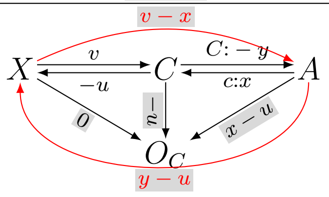
Rule	**Conditions**	**Pre-existing and generated edges**
New Lower Case (nLC):	$-u \leq 0$ and (O_C does not exists or $v-u \geq y-x$)	
New Upper Case (nUC):	O_C does not exist or $v-u \geq y-x$	
UC*	$Y \not\equiv C$	
Oracle (ORC):	X is **not** a contingent node, $v-u<y-x$, and O_C exists.	

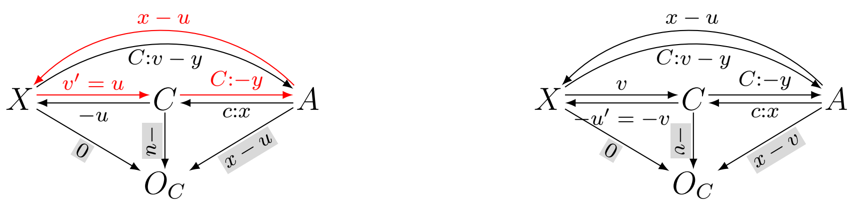
Table 2: A possible rule update. nLC replaces LC. nUC and UC* replace UC. $C\xrightarrow{-u}O_C$ and $A\xrightarrow{x-u}O_C$ are after the addition of $X\xrightarrow{0}O_C$ and the NC rule could determine them; we propose to add such values directly to understand the rule better. The constrains $X\xrightarrow{v-x}A$ and $A\xrightarrow{y-u}X$ are necessary to bound timepoints X and A as explained in the next section.

4.2 Some Issues with the New Rules

The new rules nLC, nUC, and ORC contain a new aspect for AC checking that is not present in the propagation rules for DC checking of STNUs proposed so far. Rules nLC and nUC are alternatives to rule ORC, and the constraints determined by nLC and nUC are not compatible with the ones determined by ORC.

In case there is an oracle timepoint for a contingent link, nLC and nUC can be applied if the interval span between the external timepoint X and the contingent one C, i.e., $v-u$, is greater than the span of the contingent link, i.e., $y-x$. The constraint values v and u can be modified during an AC checking by the propagation of external values. The bounds of a contingent link, x and y, cannot be modified. Therefore, the span $v-u$ might become smaller than $y-x$. If such an event occurs, only the LC rule should be applied, and the previous constraints between A and X should not be present, but they are already propagated.

For example, let us consider the sample STNU graph pattern in Figure 3. Let us assume that, initially, $X\xrightarrow{v}C$ and $C\xrightarrow{-u}X$ are such that $v-u \geq y-x$. Therefore, the rules nLC and nUC are applied, and they determine the constraints $A\xrightarrow{x-u}X$ and $X\xrightarrow{C:v-y}A$, respectively.



(a) Sample STNU graph showing that values determined by nLC and ORC are incompatible

(b) Sample STNU graph showing that values determined by nUC and ORC are incompatible

Figure 3: Two sample cases to show that values determined by nLC/nUC and ORC are incompatible.

There are two cases.

1. v becomes smaller

Let us assume that, during the checking, v becomes smaller, e.g., it becomes v' equal to u, i.e., $v' = u$ (see Figure 3a). Such a value is still a value for which the network is still dynamically controllable if the oracle O_C is considered. Indeed, because $v' - u = 0$ and $0 \not\geq y - x$ by definition of contingent link, ORC is applied (for now, we add only the gray values of ORC rule), and the new constraints (with the oracle behavior) guarantee to schedule X correctly at runtime.

On the other hand, the already present $A \xrightarrow{x-u} X$ determines that the cycle (A, X, C, A) becomes negative because $x-u+u-y = x-y < 0$ in the AllMax projection.[1] Such a negative cycle shows that if the contingent link assumes its maximum duration, the constraint $A \xrightarrow{x-u} X$ is too strict. Even assuming that the oracle replaces the contingent link with two ordinary constraints representing the exact duration, the propagation of such constraints cannot change the value of $A \xrightarrow{x-u} X$ when the duration is maximum. So, in any case, $A \xrightarrow{x-u} X$ must be removed from the network. It is also necessary to remove all possible derived constraints from $A \xrightarrow{x-u} X$ so far in the network.

As concerns the last added wait $X \xrightarrow{C:v-y} A$, it is useless, and it will be replaced by a stricter constraint every time that the oracle reveals the exact duration of the contingent link.

In summary, to manage this case, it is necessary to remove $A \xrightarrow{x-u} X$ and all its derived constraints, apply the ORC rule with its red constraint

[1]The check of a negative cycle in AllMax projection is the final phase of the Morris' DC checking algorithm. The AllMax projection is an STN derived from the network where all upper-case constraints are transformed into ordinary constraints, and lower-case constraints are not considered. The goal of such a phase is to verify that the network is controllable when all contingent links assume their maximum value. The constraints implied by the assumption that all contingent links assume their minimum values are already present in the network as ordinary constraints.

$A \xrightarrow{y-u} X$.

2. $-u$ becomes smaller (more negative)
Let us assume that, during the checking, $-u$ becomes smaller, e.g., $-u' = -v$. Such a value is still one for which the network is still dynamically controllable if the oracle O_C is considered (see Figure 3b).

Again, since $0 < y - x$, nLC is not applied (this is a critical fact!), while ORC is applied (for now, we add only gray values of ORC), and the new constraints (with the oracle behavior) guarantee to schedule X correctly at runtime.

In this case, no negative cycle (X, A, C, X) arises because in the AllMax graph, the constraint $A \xrightarrow{c:x} C$ is not considered, and, therefore, cannot generate a negative cycle.[2]

In this case, it is the wait $X \xrightarrow{C:v-y} A$ that could become too strict. Indeed, when $v > y$, the wait becomes positive and, by LR rule, an ordinary constraint $X \xrightarrow{v-y} A$. Now, suppose that, at runtime, the oracle reveals that $C - A = x$. The constraints between X and A should be updated as $X \xrightarrow{v-x} A$ and $A \xrightarrow{-v+x} X$. The new constraint $X \xrightarrow{v-x} A$ is weaker than the already present $X \xrightarrow{v-y} A$, so it is not added. Now, $X \xrightarrow{v-y} A$ and $A \xrightarrow{-v+x} X$ determines a negative cycle at runtime!

Therefore, it is necessary to remove $X \xrightarrow{C:v-y} A$ and all its derived constraints; then, apply ORC including its red constraint $X \xrightarrow{v-x} A$.

Note that since $X \xrightarrow{C:v-y} A$ could have a positive value and, therefore, becomes an ordinary constraint, it is not possible to rely on the upper-case letter to identify it and all its derived constraints.

A first new AC-checking algorithm that manages contingent links with oracles is the following ($\Delta_X = v - u$ and $\Delta_C = y - x$).

[2]Note: if nLC had applied with the new value for $-u$, then $A \xrightarrow{x-u} X$ would have become $A \xrightarrow{x-v} X$, and in the AllMax graph, there would have been a negative cycle.

$$\begin{tikzcd}
Y \arrow[r, shift left, "v'"] \arrow[rrr, bend left=50, "\{C{:}v'+v-y\}_{X_C}"'] & X \arrow[l, shift left, "-u'"] \arrow[r, shift left, "v"] \arrow[rr, bend left=50, "\{C{:}v-y\}_{X_C}"'] & C \arrow[l, shift left, "-u"] \arrow[r, shift left, "C{:}-y"] & A \arrow[l, shift left, "c{:}x"] \arrow[lll, bend right=60, "\{x-u-u'\}_{X_C}"'] \arrow[ll, bend right=60, "\{x-u\}_{X_C}"'] \\
 & & O_C &
\end{tikzcd}$$

Figure 4: Sample STNU having labeled values generated by nLC and nUC.

Algorithm 1: `AC-CHECKING`

```
Input: STNU $G = (V, E)$. A contingent node $C$ can have an oracle $O_C$.
Output: YES if the STNU is agile controllable, NO otherwise.
AC := $\top$;
do
    Apply Morris' rules: NC, CC, and LR;
    Apply rules: nLC and nUC;
    foreach oracle $O_C$, $C \xrightarrow{-u} X$, $X \xrightarrow{v} C$ do
        if $\Delta_X < \Delta_C$ and ORC has not been applied then
            Remove possible previously generated wait between $X$ and $A$
              and all its derived edges;
            Remove possible previously nLC-generated constraint $X - A$
              and all its derived edges;
            Apply rule ORC;
    AC := Check consistency of corresponding AllMax graph;
while AC and some new edges have been added;
return AC;
```

4.3 An Alternative Approach to Avoid Backtracking

Backtracking can slow down the actual computation of checking, and it is not simple to manage cases where it is necessary to manage backtracking for more contingent links.

Accepting the cost of more memory, it seems more straightforward to properly label some derived values (like the ones derived by nLC or nUC when the involved contingent node has an oracle) and use them only while they are valid.

In more detail, let us denote by $\{v\}_{X_C}$ a value derived by a value determined using the nLC or nUC rule involving a node X, a contingent one C when X must occur before C and C has an oracle O_C.

Figure 4 shows an example of an STNU where values determined by nLC and nUC are labeled.

$$\begin{tikzcd}
Y \arrow[r, shift left, "v'"] \arrow[drrr, bend right, "\{v'\}_{\overline{X_C}}"'] & X \arrow[l, shift left, "-u'"] \arrow[r, shift left, "v"] \arrow[drr, "\{0\}_{\overline{X_C}}"'] & & C \arrow[ll, shift left, "-u"] \arrow[r, shift left, "C{:}-y"] \arrow[d, "\{-v\}_{\overline{X_C}}"'] & A \arrow[l, shift left, "c{:}x"] \arrow[dl, "\{x-v\}_{\overline{X_C}}"] \\
& & & O_C &
\end{tikzcd}$$

Figure 5: Sample STNU having labeled values generated by ORC.

While a proper wait can only be backpropagated, the value determined by the nLC can be forward propagated. Therefore, it is also possible to have constraints that require combining different labeled values $X \xrightarrow{\{v\}_{X_C}} A \xrightarrow{\{u\}_{Y_D}} F$. In this case, we combine the values remembering the labels $X \xrightarrow{\{v+u\}_{X_C Y_D}} F$.

When ORC is involved, let us denote by $\{v\}_{\overline{X_C}}$ a value derived by the rule ORC considering a node X, a contingent one C when X must occur before C and C has an oracle O_C.

Figure 5 shows an example of an STNU where values determined by ORC are labeled.

Labeled values can be considered for propagation only when their labels do not contain opposite components. For example, the two constraint values $X \xrightarrow{\{v\}_{X_C}} A \xrightarrow{\{u\}_{\overline{X_C}}} F$ cannot be combined by the NC rule. Indeed, $X \xrightarrow{\{v\}_{X_C}} A$ was derived by an initial nUC rule on contingent link (A, C), while $A \xrightarrow{\{u\}_{\overline{X_C}}} F$ was derived by an initial ORC rule on the same contingent link (A, C); the two rule values are mutually exclusive and, therefore, cannot be combined.

Now, let us consider a case in which a negative cycle is detected. A negative cycle is detected when there is a self-loop on a node having a negative value v. The label ℓ associated with v can have one of the following configurations:

1. ℓ label. It occurs when the negative cycle does not depend on any contingent. The network is not AC. AC check must stop.

2. ℓ has only one component, e.g., $\{v\}_{X_C}$. The negative cycle depends on the values determined by the nLC/nUC rule involving X and C. All values having label X_C must be ignored from now on. If also all values labeled by $\overline{X_C}$ must be ignored (because of a previous negative cycle involving them), then it is not possible to have a schedule for X satisfying all constraints. The network is not AC. AC check must stop.

3. ℓ has two or more components. e.g., $\{v\}_{X_C \overline{Y_D}}$. To discuss this case, let us consider $\ell = \{v\}_{X_C \overline{Y_D}}$. In this case, values having labels containing $X_C \overline{Y_D}$ cannot be more propagated. On the contrary, all values having a label containing any other combinations of X_C and Y_D (e.g., $X_C Y_D, \overline{X_C} Y_D, \overline{X_C Y_D}$) can be propagated.

If all other combinations of components X_C, Y_D have been blocked in the past, then it is not possible to find a configuration for timepoints X, C, D that is agile controllable. Hence, the network is not AC.

5 Discussion

A constraint propagation algorithm based on the new rules constitutes a possible approach to the AC-checking of an STNU with contingent oracles. However, as detailed in Section 4.2, the constraint propagation may lead to a tightening of the allowed span Δ_X between an external timepoint X and a contingent timepoint C with contingent oracle such that Δ_X becomes smaller than the contingent span Δ_C between the activation timepoint and the contingent timepoint. In this case, the new rules should not be applied, and any previously derived constraint c between A and X should be removed, as well as any other constraints derived from c.

A first possible solution to manage such a situation is presented in Algorithm 1, which is a loop with a first application of the subset of Morris and Muscettola's rules that are not replaced by our new rules, followed by the application of the new rules, and finally by the removal of any derived constraints incompatible with $\Delta_X < \Delta_C$. However, this backtracking approach can significantly slow down the AC-checking, as it requires removing some of the derived constraints with some rules and applying other rules again. This performance decrease of the new constraint propagation algorithm compared to a classical algorithm using only Morris and Muscettola's rules is even more pronounced when several contingent oracles are involved. On the positive side, the memory requirements of the new algorithm do not increase with respect to the classical algorithm.

In Section 4.3, we discussed an approach alternative to the backtracking algorithm. The approach requires labeling derived constraints based on the rules used to derive the constraints. With this approach, when a tightening such that $\Delta_X < \Delta_C$ holds, it is sufficient, in the propagation, to ignore constraints with specific labels without having to delete any derived constraints. This has the benefit of avoiding backtracking and the corresponding significant slowdown in the AC-checking. However, the price to pay in this case is in terms of memory, as more complex labels have to be managed.

We observe that the new rule set is not commutative, meaning that the result of a complete constraint propagation (i.e., until quiescence) based on the rule set depends on the order of application of the rules. An implication of this fact, leading to two possible approaches to check the dynamic controllability of an STNU with contingent oracles, is outlined here. More sophisticated and efficient approaches for AC-checking based on the proposed rule set are currently being investigated.

6 Conclusion

We have shown that there exist process models that the traditional STNU cannot represent faithfully with respect to the temporal aspect. In particular, this is the case of process models in which contingent durations may be known before their completion. This representational bias results in wrong results of the checks for dynamic controllability as such STNUs may be identified as not dynamically controllable when they actually are.

Here, we introduced a conservative extension of the STNU with contingent oracles and formalized agile controllability as an extension of dynamic controllability for such an extended STNU. We discussed possible algorithmic approaches to check the agile controllability of such networks. This initial discussion forms the basis for a research agenda to define sound and complete solutions to efficiently check the controllability of a broader set of process models.

References

[1] Claudio Bettini, X Sean Wang, and Sushil Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3):269–306, 2002.

[2] Carlo Combi, Mauro Gambini, Sara Migliorini, and Roberto Posenato. Representing business processes through a temporal data-centric workflow modeling language: An application to the management of clinical pathways. *IEEE Trans. Syst. Man Cybern. Syst.*, 44(9):1182–1203, 2014.

[3] Carlo Combi and Roberto Posenato. Controllability in temporal conceptual workflow schemata. In *Lecture Notes in Computer Science*, volume 5701, pages 64–79. 2009.

[4] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.

[5] Kagedan DJ, Ahmed M, Devitt KS, and Wei AC. Enhanced recovery after pancreatic surgery: a systematic review of the evidence. *HPB (Oxford)*, 17:11–17, 2015.

[6] Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of business processes with temporal variables. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 40–47, 2019.

[7] Johann Eder, Euthimios Panagos, and Michael Rabinovich. Workflow time management revisited. In *Seminal contributions to information systems engineering*, pages 207–213. Springer, 2013.

[8] Luke Hunsberger. Efficient execution of dynamically controllable simple temporal networks with uncertainty. *Acta Informatica*, 53:89–147, 2016.

[9] Luke Hunsberger, Roberto Posenato, and Carlo Combi. The Dynamic Controllability of Conditional STNs with Uncertainty. In *Workshop on Planning and Plan Execution for Real-World Systems: Principles and Practices (PlanEx) @ ICAPS-2012*, pages 21–29, June 2012.

[10] Andreas Lanz, Manfred Reichert, and Barbara Weber. Process time patterns: A formal foundation. *Inf. Syst.*, 57:38–68, 2016.

[11] Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *20th National Conf. on Artificial Intelligence (AAAI-2005)*, pages 1193–1198, 2005.

[12] Roberto Posenato, Francesca Zerbato, and Carlo Combi. Managing Decision Tasks and Events in Time-Aware Business Process Models. In *Business Process Management. BPM 2018*, volume 11080 of *LNCS*, pages 102–118. Springer, 2018.

[13] Thierry Vidal. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[14] Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.*, 11(1):23–45, jan 1999.