Agile Controllability of Simple Temporal Networks with Uncertainty and Oracles

Johann Eder
University of Klagenfurt, Austria

Roberto Posenato
University of Verona, Italy

Carlo Combi
University of Verona, Italy

Marco Franceschetti
University of St. Gallen, Switzerland

Franziska S. Hollauf
University of Klagenfurt, Austria

Abstract

Simple temporal networks with uncertainty (STNUs) have achieved wide attention and are the basis of many applications requiring the representation of temporal constraints and checking whether they are conflicting. Dynamic controllability is currently the most relaxed notion to check whether a system can be controlled without violating temporal constraints despite uncertainties. However, dynamic controllability assumes that the actual duration of a contingent activity is only known when the end event of this activity takes place. The recently introduced notion of agile controllability considers when this duration is known earlier, leading to a more relaxed notion of temporal feasibility. We extend the definition of STNUs to STNUOs (Simple Temporal Networks with Uncertainty and Oracles) to represent the point in time at which information about a contingent duration is available. We formally define agile controllability as a generalization of dynamic controllability considering the timepoints of information availability. We propose a set of constraint propagation rules for STNUOs leading to an algorithm for checking agile controllability.

2012 ACM Subject Classification Computing methodologies → Temporal reasoning; Theory of computation → Dynamic graph algorithms

Keywords and phrases Temporal constraint networks, contingent durations, agile controllability

Digital Object Identifier 10.4230/LIPIcs.TIME.2024.4

Supplementary Material *Software*: `https://git-isys.aau.at/ics/Papers/stnuo.git`
Dataset: `https://profs.scienze.univr.it/~posenato/software/benchmarks/OSTNUBenchmarks2024.tgz`

1 Introduction

Simple temporal networks with uncertainty (STNUs) [17, 4] have gained wide recognition for modeling temporal constraints. They extend Simple Temporal Networks (STNs) [5] by allowing one to represent uncertainties, i.e., they include so-called contingent activities, finishing at contingent timepoints and having a duration set by the environment of the considered system. System controllers only know the admissible range of such contingent durations, but they can only observe but do not decide on their actual values. STNUs are comparatively easy to use and easy to understand but expressive enough to represent temporal constraints for a large number of applications, for example, constraint-based planning [17], business processes [14], requirements engineering [7], or legal smart contracts [15], to name but a few.

An important question in all these applications is whether the constraints modeled in an STNU are temporally correct: can the controller derive a schedule for a process observing all constraints? Can the controller steer the execution of a process without violating any constraint? Are the elicited temporal requirements in conflict? Although the notion of satisfiability is sufficient for STNs, uncertainty in STNUs requires a more sophisticated notion of correctness. Dynamic controllability [17] is currently the most studied notion for the temporal correctness of STNUs. It requires a viable execution strategy (assignment of values to timepoints) that does not violate any constraint and where later timepoints may depend on earlier timepoints but not vice versa. For checking dynamic controllability, effective and efficient methods with polynomial complexity have been proposed [16, 4, 12].

However, dynamic controllability assumes that the duration of contingent activities, hence the values of contingent timepoints, are only known when they happen. This is adequate in many applications where the actual duration of some activity is only known when the end event of this activity is observed. An example of such an activity is a money transfer in the EU, where there is a legal requirement that the transfer does not last longer than four days. However, the actual duration is only known when the amount is credited to the receiver's account. In this case, the controller cannot schedule an event that has to occur exactly one day before the contingent timepoint (the receipt of the transferred amount) without (potential) violation of this temporal constraint.

However, in other applications, the duration of a contingent activity can be known earlier. For example, a delivery time between 4 and 6 weeks is guaranteed in order processing. However, the delivery date is communicated within a week after placing the order. In this case, it is perfectly feasible for the controller to schedule an event precisely two days before the contingent timepoint (day of the delivery). For such a scheduling decision, the controller recognizes that the duration of the contingent activity is known before the contingent timepoint takes place.

Now the question arises of how the notion of dynamic controllability can be generalized such that in a viable execution strategy, a timepoint may depend not only on timepoints which *are* earlier, but also which *are known* earlier. We call this novel notion of controllability *agile* because information about future timepoints may be used as soon as it is available.

This notion of *agile controllability* has first been introduced in [22, 21] together with an algorithm to check agile controllability based on the propagation rules in [16]. Here, we further formalize the extension of STNUs by introducing the notion of oracles, which represent the timepoints when the duration of a contingent activity is revealed. We formally define agile controllability by extending the notion of a viable execution strategy based on the available information of durations rather than only the occurrence of events. Furthermore, we develop an algorithm for checking agile controllability based on an extension of the constraint propagation rules presented in [4].

Therefore, the original contributions of this paper are:

1. The formal definition of STNUO (Simple Temporal Network with Uncertainty and Oracles).
2. A formal definition of Agile Controllability.
3. ORUL, a set of rules for propagating constraints in STNUOs.
4. An algorithm for checking Agile Controllability of STNUOs.
5. A proof-of-concept implementation of the checking algorithm.

The rest of the paper is organized as follows. In Section 2, we review related work and introduce the basic terms and definitions. In Section 3, we define STNUOs as an extension of STNUs with so-called oracles and define Agile Controllability as an extension of the notion of viable execution strategy. Section 4.1 discusses using oracles in execution strategies and

presents ORUL, a set of propagation rules to derive implicit constraints, and a backtracking algorithm to check agile controllability based on ORUL. Section 5 discusses the experimental evaluation of a proof-of-concept implementation of the checking algorithm, and finally, in Section 6, we draw some conclusions.

2 Background and Related work

2.1 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 comprises 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 [11]:

▶ **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 [5] to study the Simple Temporal Problem, that is, 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/environment*, who determines the *duration* of each contingent link. Therefore, given a contingent link (A, x, y, C), once the controller executes the activation timepoint A, the environment decides the duration $d \in [x, y]$ and reveals it at time $A + d$, that is $C = A + d$.

An important property of the STNU is the *dynamic controllability*. To define it, we must formally introduce some concepts we recall from [11].

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

▶ **Definition 3** (Schedule). *A* schedule *for an STNU $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is a mapping $\xi \colon \mathcal{T} \cup \{\perp\} \to \mathbf{R}$, where we assume that $\xi(\perp) = +\infty$. Ξ denotes the set of all schedules for an STNU. For historical reasons, we represent $\xi(X)$ as $[X]_\xi$.*

After having formally introduced the durations decided by *nature*, i.e., a *situation*, and the schedules of an STNU, i.e., the assignments of all timepoints to real values, we have to merge such aspects, to consider a strategy that, given a situation decided by nature, finds a suitable schedule.

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

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

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

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

where $S(\omega')^{\leq k}$ *is the set of contingent link durations observed up to and including time* k*, called* history[1] 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, that is, an execution strategy that assigns the executable timepoints with the guarantee that all constraints will be satisfied, irrespectively of the duration values (within the specified bounds) the contingent links will be revealed to take [11].

2.2 Checking Dynamic Controllability

Each STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ has a corresponding graph $\mathcal{G} = (\mathcal{T}, \mathcal{E}_{\text{o}} \cup \mathcal{E}_{\text{lc}} \cup \mathcal{E}_{\text{uc}})$, also called *distance graph*, where the timepoints in $\mathcal{T}$ serve as the graph's nodes and the constraints in $\mathcal{C}$ and $\mathcal{L}$ correspond to labeled, directed edges. In particular:

- $\mathcal{E}_{\text{o}} = \{X \xrightarrow{\delta} Y \mid (Y - X \leq \delta) \in \mathcal{C}\}$
- $\mathcal{E}_{\text{lc}} = \{A \xrightarrow{c:x} C \mid (A, x, y, C) \in \mathcal{L}\}$, 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 to distinguish them from the LC and UC edges.

Constraint propagation algorithms based on applying constraint propagation rules on the corresponding graph have been proposed to check whether an STNU is dynamically controllable [17, 16, 3, 12, 13]. A constraint propagation algorithm applies constraint propagation rules to derive implicit constraints from 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 cycle* is found (the network is not dynamically controllable). From now on, given a set of propagation rules R, we will call *closure* of a set $\mathcal{N}$ of temporal constraints the set of constraints derived by applying the propagation rules in R.

Morris and Muscettola were the first to propose in [17] an algorithm based on constraint propagation that exhibits time complexity $O(n^5)$. In contrast, Cairo et al. proposed in [3] a new set of rules that improve the time complexity of the dynamic controllability (DC) check

[1] Also called *pre-history* in previous work [11, 2].

Table 1 Edge-generation rules used by RUL [4, Fig. 2].

Rule	Pre-existing and generated edges	Applicability Conditions
(R)	$W \xrightarrow{v} Y \xrightarrow{u} X$, $W \xrightarrow{u+v} X$	none
(U)	$X \xrightarrow{v} C \underset{c:x}{\overset{C:\,-y}{\rightleftarrows}} A$, $X \xrightarrow{max\{v-y,-x\}} A$	$(A, x, y, C) \in \mathcal{L}$
(L)	$X \xleftarrow{v} C \xleftarrow{c:x} A$, $X \xleftarrow{x+v} A$	$v \leq 0$ or $X \in \mathcal{T}_C$, $X \not\equiv C$, $\exists (B, s, t, X) \in \mathcal{L}$, $v \leq t$

algorithm to $O(mn + k^2 n + kn \log n)$, where n is the number of timepoints, m is the number of constraints, and k is the number of contingent links. Table 1 shows the propagation rules proposed by Cairo et al., which we will use in the following.

2.3 Related Work: Making Contingent Links Flexible

The motivation to study the flexibility for contingent links comes from different application domains, such as business process modeling [23, 21, 20], robotics [24], and so on. Here, we briefly introduce different approaches to managing such flexibility.

The first, proposed in [23, 19], introduces the concept of guarded link: it is a contingent link, and thus its duration is not controlled by the system but has bounds that can be shrunk until some specific durations, named guards. In [23], the authors extend STNUs' propagation rules to deal with such guarded links. In contrast, in [19], the propagation rules for checking DC are also extended to deal with conditional execution paths, where, according to some conditions set during the network execution, only specific time points are executed.

In a second research line, in [1], the authors discuss some degrees of strong and dynamic controllability for STNUs, evaluating how far a network is from being controllable. Such metrics approximate the probability that an STNU can be dispatched offline (strong controllability) or online (dynamic controllability). Here, the focus is on uncontrollable networks. Such metrics are further generalized to Probabilistic Simple Temporal Networks (PSTNs). Taking into account even more recent research results, in [24], the authors discuss the robustness measure of PSTNs, that is, the probability of success in execution. They introduce and discuss degrees of weak/strong/dynamic controllability, robustness under a predefined dispatching protocol, and the PSTN expected execution utility.

There are several approaches that allow the representation of a timepoint when a contingent duration is revealed. The approach proposed in [25] proposes weak controllability, where all contingent durations are already known at the beginning of the process. Weak controllability can be seen as a special case of agile controllability proposed here.

The temporal variables considered in [6] are means of receiving temporal information from the process environment, for example, as output of process activities.

In [9], the authors introduce a further character of flexibility in the context of temporal business processes, making the durations of non-contingent activities known earlier. Indeed, they introduce and discuss the concept of semi-contingent task duration: it is a duration under the system's control until the task is initiated. Then, such duration becomes only observable but not under the system's control. Simple Temporal Networks with Semi-Contingency and Uncertainty (STNSUs) are then introduced, and dynamic controllability is studied for this new kind of temporal constraint network.

In [8, 10], the authors introduce a further flexibility aspect as they extend STNUs and Conditional Simple Temporal Networks with Uncertainty (CSTNUs) also to include a new kind of timepoints named *parameters*, whose occurrence must be fixed as soon as the network execution starts. A dynamic controllability check algorithm is proposed for this new kind of network.

3 Extending STNUs with Oracles

As the STNU does not allow decoupling the value of a contingent duration and the time of occurrence of the associated timepoint, we introduce a new kind of timepoint called *oracle*. An oracle O_C is a timepoint associated with a contingent link (A, C). When O_C is executed, it reveals the associated contingent link duration. In other words, O_C can reveal the duration of the contingent link before the contingent timepoint C occurs. We extend the formal definition of an STNU in [11] with oracles as follows:

▶ **Definition 7** (STNU with Oracles). *An STNU with Oracles (STNUO) is a tuple* $(\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$, *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.* $\mathcal{T}_O \subseteq \mathcal{T}_X$, *is the set of oracle 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* $C_1 \neq C_2$.
- $\mathcal{O}\colon \mathcal{T}_C \to \mathcal{T}_O \cup \{\bot\}$ *is a function that associates a contingent timepoint with its corresponding oracle, if any. For the sake of simplicity and without loss of generality, we assume that each oracle is associated with a single contingent timepoint.*

The environment decides the duration d of a contingent link (A_i, x_i, y_i, C_i), revealed at time $A_i + d$ or when the associated oracle O_i is executed. The requirement that only non-contingent nodes can be oracles does not reduce expressiveness, as oracles can be closely linked to contingent nodes. If $\mathcal{O}(C) = \bot$, the contingent node C does not have an oracle.

In the following, we extend the concept of dynamic execution strategy, replacing the concept of history with the concept of *Oracle-extended History* (OH) to consider also the presence of oracle timepoints. Therefore, we prefer to call this new dynamic execution strategy as *agile execution strategy*. Then, we introduce the concept of Agile Controllability.

The definitions of situation, schedule, execution strategy, and viable execution strategy are straightforward extensions of Definitions 2–5, respectively, to also include oracle time points.

In the definition of dynamic controllability, the *history until* k is the set of all contingent durations whose contingent timepoints occurred before or at time k. For STNUOs, the concept of history must also include all the durations revealed by oracles executed before or at k. Therefore, we call such history as *Oracle-extended History (OH) at time* k. Thus, OH contains information about the past and already-known information about the future.

▶ **Definition 8** (Oracle-extended History (OH)). *Given a schedule* ξ *for an STNUO* $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$, *and a time* k, *the Oracle-extended History (OH) until* k *is:*

$$\xi^{\leq k} = \{\omega_i \mid \omega_i = [C]_\xi - [A]_\xi \text{ and } \min\{[C]_\xi, [\mathcal{O}(C)]_\xi\} \leq k\}.$$

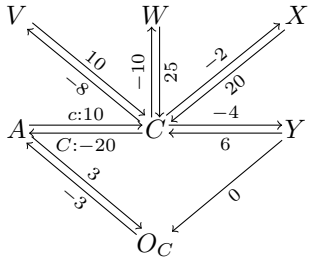


Figure 1 An STNUO with contingent link $(A, 10, 20, C)$. Oracle O_C is associated to C.

▶ **Definition 9** (Agile Execution Strategy with Oracles). *Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ be an STNUO. An* execution strategy with oracles S_O *for $\mathcal{N}$ is* agile *if, for any two situations ω', ω'' and any executable timepoint $X \in \mathcal{T}$, it holds that:*

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

where $S_O(\omega)$ is a schedule determined by the execution strategy with oracles S_O given the situation ω, and $S_O(\omega)^{\leq k}$ is OH until k. Since OH also considers contingent durations observed and revealed until time k, we say that the dynamic execution strategy implements the instantaneous reaction *semantics.*

▶ **Definition 10** (Agile Controllability (AC)). *An STNUO $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ is* agilely controllable *if it admits a viable agile execution strategy with oracles. We refer to agile controllability (AC) as the property of being agilely controllable.*

▶ **Example 11.** Let us consider an STNUO $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ as depicted in Figure 1, where O_C is the oracle for C that must be executed 3 time units after the activation timepoint A. Let $d \in [10, 20]$ be the duration revealed by the oracle O_C.
V can neither be scheduled with nor without an oracle because if the contingent link lasts 10, the oracle is executed too late to allow V to be executed, satisfying the constraint with C. W must be scheduled before the oracle to satisfy the constraint with C. Therefore, the oracle is not relevant for scheduling W. X can be scheduled without oracle (for example, $X = A + 2$) or with oracle (for example, $X = O_C - 3 + d - 4 = A + d - 4$, where 4 is one of the possible values to choose.) Y can be scheduled only with oracle: $Y = O_C - 3 + d - 5 = A + d - 5$. Thus, Y can be executed only after O_C.

The notion of agile controllability is strictly more general than the notion of dynamic controllability.

▶ **Lemma 12.** *Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ be an STNUO. If the STNU $\mathcal{N}' = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ is dynamically controllable, then $\mathcal{N}$ is agilely controllable.*

Proof. The lemma follows directly from the definition as if $\mathcal{N}$ satisfies the requirements for dynamic controllability; it also satisfies those of agile controllability, as any viable dynamic execution strategy is also a viable agile execution strategy. ◀

4 Checking Agile Controllability of STNUO

This section presents a procedure for checking whether an STNUO is agilely controllable. The procedure is based on the rules introduced in Table 1. For the following considerations, let X be a non-contingent node, C a contingent node (with activation node A), and m/M the minimum/maximum duration of the considered contingent link (A, m, M, C).

4.1 On The Usage of Oracles

The canonical problem we deal with here arises when a timepoint X depends on a future contingent timepoint C, i.e., there is a constraint $X \leq C + \delta$, where $\delta < 0$. The standard STNU semantics assumes that the value of a contingent timepoint, here C, is only revealed when C occurs. Therefore, the value for X must not violate the constraint for any possible C, i.e., $\exists X \forall C \mid X \leq C + \delta$. Such a set is only possible if $X \leq A + m + \delta$, which is the constraint derived by the L rule in Table 1, i.e., at the latest, X has to be executed at least δ time units before the earliest execution of C.

On the other hand, if there is a constraint $C \leq X + \delta'$, $0 \leq \delta'$, then STNU is DC only if $M - m \leq \delta + \delta'$, that is, if the difference between the smallest and the greatest distance between X and C is larger than the contingency of C (i.e., the difference between the maximum and minimum distances between A and C); otherwise, the constraints conflict.

▶ **Example 13.** The node X in Figure 1 can be scheduled since the contingency of C is 10 smaller than 18, the range of possible distances between C and X. For node Y, these values are 10 and 2. Applying the rules U and L to X, C, and A leads to a negative cycle.

However, if the duration of contingent activity is revealed earlier by an oracle at time point O_C, then it is sufficient that $\forall C \exists X \mid X \leq C + \delta$. However, the following constraint has to hold: $O_C \leq X$ and consequently $O_C \leq C + \delta$. So, the essence of using an oracle is that the sequence of quantifiers is changed from $\exists X \forall C$ to $\forall C \exists X$. However, the price is the introduction of an additional constraint, which could conflict with other constraints.

For the propagation of constraints, this has the following consequences:

(1) If the oracle is not used, then the L and U rules must be applied.

(2) If the oracle is used, then the L and U rules must not be applied on X and C, but the additional constraint $O_c \leq X$ must be added to the OSTNU.

Generally, the STNUs derived in (1) or (2) are not equivalent and admit different closures. Moreover, one closure could contain a negative cycle, and the other not.

▶ **Example 14.** Using the oracle avoids the negative cycle resulting from propagation in case (1), such as Y in Figure 1. On the other hand, using the oracle as in case (2) might lead to a conflict that was not there, such as W in Figure 1.

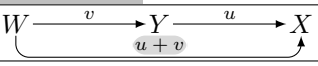
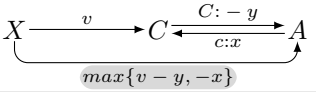
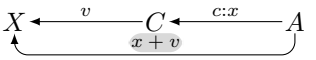
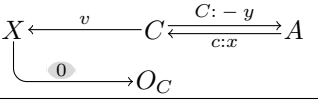
These possible configurations are all considered in the definition of a viable execution strategy: the value of X may be a function of the duration of a contingent activity d if C is in the history of X (i.e., is before X), or O_C is in the oracle-extended history of X (i.e., O_C is before X). The disjunction in the definition of oracle-extended history also leads to a choice in applying rules to propagate constraints.

▶ **Example 15.** In Figure 1, for timepoint X, there is the choice to either apply the rules L and U or consider the oracle O_C and introduce constraint $O_c \leq X$.

An interesting question is determining when an oracle is necessary to guarantee Agile Controllability. Basically, we can only use an oracle if there is one for a contingent node. Then, we only have to consider whether to use an oracle if there is a negative link from the considered contingent node to a non-contingent node. This link could be the result of constraint propagation.

We call $\mathcal{U} = \{(X, C) \mid X \in \mathcal{T}_X, C \in \mathcal{T}_C, \mathcal{O}(C) \neq \bot\}$ the set of all **potential oracle candidates**. If there is constraint $X \leq C + \delta$, or a constraint $C \leq X + \delta'$ with $C \in \mathcal{T}_C$, $\mathcal{O}(C) \neq \bot$, $X \in \mathcal{T}_X$, $\delta < 0$, and $0 \leq \delta'$, then we call (X, C) an ***oracle candidate*** since a viable execution strategy *could* require the usage of the oracle.

Table 2 ORUL: propagation rules extending RUL ones for checking Agile Controllability of STNUO. $\mathcal{U}^+$ is the set of all oracle candidates for which the oracle should be used, and $\mathcal{U}^-$ is the set of all oracle candidates for which the oracle should not be used.

Rule	Pre-existing and generated edges	Conditions
Relax (REL)		none
Upper (UPP)		$(X,C) \in \mathcal{U}^-$ or $\mathcal{O}(C) = \bot$ or $X \in \mathcal{T}_C$
Lower (LOW)		$X \in \mathcal{T}_\mathcal{X}$, $((X,C) \in \mathcal{U}^-$ or $\mathcal{O}(C) = \bot)$, $v \leq 0$, or $X \in \mathcal{T}_C$, $X \not\equiv C$, $\exists (B,w,y,X) \in \mathcal{L}$, $v \leq y$
Oracle (ORC)		$X \in \mathcal{T}_X$, $(X,C) \in \mathcal{U}^+$

▶ **Example 16.** In Figure 1, all pairs $(V,C), (W,C), (X,C), (Y,C)$ are oracle candidates.

Now consider the case where there are a pair of constraints, $X \leq C + \delta$ and $C \leq X + \delta'$, with $\delta < 0$, and $\delta + \delta' \leq M - m$, the contingency of C. In such a configuration, there is no viable execution strategy without using the oracle O_C. Therefore, we call (X,C) ***oracle dependent***.

▶ **Example 17.** In Figure 1, (Y,C) is oracle dependent.

Constraints can only become stricter (and never removed) during constraint propagation. Therefore, if an oracle candidate (X,C) becomes oracle dependent due to the propagation of constraints, it will remain oracle dependent.

While for oracle-dependent pairs, any viable solution must use the oracle, constraint propagation could, but not necessarily, make oracle dependent some oracle candidates. Whether a pair becomes an oracle candidate or oracle dependent might also depend on which oracles are used for constraint propagation and which are not.

As there is no way of deciding upfront whether a (X,C) pair will become oracle dependent, it will be necessary for oracle candidates, which are not oracle dependent, to explore both options: to use oracle and not to use the oracle. For a particular constraint propagation, it is necessary to decide for which (X,C) pairs to use the oracle.

4.2 Propagation rules for STNUOs

In the previous section, we argued that exploring whether an oracle has to be used for an oracle candidate might be necessary. For guiding the propagation of constraints, we maintain two sets of oracle candidates:

- $\mathcal{U}^+$ is the set of all oracle candidates for which the oracle has to be used, and
- $\mathcal{U}^-$ is the set of all oracle candidates for which the oracle has not to be used.

In Table 2, we propose ORUL, a set of constraint-propagation rules based on RUL set [4], modified for checking agile controllability. ORUL uses sets $\mathcal{U}^+$ and $\mathcal{U}^-$ to guide the rules' application. Briefly, the REL rule is the same as the R rule in Table 1. The conditions for the UPP and LOW rules are extended with additional restriction $(X,C) \in \mathcal{U}^-$ such that the rules are only applied if oracles should *not* be used. The new ORC rule inserts the necessary constraints for using an oracle when $(X,C) \in \mathcal{U}^+$.

The following theorem states conditions on $\mathcal{U}^+$ and $\mathcal{U}^-$ to guarantee AC.

▶ **Theorem 18.** *Let $\mathcal{N} = (\mathcal{T}, \mathcal{C}, \mathcal{L}, \mathcal{O})$ be an STNUO and let $\mathcal{U} = \{(X, C) \mid X \in \mathcal{T}_X, C \in \mathcal{T}_C, \mathcal{O}(C) \neq \bot\}$ be the set of all possible pairs (ordinary node, contingent node) where the contingent node has its corresponding oracle.*

$\mathcal{N}$ is Agilely Controllable if $\exists\, \mathcal{U}^+, \mathcal{U}^-$ such that $\mathcal{U} = \mathcal{U}^+ \cup \mathcal{U}^-$, $\mathcal{U}^+ \cap \mathcal{U}^- = \emptyset$, and the closure of $\mathcal{N}$ considering $\mathcal{U}^+$, $\mathcal{U}^-$ for the propagation rules in Table 2 does not include a negative cycle.

Proof Sketch. We show that ORUL is sound, i.e., if ORUL does not lead to a negative cycle, then the STNUO is AC. It is straightforward to show that the propagated constraints are finite if no negative cycle is derived. Hence, let $\mathcal{N}^*_{\mathcal{O}}$ be the closure of $\mathcal{N}$ with respect to ORUL, and $\mathcal{N}^*_{\mathcal{R}}$ be the set of all constraints derived by RUL. Cairo and Rizzi showed that the constraints of $\mathcal{N}^*_{\mathcal{R}}$ are necessary and sufficient to decide whether $\mathcal{N}$ is DC [4]. We show that $\mathcal{N}^*_{\mathcal{O}}$ contains all the constraints of $\mathcal{N}^*_{\mathcal{R}}$ except those that are not necessary if oracles are used, while it includes all the constraints needed for the oracle candidates in $\mathcal{U}^+$. It also excludes all constraints that are derived from unnecessary ones. Hence, $\mathcal{N}^*_{\mathcal{O}}$ is sufficient to decide whether $\mathcal{N}$ is AC.

If $\mathcal{U}^+ = \emptyset$, and thus $\mathcal{U}^- = \mathcal{U}$, then the rules in Table 2 are the same as in Table 1, and $\mathcal{N}^*_{\mathcal{R}} = \mathcal{N}^*_{\mathcal{O}}$. As in [4], we can conclude that the STNUO is DC and hence AC if no negative cycle is derived.

If $\mathcal{U}^+ \neq \emptyset$, let $(X, C) \in \mathcal{U}^+$, $(A, m, M, C) \in \mathcal{L}$, $(X - C \leq \delta) \in \mathcal{C}, (C - X \leq \delta') \in \mathcal{C}$, $\delta < 0 < \delta', 0 \leq \delta' + \delta < M - m$, and $\mathcal{O}(C) = O$. Since oracle O is used for X, $(O - X \leq 0) \in \mathcal{N}^*_{\mathcal{O}}$ and the LOW and UPP rules are not applied for the triple $\langle X, A, C\rangle$. Hence, the LOW- and UPP-derived constraints between A and X are not in $\mathcal{N}^*_{\mathcal{O}}$. For a viable execution strategy, these constraints are not necessary, as for any duration d of the contingent activity (A, m, M, C), $C = A + d$. There exists a value for X that satisfies constraints $(X - C \leq \delta)$ and $(C - X \leq \delta')$. In fact, both X and A are executable timepoints, and d is available before X, therefore $X = A + d + \delta$ is admissible and allows for the satisfaction of constraints $(X - C \leq \delta)$ and $(C - X \leq \delta')$. Therefore, if the propagation of such constraints does not determine negative cycles, then the network is AC.

We may also observe that any constraint that can be derived from $\mathcal{N}$ by rules in Table 1 which is not in $\mathcal{N}^*_{\mathcal{O}}$, would be the result of a sequence of rule applications, starting by applying the UPP or LOW rules to some $\langle X, A, C\rangle$, where (X, C) in $\mathcal{U}^+$, and hence is not necessary. ◀

4.3 A Checking Algorithm

We propose the backtracking Algorithm 1 to check whether an STNUO is agilely controllable.

The algorithm aims to check whether there exist sets $\mathcal{U}^+$ and its complementary $\mathcal{U}^-$ such that the propagation with these sets does not lead to a negative cycle. As outlined at the end of Section 4.1, there is no way to know in advance sets $\mathcal{U}^+$ and $\mathcal{U}^-$. Therefore, the algorithm computes these sets incrementally and with backtracking. To reduce the effort of backtracking, the general strategy is to delay decisions about *potential* membership of a pair (X, C) in these sets as late as possible, but for those having an explicit condition for membership to $\mathcal{U}^+$ or $\mathcal{U}^-$. For this reason, $\mathcal{U}^+$ and $\mathcal{U}^-$ are both empty when the algorithm starts, and only at the end, if the network is agilely controllable, they satisfy the conditions of Theorem 18[2].

[2] More precisely, $\mathcal{U}^+ \cap \mathcal{U}^- = \emptyset$ and $\mathcal{U}^+ \cup \mathcal{U}^- \subseteq \mathcal{U}$. Indeed, the algorithm does not consider pairs (X, C) for which there are no explicit constraints. Such possible pairs are not effective with respect to constraint propagations.

Algorithm 1 CheckAC.

```
   Input: 𝒩 an STNUO, 𝒰⁻, 𝒰⁺
   Output: Agile controllability status
 1 ok ← applyRules(𝒩,𝒰⁻,𝒰⁺);
 2 if ok then
 3     save(𝒩,𝒰⁻,𝒰⁺);
 4     𝒰⁰ ← getOpenOracles(𝒩,𝒰⁻,𝒰⁺);
 5     if 𝒰⁰ ≠ ∅ then
 6         (X,C) ← select(𝒰⁰);
 7         if interval(X,C) > contingencyInterval(C) then
 8             𝒰⁻ ← 𝒰⁻ ∪ {(X,C)};
 9             ok ← checkAC(𝒩,𝒰⁻,𝒰⁺);
10             if not ok then
11                 restore(𝒩,𝒰⁻,𝒰⁺);
12         if (X,C) ∉ 𝒰⁻ then
13             𝒰⁺ ← 𝒰⁺ ∪ {(X,C)};
14             ok ← checkAC(𝒩,𝒰⁻,𝒰⁺);
15             if not ok then
16                 restore(𝒩,𝒰⁻,𝒰⁺);
17 return ok
```

The algorithm is recursive. It starts with applying all the rules in Table 2. If the check is positive (no negative cycle was discovered), it determines which oracle-dependent pairs have not yet been considered and tries to assign each to $\mathcal{U}^-$ or $\mathcal{U}^+$ recursively. When a negative cycle is discovered, the STNUO with the current sets $\mathcal{U}^+$ and $\mathcal{U}^-$ is not agilely controllable and, therefore, backtracking is required (procedure `restore`). If there is no negative cycle, the algorithm checks whether there are still undecided oracle candidates. Heuristically, one undecided candidate pair (X, C) is chosen. Suppose a solution without applying the oracle for this pair is still possible (no oracle dependency). In that case, we decide not to use the oracle (i.e., inserting (X, C) in $\mathcal{U}^-$) and proceed recursively, invoking the checking procedure. If it fails, we restore the STNUO and continue with the decision to use the oracle (i.e., inserting (X, C) in $\mathcal{U}^+$) and recursively invoke the checking procedure.

When constraints lead to an oracle-dependent pair (X, C) (line 12 of Algorithm 1), such a pair is inserted into $\mathcal{U}^+$. When a constraint $C \xrightarrow{v} X$ with a non-positive v is derived, (X, C) is inserted into $\mathcal{U}^-$ (line 8 of Algorithm 1). The procedure terminates when either no additional constraints can be derived, and the procedure returns true, or a negative cycle is detected, and the procedure returns false.

The algorithm uses the following auxiliary procedures.

`getOpenOracles`$(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$

It returns the set of oracle candidates (X, C), i.e., the set of all pairs (X, C) for which either a constraint $C \xrightarrow{v} X$ with a non-positive v or a constraint $X \xrightarrow{w} C$ with a non-negative w exist and (X, C) is neither in $\mathcal{U}^-$ nor in $\mathcal{U}^+$.

`select`$(\mathcal{U}^0)$

It returns one of the oracle candidate pairs (X, C), not yet in $\mathcal{U}^-$ neither in $\mathcal{U}^+$. Heuristically, it returns the pair with the smallest difference between its interval and the contingency interval of C.

`applyRules`$(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$

It iteratively applies rules of Table 2 to generate additional constraints. It returns false if a negative cycle is discovered; true, otherwise.

`save`$(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$

It saves the current set of constraints such that `restore`$(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ can reconstruct the set of constraints as they were when the corresponding `save`$(\mathcal{N}, \mathcal{U}^-, \mathcal{U}^+)$ was executed. These procedures support backtracking if a negative cycle is found based on a decision on the inclusion of pairs in $\mathcal{U}^-$ resp. $\mathcal{U}^+$.

`interval`(X, C)

It returns the value $\delta + \delta'$ derived by the constraints $(X - C \leq \delta)$ and $(C - X \leq \delta')$.

`contingencyInterval`(C)

It returns the value $u - l$ relative to the contingent link (A, l, u, C) associated with the input contingent timepoint C.

▶ **Example 19.** Figure 2 presents the process of applying the algorithm on an example STNUO where $(A, 20, 30, C)$ is a contingent link, O_C is its oracle, and X and Y are non-contingent nodes. Propagated self-loops (e.g., from X to X) are excluded from the figure for readability reasons except for negative cycles. In one step, newly propagated constraints by the rules in Table 2 are depicted as follows: rules propagated by REL are colored gray, by UPP orange, by LOW olive, and by ORC blue. Negative cycles are marked in red. Pre-existing or pre-propagated constraints are black.

The algorithm starts with empty $\mathcal{U}^+$ and $\mathcal{U}^-$. In the first step (line 1 of Algorithm 1), the rules are applied by propagating three new constraints via REL as presented in Figure 2 (step I). This intermediate STNUO is checked for a negative cycle. Since it does not have one (is *ok*), the algorithm proceeds with saving this state and getting the open oracles (lines µ3 and 4 of Algorithm 1). $\mathcal{U}^0$ includes then the following pairs: (X, C) and (Y, C).

Next, (X, C) is selected to check whether to use the oracle for it (line 6 of Algorithm 1). In this case, `interval`$(X, C) = 15 - (-4) = 19$ and `contingencyInterval` $(X, C) = 30 - 20 = 10$. Since the contingency interval is smaller, (X, C) is put in $\mathcal{U}^-$ (lines 7-9 of Algorithm 1).

`CheckAC` calls itself. The rules are applied, and new constraints are propagated via REL as presented in Figure 2 (Step II). The STNUO now has a negative cycle. Indeed, the self-cycle in A equal to -1 is derived. More precisely, applying rule REL on timepoints A, X, and C, we obtain $(A, C) = 16 + 13 = 29$; then, applying rule UPP on A, C, A, we obtain the negative self-cycle $29 - 30 = -1$ in A.

Thus, the status is not *ok*, and the algorithm backtracks (lines 10 and 11 of Algorithm 1): The STNUO from step (I) is restored (see step III in Figure 2) and (X, C) is removed from $\mathcal{U}^-$. Thus, the algorithm checks what happens if the oracle is used for (X, C) (lines 12-16 of Algorithm 1). (X, C) is put in $\mathcal{U}^+$.

`CheckAC` calls itself. The rules are applied again, and new constraints are propagated (see step IV in Figure 2). This STNUO is without a negative cycle (status is *ok*). The algorithm stores this state and gets again the open oracles. $\mathcal{U}^0$ now includes only (Y, C). For this pair, `interval`$(Y, C) = \infty$ (since there is only the constraint $C - Y \leq 9$) and `contingencyInterval`$(Y, C) = 30 - 20 = 10$. Since the contingency interval is smaller, (Y, C) is put in $\mathcal{U}^-$.

`CheckAC` calls itself again, and the rules are applied (see Step V in Figure 2). No constraint leading to a negative cycle was propagated (status is *ok*). This state is saved, and the algorithm checks for open oracles. None are found; hence, $\mathcal{U}^0$ is empty. The algorithm ends with the result that the STNUO is *agilely controllable*.

4.3.1 Computational Complexity

Since `CheckAC` is a recursive backtracking algorithm, it is possible to give an upper bound to its space complexity assuming the worst case that each oracle must be paired with each other timepoint. In such a case, the depth of the recursion is $O(kn)$ (assuming that each of k contingent timepoints has an oracle). For each level of recursion, it is necessary to store

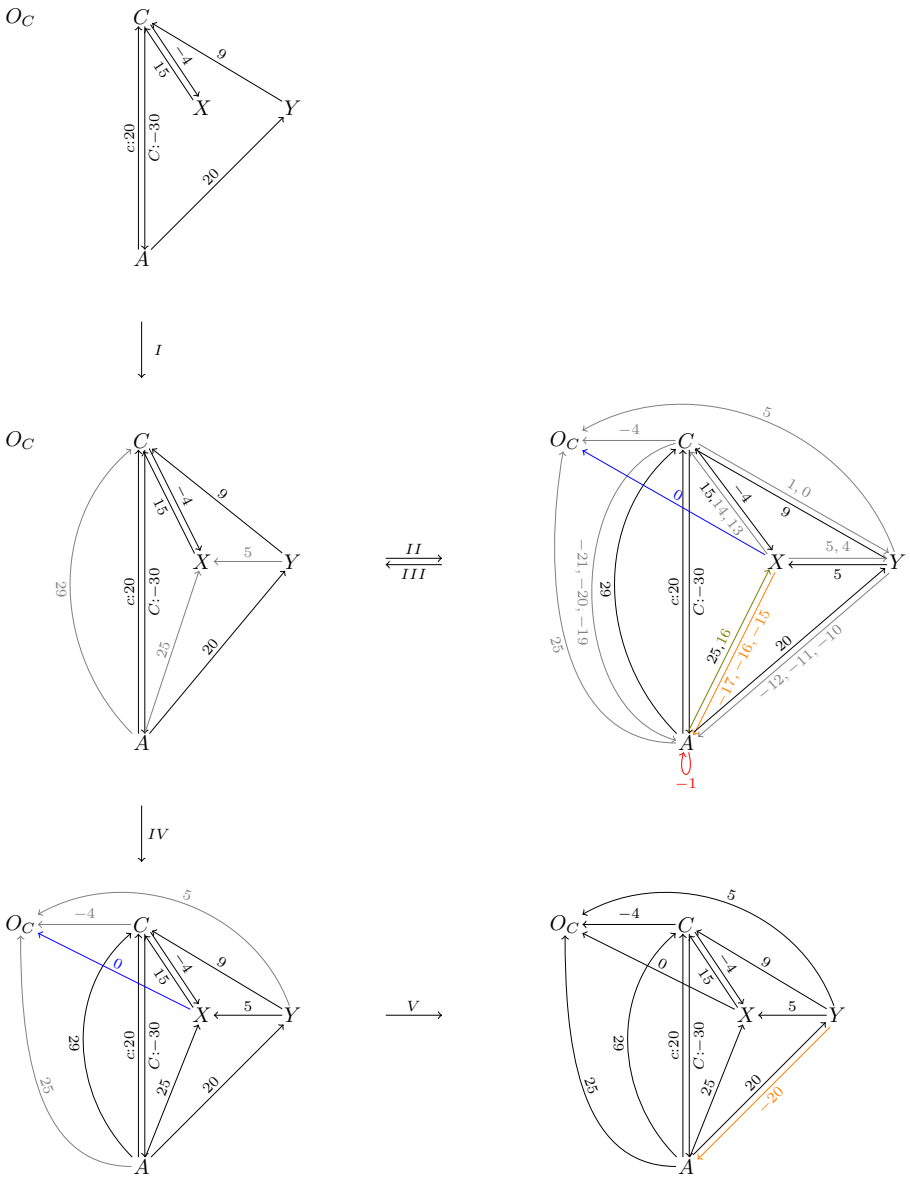


Figure 2 Applying CheckAC on an STNUO example.

(`save` procedure) the configuration of the network and the two sets $\mathcal{U}^+$ and $\mathcal{U}^-$. Such an operation requires space $O(n^2)$, where n is the number of all timepoints. Therefore, the computational space required by the algorithm is $O(kn^3)$.

5 Experimental Evaluation

The algorithm for checking the agile controllability of an STNUO presented in Section 4.3 has been implemented as a proof-of-concept prototype to test the algorithm's correctness and analyze its feasibility.

The algorithm has been implemented in Java. Experiments with this implementation were executed on an Ubuntu 22 machine having 16GB of RAM and an AMD EPYC-Rome (8) @ 2.6GHz CPU. The experiments use as input data from the OSTNU benchmark, which is available online[3].

The benchmark includes 30 random STNUO instances with 30 nodes (5 contingent and 2 oracles). In all cases, the implementation produced the correct result. We ran the checking algorithm 100 times on each example and determined the average AC checking execution time. All execution average times are below 3 s. Therefore, our approach to determining the AC property is comparable to that presented by Posenato *et al.* [22]. This allows us to conclude that the algorithm is feasible for realistically sized STNUOs. Nevertheless, we will continue optimizing the algorithm and its implementation. For example, we will want to consider the DC checking algorithm in [13], which implements a DC checker based on the rules in Table 1 in a more efficient way, for the `applyRules`($\mathcal{N}$,$\mathcal{U}^-$,$\mathcal{U}^+$) procedure, and to improve the heuristics for the `select`($\mathcal{U}^0$) procedure.

The source code of the prototype implementation, the parser of the data sets used for the experiments, and the complete results are publicly available in an online repository[4].

6 Discussion and Conclusions

We proposed agile controllability (AC) as a proper generalization of the well-established notion of dynamic controllability for STNUs, leading to a more relaxed notion of temporal correctness, which is still strong enough to guarantee that a controller can steer the execution of a process in a way that no temporal constraint is violated despite uncertainties. The main distinction to dynamic controllability is that available information about *future* durations of contingent links can be utilized for scheduling and dispatching timepoints. STUNOs, Simple Temporal Networks with Uncertainty and Oracles, can express *when* information about the timepoint of future contingent links is available. We presented a formal definition of viable execution strategies that utilize such advanced information. We also presented a set of rules for propagating constraints, leading to an algorithm that effectively checks whether an STNUO is agilely controllable.

AC is expected to support a wide range of applications as it provides a less restrictive notion of temporal correctness of plans, processes, requirements, contracts, etc. The presented algorithm seems feasible for typical problem sizes in many of these application areas. Nevertheless, improving implementations of this algorithm will further extend the approach's applicability. Indeed, the algorithm `CheckAC` and its related proof-of-concept implementation,

[3] `https://profs.scienze.univr.it/~posenato/software/benchmarks/OSTNUBenchmarks2024.tgz` [18]
[4] `https://git-isys.aau.at/ics/Papers/stnuo.git`

as it is presented here, is intended to demonstrate the existence of an effective backtracking algorithm to check the agile controllability of an STNUO. It is not optimized, and exploring numerous possibilities to develop a significantly more efficient implementation of this basic algorithm is the subject of ongoing research.

References

1 Shyan Akmal, Savana Ammons, Hemeng Li, Michael Gao, Lindsay Popowski, and James C. Boerkoel Jr. Quantifying controllability in temporal networks with uncertainty. *Artif. Intell.*, 289:103384, 2020. doi:10.1016/J.ARTINT.2020.103384.

2 Arthur Bit-Monnot and Paul Morris. Dynamic Controllability of Temporal Plans in Uncertain and Partially Observable Environments. *Journal of Artificial Intelligence Research*, 77:1311–1369, August 2023. doi:10.1613/jair.1.13065.

3 Massimo Cairo, Luke Hunsberger, and Romeo Rizzi. Faster dynamic controllability checking for simple temporal networks with uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*, volume 120 of *LIPIcs*, pages 8:1–8:16, 2018. ISBN: 9783959770897. doi:10.4230/LIPIcs.TIME.2018.8.

4 Massimo Cairo and Romeo Rizzi. Dynamic controllability of simple temporal networks with uncertainty: Simple rules and fast real-time execution. *Theoretical Computer Science*, 797:2–16, 2019. doi:10.1016/J.TCS.2018.11.005.

5 Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991. doi:10.1016/0004-3702(91)90006-6.

6 Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of business processes with temporal variables. In *ACM SAC 2019*, pages 40–47, 2019. doi:10.1145/3297280.3297286.

7 Johann Eder, Marco Franceschetti, and Josef Lubas. Time and processes: Towards engineering temporal requirements. In *Proceedings of the 16th International Conference on Software Technologies, ICSOFT 2021*, pages 9–16. SCITEPRESS, 2021. doi:10.5220/0010625400090016.

8 Marco Franceschetti and Johann Eder. Checking temporal service level agreements for web service compositions with temporal parameters. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 443–445. IEEE, 2019. doi:10.1109/ICWS.2019.00080.

9 Marco Franceschetti and Johann Eder. Semi-contingent task durations: Characterization and controllability. In Marcello La Rosa, Shazia W. Sadiq, and Ernest Teniente, editors, *Advanced Information Systems Engineering - 33rd International Conference, CAiSE 2021, Melbourne, VIC, Australia, June 28 - July 2, 2021, Proceedings*, volume 12751 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2021. doi:10.1007/978-3-030-79382-1_15.

10 Marco Franceschetti, Roberto Posenato, Carlo Combi, and Johann Eder. Dynamic Controllability of Parameterized CSTNUs. In *ACM SAC 2023*, pages 965–973, 2023. doi:10.1145/3555776.3577618.

11 Luke Hunsberger. Efficient execution of dynamically controllable simple temporal networks with uncertainty. *Acta Informatica*, 53:89–147, 2016. doi:10.1007/S00236-015-0227-0.

12 Luke Hunsberger and Roberto Posenato. Speeding up the RUL$^-$ Dynamic-Controllability-Checking Algorithm for Simple Temporal Networks with Uncertainty. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, volume 36, pages 9776–9785. AAAI Press, 2022. doi:10.1609/aaai.v36i9.21213.

13 Luke Hunsberger and Roberto Posenato. A Faster Algorithm for Converting Simple Temporal Networks with Uncertainty into Dispatchable Form. *Information and Computation*, 293, June 2023. doi:10.1016/j.ic.2023.105063.

14 Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Simple Temporal Networks with Partially Shrinkable Uncertainty. In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence (ICAART 2015)*, volume 2, 2015. doi:10.5220/0005200903700381.

15 Josef Lubas and Johann Eder. A time-aware model for legal smart contracts. In Han van der Aa, Dominik Bork, Henderik A. Proper, and Rainer Schmidt, editors, *Enterprise, Business-Process and Information Systems Modeling*, pages 121–135, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-34241-7_9.

16 Paul Morris. Dynamic controllability and dispatchability relationships. In *CPAIOR 2014*, volume 8451 of *LNCS*, 2014. doi:10.1007/978-3-319-07046-9_33.

17 Paul H. Morris and Nicola Muscettola. Temporal dynamic controllability revisited. In *20th National Conf. on Artificial Intelligence (AAAI-2005)*, 2005. URL: https://cdn.aaai.org/AAAI/2005/AAAI05-189.pdf.

18 Roberto Posenato. CSTNU Tool: A Java library for checking temporal networks. *SoftwareX*, 17:100905, 2022. doi:10.1016/j.softx.2021.100905.

19 Roberto Posenato and Carlo Combi. Adding flexibility to uncertainty: Flexible simple temporal networks with uncertainty (FTNU). *Inf. Sci.*, 584:784–807, 2022. doi:10.1016/J.INS.2021.10.008.

20 Roberto Posenato and Carlo Combi. Flexible temporal constraint management in modularized processes. *Inf. Syst.*, 118:102257, 2023. doi:10.1016/J.IS.2023.102257.

21 Roberto Posenato, Marco Franceschetti, Carlo Combi, and Johann Eder. Some results and challenges Extending Dynamic Controllability to Agile Controllability in Simple Temporal Networks with Uncertainties. TechRep 1/2023, Dip. Informatica-Univ. di Verona, 2023. URL: https://iris.univr.it/handle/11562/1116013.

22 Roberto Posenato, Marco Franceschetti, Carlo Combi, and Johann Eder. Introducing agile controllability in temporal business processes. In *Enterprise, Business-Process and Information Systems Modeling - 25th International Conference, BPMDS 2024, and 29th International Conference, EMMSAD 2024*, volume 511 of *Lecture Notes in Business Information Processing*, pages 87–99. Springer, 2024. doi:10.1007/978-3-031-61007-3_8.

23 Roberto Posenato, Andreas Lanz, Carlo Combi, and Manfred Reichert. Managing time-awareness in modularized processes. *Softw. Syst. Model.*, 18(2):1135–1154, 2019. doi:10.1007/S10270-017-0643-4.

24 Michael Saint-Guillain, Tiago Vaquero, Steve A. Chien, Jagriti Agrawal, and Jordan R. Abrahams. Probabilistic temporal networks with ordinary distributions: Theory, robustness and expected utility. *J. Artif. Intell. Res.*, 71:1091–1136, 2021. doi:10.1613/JAIR.1.13019.

25 Thierry Vidal and Hélène Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *J. Exp. Theor. Artif. Intell.*, 11(1), 1999. doi:10.1080/095281399146607.