SoftwareX 17 (2022) 100905

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

CSTNU Tool: A Java library for checking temporal networks

Roberto Posenato

Department of Computer Science, University of Verona, Verona, Italy

ARTICLE INFO

Article history:
Received 17 August 2021
Received in revised form 4 November 2021
Accepted 18 November 2021

Keywords:
Temporal constraint network
Consistency check
Dynamic controllability check
Constraint propagation algorithm

ABSTRACT

This paper presents *CSTNU Tool*, a Java library for representing and checking different kinds of temporal constraint networks. In particular, CSTNU Tool offers an optimized implementation of some constraint-propagation algorithms to check the dynamic consistency/controllability (DC) of *Conditional Simple Temporal Networks (CSTNs)*, *Conditional Simple Temporal Networks with Uncertainty (CSTNUs)*, and *Flexible Simple Temporal Networks with Uncertainty (FTNUs)*. The optimization is with respect to the management of labeled values that are present in conditional and flexible networks.

The library offers also a simple GUI application to build/manage and check temporal networks in an intuitive way, and some Java programs for generating random temporal networks according to some input parameters.

Code metadata

Current code version	4.3
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00153
Code Ocean compute capsule	https://codeocean.com/capsule/7405126/
Legal Code License	LGPL-3.0-or-later, CC0-1.0
Code versioning system used	SVN
Software code languages, tools, and services used	Java. Bash for utility scripts (not necessary)
Compilation requirements, operating environments & dependencies	Java $\geq$ 11
If available Link to developer documentation/manual	https://profs.scienze.univr.it/posenato/svn/sw/CSTNU/trunk/README.md
Project web site	https://profs.scienze.univr.it/~posenato/software/cstnu/
Support email for questions	roberto.posenato@univr.it

Software metadata

Current software version	4.3
Permanent link to executables of this version	https://profs.scienze.univr.it/~posenato/software/cstnu/bin/CstnuTool-4.3.tgz
Legal Software License	LGPL-3.0-or-later, CC0-1.0
Computing platforms/Operating Systems	Unix-like, Linux, OS X, Microsoft Windows
Installation requirements & dependencies	Java $\geq$ 11
If available, link to user manual	http://profs.scienze.univr.it/~posenato/software/cstnu
Support email for questions	roberto.posenato@univr.it

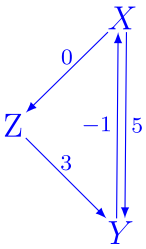
1. Motivation and significance

Constraint-based temporal reasoning has been widely used in some applications across heterogeneous domains [1]. Over the years, different proposals have been presented to address specific requirements that frequently arise in real-world applications.

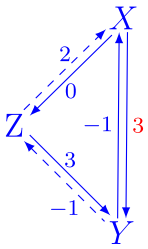
The most commonly used formalism is the *Simple Temporal Network (STN)* model, in which a set of real-valued variables, called *time-points*, are subject to binary difference constraints [2]. Time-points represent the occurrence of events while binary difference constraints represent the minimal/maximal allowed distance between pairs of events.

E-mail address: roberto.posenato@univr.it.
URL: https://www.di.univr.it/?ent=persona&id=102.

https://doi.org/10.1016/j.softx.2021.100905



(a) $\mathcal{S}$ graph



(b) $\mathcal{S}$ APSP graph. Dashed edges are added by the Floyd-Warshall algorithm. The red value is the new value for the original constraint $Y - X \leq 5$.

Fig. 1. $\mathcal{S}$ graph representations.

Example 1. Let $\mathcal{S}$ be an STN defined by time-points X, Y, and Z and the following temporal constraints

$$Y - X \leq 5$$
$$X - Y \leq -1$$
$$\mathrm{Z} - X \leq 0$$
$$Y - \mathrm{Z} \leq 3$$

Usually, Z is a special time-point that represents the first executed time-point and whose value is fixed at 0. In this way, expression like $\mathrm{Z} - X \leq 0$ and $Y - \mathrm{Z} \leq 3$ can be simplified as unary constraints $X \geq 0$ and $Y - \mathrm{Z} \leq 3$, respectively.

Therefore, $\mathcal{S}$ represents the fact that the temporal distance between Y and X must be in $[1, 5]$, the distance between Y and Z must be in $[0, 3]$, and the distance between X and Z must be non-negative.

The main problem regarding STN is the *consistency* problem, i.e., to check if an STN admits time-point assignments that satisfy all temporal constraints of the network.

An efficient way to represent an STN is to use a graph, called *STN/distance graph*, where vertices represent time-points and edges represent binary constraints, i.e., an edge $X \xrightarrow{\delta} Y$ represents the constraint $Y - X \leq \delta$ [2]. Given an STN, checking its consistency is equivalent to determine all-pairs-shortest-paths (APSP) matrix of its graph using an algorithm like Floyd–Warshall or Johnson [2]: the STN is consistent if and only if the main diagonal of ASPSP matrix is non-negative, i.e., there is no negative cycle in the graph. Moreover, since the APSP matrix contains the minimal distances between any pair of time-points, the corresponding graph represents the minimal STN graph equivalent to the original one.

Example 2. Fig. 1(a) shows the graph associated to the previous $\mathcal{S}$. Fig. 1(b) depicts the APSP graph obtained executing Floyd–Warshall algorithm. Dashed edges represent the implicit constraints obtained propagating the original constraints. As regards constraint $X \xrightarrow{5} Y$, the algorithm determines that the constraint must be restricted to value 3 because values in $(3, 5]$ are not admissible by other constraints.

Beyond STNs, a significant amount of research has focused on temporal reasoning in the presence of uncertainty. Temporal uncertainty arises, for example, when the duration of some activities can be only observed in real-time by the executor of the network. A *contingent link* represents an interval whose duration is bounded but uncontrollable, i.e., its value is decided by an external agent during execution; Usually, a contingent link between two time-points X and Y is represent as $(X, [l, u], Y)$, where l and u are real values and range $[l, u]$ represents the possible durations of the contingent link. X is set (or, equivalently, is executed) by the executor of the network while Y is only observed once the external agent executes it. Y is said *contingent* time-point. An STN containing contingent links is called a *Simple Temporal Network with Uncertainty (STNU)* [3]. In STNUs, the consistency problem is replaced by the *dynamically controllable (DC)* problem, i.e., to check if there exists a strategy for executing the non-contingent time-points such that all constraints are guaranteed to be satisfied no matter how the durations of the contingent links turn out during execution [4].

Example of possible applications of STNU are in the field of robot control [5], web-services compositions [6], and business processes [7,8].

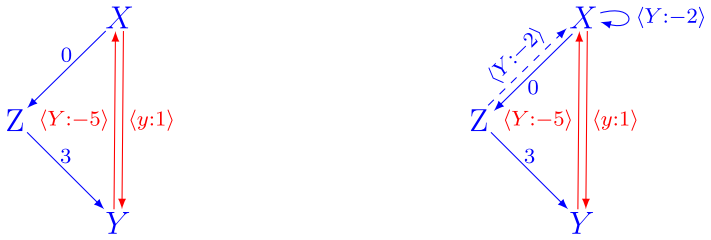
Example 3. Fig. 2(a) shows the graph associated to the STNU $\mathcal{S}'$ defined by time-points X, Y, and Z, the contingent link $(X, [1, 5], Y)$, and the following temporal constraints

$$\mathrm{Z} - X \leq 0$$
$$Y - \mathrm{Z} \leq 3.$$

The representation of a contingent link using edges was proposed by Morris and Muscettola [9]: the upper bound of the contingent range is represented as $Y \xrightarrow{\langle Y:-5\rangle} X$, called *upper-case (label) constraint*, while the lower bound is represented as $X \xrightarrow{\langle y:1\rangle} Y$, called *lower-case (label) constraint*. Upper-case constraints can be propagated assuming the meaning of *wait* constraints. For example, $D \xrightarrow{\langle Y:-10\rangle} X$ means that *"D must wait the occurrence of Y* ***or*** *at least* 10 *units after the execution of X for being executed"*. Morris and Muscettola determined the first polynomial-time DC checking algorithm for STNU [9]. It can be viewed as an extension of the Floyd–Warshall algorithm in which also upper/lower case constraints are considered/propagated. Then, many other propagation-based algorithms have been proposed to make the DC check even faster [9–16].

The above $\mathcal{S}'$ is not dynamic controllable. Indeed, contingent link $(X, [1, 5], Y)$ guarantees that Y can occur 5 units after the occurrence of X while constraint $Y \leq 3$ requires that Y must occur within 3 units after Z. Since X cannot be executed before Z, it is not possible to guarantee that Y can occur at any admissible time $(3, 5]$ after X without violating the constraint $Y \leq 3$. Fig. 2(b) shows the graph obtained by the execution of Morris and Muscettola DC checking algorithm [9]. The wait $X \xrightarrow{\langle Y,-2\rangle} X$ is equivalent to a negative loop in STNs, i.e., $\mathcal{S}'$ is not dynamic controllable.

Although STNUs have been successful in some domains, other domains require also constraints with condition. For example, in the health-care domain, medical tests on a patient frequently generate information in real-time that can affect which pathway that patient will follow [17]. Different pathways have different set



(a) $\mathcal{S}'$ graph where the contingent link is represented by an upper-case constraint (red) and a lower-case one (red).

(b) Negative loop in $\mathcal{S}'$ graph determined by Morris-Muscettola DC checking algorirthm [9].

Fig. 2. $\mathcal{S}'$ graph and its derived graph.

of constraints. It has to be guaranteed that any possible execution of the temporal network strictly satisfies all specified temporal constraints no matter which test outcomes are observed. The *Conditional Simple Temporal Network (CSTN)* model allows the representation of temporal constraints in conjunction with scenarios [18].

In detail, in a CSTN there can be *boolean observations* $p, q, r, s, \ldots$, each of them is associated to a proper time-point, called *observation time-point*. During an execution, when an observation time-point is executed, e.g., $P?$, then the associated boolean observation, e.g., p, is set to $\top/\bot$ by the environment.

A conjunction of some literals of the boolean observations is called *(propositional) label*. A label is $\top$ if and only if all its literal are $\top$ after that all observations have been set. A complete assignment of the observations of a CSTN is a *scenario*.

For example, label $p\neg q$ is true if and only if after the execution of the corresponding observation time-points, $p = \top$ and $q = \bot$.

Each constraint in a CSTN may be associated to a label that represents the *condition* for considering the constraint. For example, $X \xrightarrow{\langle \delta, p\neg q\rangle} Y$ represents the constraint $Y - X \leq \delta$ that must be satisfied in all *scenarios* in which $p\neg q$ is $\top$.

It is possible to have more conditional constraints between the same pair of time-points, each of them having a different label. Therefore, a CSTN graph can be a multi-graph.

As shown in [19], it is more efficient to represent the possible different conditional constraints between a pair of time-points as one constraint with a set of labeled values because it is possible to maintain minimal the size of such a set by applying a set of minimization rules derived from Assumption-based Truth Maintenance Systems (ATMS) [19]. For example, if in a CSTN network there are the boolean observation p and q, the edge $X \xrightarrow{\langle 10, p\neg q\rangle, \langle 4, q\rangle} Y$ represents two constraints:

1. $Y - X \leq 10$ that must be satisfied in scenario where $p\neg q$ is $\top$, and
2. $Y - X \leq 4$ that must be satisfied in scenarios where q is $\top$.

Adding the labeled value $\langle 4, \neg q\rangle$ to the edge, it results that the constraint ≤ 4 must be considered in all scenarios ($q \vee \neg q \equiv \top \equiv \boxdot$), and it is the most restrictive in the set. Therefore, the edge can be simplified as $X \xrightarrow{\langle 4, \boxdot\rangle} Y$.

A CSTN is said to be *dynamically consistent (DC)* if it admits a strategy that, reacting to the past observation, guarantees the satisfaction of all relevant constraints no matter which outcomes are observed during execution.

According to the delay by which an executor can acquire the value of observations, there are different kinds of dynamic consistency. For example, the Instantaneous Reaction (IR) assumption says that an executor can react instantaneously when an observation is set executing a time-point at the same instant if it is necessary; the ε-DC says that an executor can react after a delay $\varepsilon > 0$. In [20] there is a survey of the different kinds of dynamic consistency and their relations.

In the literature, there are different DC checking algorithms for CSTNs based on the conditional constraint propagation [18, 20–25].

Example 4. Fig. 3(a) shows the graph associated to the CSTN $\mathcal{S}''$ defined by time-points $P?$, Y, and Z, with $P?$ observation time-point of p, and the following conditional temporal constraints

$$Y - P? \leq 5, \neg p$$
$$P? - Y \leq -6, p$$
$$Z - P? \leq 0, \boxdot$$
$$Y - Z \leq 3, \neg p$$

Fig. 3(b) depicts the graph after the execution of the Hunsberger–Posenato DC checking algorithm [20]. $\mathcal{S}''$ is DC. The cycle between $P?$ and Y is not a negative cycle because the two constraints are not in the same scenarios. The algorithm determines the minimal constraints for an early-execution strategy. Considering the edges from time-points to Z, it is possible to determine the dynamic scheduling: Z and $P?$ must be executed at 0; at the execution of $P?$, p is set; then, Y must be executed at 0 if $p = \bot$, at 6 otherwise.

The *Conditional Simple Temporal Network with Uncertainty (CSTNU)* model extends the CSTN one allowing the representation of contingent links [26–30]. Fig. 5 depicts an interesting CSTNU instance where there are 3 contingent links and some conditional constraints that determine when activate such contingent link according to the p value.

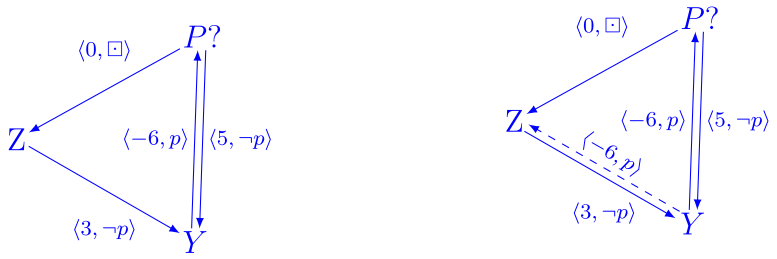
The *Flexible Simple Temporal Networks with Uncertainty (FTNU)* model, also known as Conditional Simple Temporal Network with Partially Shrinkable Uncertainty (CSTNPSU), extends the CSTNU model allowing each contingent link to be partially shrunk at design/execution time [31–33]. The idea is to allow the representation of an ideal duration range of a task that can be partially modified before the start of the task if this permits to satisfy network constraints.

In detail, in FTNUs contingent links are represented as *guarded links*. A guarded link $(X, \big[[l, l'][u', u]\big], Y)$, with $0 < l \leq l' \leq u' \leq u < \infty$, is a "contingent link" $(X, [l, u], Y)$ that it can be shrunk before its activation, i.e., before X is executed. The value l can be raised to l' at the most, while u can be decreased to u' at the most. Let l^*, u^* the values of l, u, respectively, when X is executed. Then, the guarded link becomes a contingent one with range $[l^*, u^*]$, and it is activated.

Despite the variety of applications of such models, implementations of methods for solving the consistency/controllability problem cannot be found in publicly available libraries.

The CSTNU Tool library[1] focuses on the solution of the consistency/controllability problem for the above-mentioned models

[1] The reason of CSTNU in the name is that CSTNU model was the only one considered in the initial releases.



(a) $\mathcal{S}''$ graph where $P?$ is the observation time-point of p.

(b) The $\mathcal{S}''$ graph determined by Hunsberger-Posenato DC checking algorirthm assuming IR [20].

Fig. 3. $\mathcal{S}''$ graph and its derived graph.

offering the implementation of different DC checking algorithms for each model. Users can interact with CSTNU Tool in two forms: using classes/methods from the Java package, aimed for advance users; using the integrated graphical editor to create/modify, and check a temporal constraint network easily, aimed for beginners.

2. CSTNU Tool: Software description

2.1. Software architecture

The CSTNU Tool is a Java library that runs in any JVM $\geq$ 11. The library is composed by two main packages: `it.univr.di.labeledvalue` and `it.univr.di.cstnu`.

Package `it.univr.di.labeledvalue` contains classes for representing and managing set of labeled values.

Package `it.univr.di.cstnu` contains classes for representing and managing some kinds of temporal constraint networks. It is divided in three sub-packages: `graph`, `algorithms`, and `visualization`. The `graph` sub-package contains classes for managing the graph of a temporal constraint network where nodes/edges can contain sets of labeled values. The `algorithms` sub-package is the core of the library; it contains a class for each kind of considered temporal constraint network that allows representing and checking of temporal constraint network instances. Moreover, it contains two Java programs for generating random CSTNs/STNUs of different types and sizes. The `visualization` sub-package contains the GUI application `TNEditor` to create/modify/visualize and check any kind of the networks mentioned above. The GUI of `TNEditor` is based on JUNG library [34].

2.2. Software functionalities

Each instance of a temporal constraint network is represented as an object of the corresponding class in `it.univr.di.cstnu.algorithms`. Such an object contains the graph as an object of the class `it.univr.di.cstnu .graph.TNGraph` and some information about how to check the consistency/controllability. The check of the consistency/controllability property is done by the method `checkConsistency()`/`checkControllability()`. Some classes offer different implementations of consistency/controllability checks. A specific implementation can be selected by a method before calling the check method. To ease the selection of the appropriate combination of checking algorithm, the kind of DC (e.g., *IR*, ε, *etc.*), and some other settings, there are some derived classes where such selection is pre-defined. The consistency/controllability check method always returns an object that describes the final status of the check, some statistics about the execution, and, in case of an inconsistent network, node(s) where an inconsistency was found.

Fig. 4 shows an excerpt of the UML class diagram of the `it.univr.di.cstnu` package where there are the classes for representing: (1) all kinds of possible temporal constraint networks (framed in blue), (2) result of a consistency/controllability check (framed in green), and (3) loading/saving the graph (framed in orange). Classes framed in violet are the CSTN sub-classes where some settings about the checking algorithm are already fixed.

Table A.1 in Appendix A summarizes the classes and their possible checking methods.

Package `it.univr.di.cstnu.labeledvalue` allows the representation and management of sets of labeled values in their minimal cardinality guaranteeing that each operation on a set has a linear-order computational cost.

3. Illustrative examples

We now provide an illustrative example of how to load and check the CSTNU instance shown in Fig. 5.

CSTNU Tool can load/save temporal constraint networks in GraphML format [35]. (In Appendix B, we present how temporal constraint networks are coded in GraphML in CSTNU Tool library.) Therefore, let us assume that the considered CSTNU instance is stored as a GraphML document in the file named "cstnuSoftwareX.cstnu".

The following simple Java program, using the CSTNU Tool library, loads the instance, executes the checks, and prints the result in the console.

```
1  public static void main(String[] args) {
2     File graphSource = new
      File("cstnuSoftwareX.cstnu");
3     TNGraphMLReader<CSTNUEdge> loader = new
      TNGraphMLReader<>();
4     TNGraph<CSTNUEdge> graph = null;
5     System.out.print("Loading the network...");
6     try {
7        graph = loader.readGraph(graphSource,
      EdgeSupplier.DEFAULT_CSTNU_EDGE_CLASS);
8     } catch (IOException |
      ParserConfigurationException | SAXException e) {
9        System.err.println("Format error in the
      instance source file: " + e.getMessage());
10       System.exit(1);
11    }
12    System.out.print("done.\nChecking its dynamic
      controllability...");
13    CSTNU cstnu = new CSTNU(graph);
14    CSTNUCheckStatus checkStatus = null;
15    try {
16       checkStatus =
      cstnu.dynamicControllabilityCheck();
17    } catch (WellDefinitionException e) {
18       System.err.println("The cstnu instance is not
      well defined: " + e.getMessage());
19       System.exit(1);
20    }
21    System.out.print("done\n" + checkStatus);
22 }
```

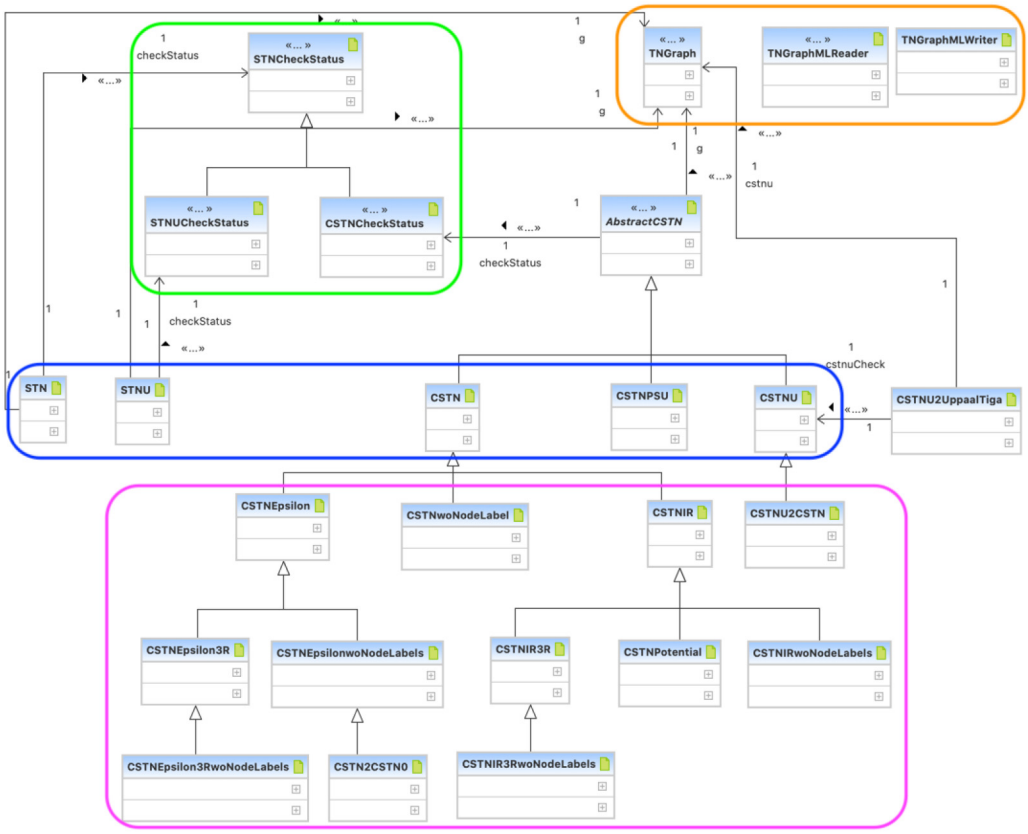


Fig. 4. Excerpt of the UML Class Diagram of the CSTNU Tool library.

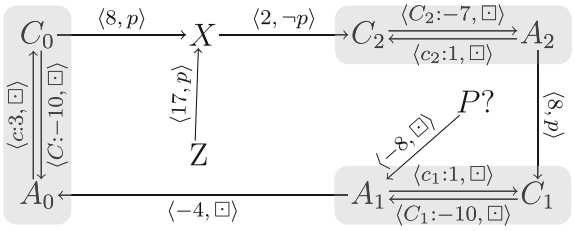


Fig. 5. A CSTNU instance. Constraints in gray regions represent contingent links.

To create an object of the CSTNU class, it is necessary to give the graph representing the network as a constructor parameter. The graph (object of `TNGraph<CSTNUEdge>`) can be loaded from a file using the helper class `TNGraphMLReader` (Line 3 and Line 7). Once the CSTNU object is created (Line 13), it is sufficient to call the method `checkControllability()` to verify the controllability of the instance (Line 16).

The check returns a `CSTNUCheckStatus` object containing the controllability status of the instance and some other statistics. All the information can be obtained in text form (Line 21). The output of an execution is:

```
Loading the network...done.
Checking its dynamic controllability...done
The check is  finished after 3 cycle(s).
The controllability check has determined that given
    network is dynamic controllable.
Some statistics:
Rule R0 has been applied 0 times.
Rule R3 has been applied 9 times.
Rule Labeled Propagation has been applied 139 times.
Rule Labeled z! has been applied 30 times.
Rule Labeled Lower Case has been applied 1 times.
Rule Labeled Cross-Lower-Case has been applied 1
    times.
Rule Labeled Letter Removal has been applied 12
    times.
The global execution time has been 78681000 ns
    (~0.07 s)
```

The instance results to be controllable. The implemented checking algorithm for CSTNU determines also all constraints that must be considered for a correct execution of the instance and adds them to the graph.

Using the program `TNEditor`, the checking task is even simpler. Fig. 6 depicts the screenshot of the program after the check of the sample instance has been completed. In general, on the left part of the program window, there is the editor-window where it is possible to load/create/edit a network instance; on the right part, there is the window where the program shows the instance after the execution of one of the different possible manipulating/checking algorithms. The selection of which algorithm to apply is done by the user pushing a button on the command bar present on the lower part of the program window. In the screenshot, the right part shows the completed graph obtained by applying the CSTNU DC-checking algorithm to the sample CSTNU instance, represented on the left part.

4. Impact and conclusions

In this work we described CSTNU Tool, a Java library for analyzing temporal constraint networks. CSTNU Tool offers an efficient implementation of dynamically-controllable/consistent checking algorithms for temporal constraint network models like CSTN, CSTNU, and FTNU.

As regards the impact of the library, it can serve for different purposes to researchers, and in application settings:

- **Applications.** In different fields there is a significant interest in considering temporal aspects for planning, scheduling,

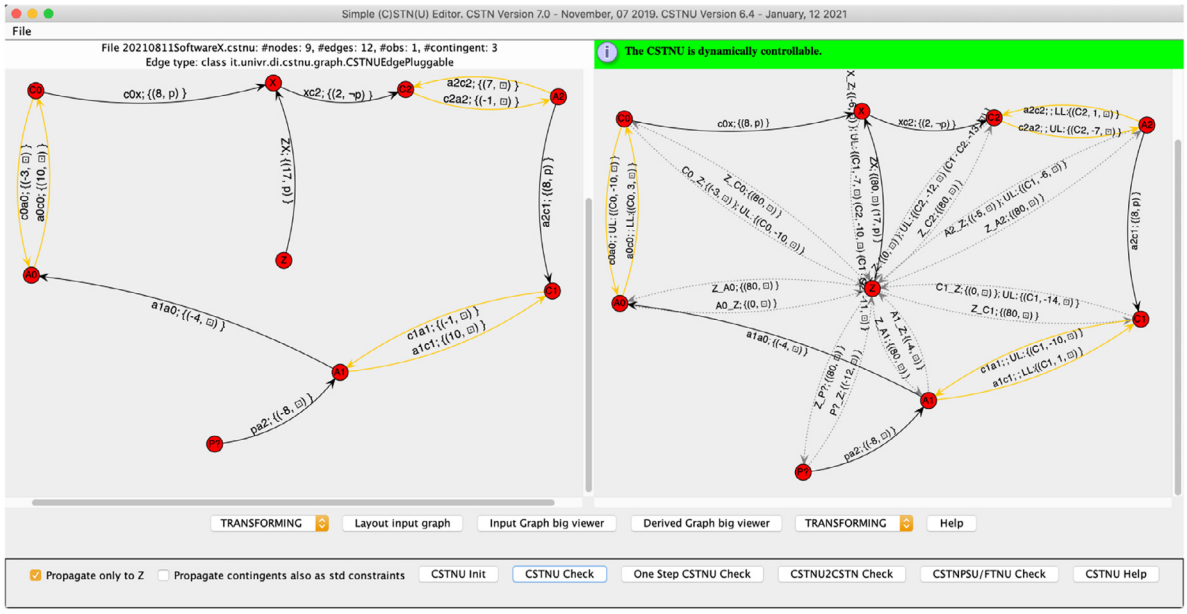


Fig. 6. `TNEditor` screenshot after the controllability check of the sample instance has been finished.

etc. and making temporal reasoning using temporal constraint networks for their simplicity and "efficiency" with respect other models. For example, in the field of planning/scheduling for robots [19,36–42], and in the field of business processes [32,43–47]. A critical aspect in such systems is the lack of efficient libraries for executing temporal reasoning efficiently. This library would contribute in building efficient solutions for such systems offering efficient algorithms for some temporal constraint network models. The tool has been already considered in [48–50].

- **Introduction to temporal constraint networks.** The GUI `TNEditor` may help researchers in the study and simulation of temporal constraint networks. For example, the intertwining of conditional constraints and contingent links is not as simple as it seems. `TNEditor` helps to understand such an intertwining, possible inconsistencies, and how it is possible restore the controllability modifying constraints in a fast way.
- **Test of new checking algorithms.** In the field of temporal reasoning, temporal constraint networks like STN/STNU represent an interesting model because they admit solving some problems efficiently. Although for richer models like CSTN such an efficiency is not possible [51], some researches have shown that some CSTN DC checking algorithms are practically efficient in many benchmarks [20,25]. The algorithms in CSTNU Tool library can be used either as a core for more advanced algorithms or as reference for different new algorithms.
  Page http://profs.scienze.univr.it/~posenato/software/cstnu/benchmarkWrapper.html presents the performances of the implemented algorithms in some benchmarks that are freely available.

In future work we plan to integrate CSTNU Tool library with implementations of *feasibility* and *dynamic scheduling* algorithms. Feasibility algorithms solve the problem of finding all inconsistencies. Dynamic scheduling algorithms allow the execution of a network in real-time.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

I would like to thank Francesca Zerbato for comments on the manuscript.

Funding

This work was partially supported by the INdAM–GNCS Project 2020 "Automated Reasoning about Time in Medical and Business Applications".

Appendix A. Classes and possible checking methods

Table A.1 summarizes the classes and their possible checking methods. The full list of the methods and corresponding method parameters is presented in http://profs.scienze.univr.it/~posenato/software/cstnu/apidocs.

Appendix B. Representation of temporal constraint network in graphml format

GraphML is an XML application for representing graphs of different types [35] in a very flexible way. In consists of two parts: a language core to describe the structural properties of a graph, and a flexible extension mechanism to add application-specific data.

In the language core, there are the elements `graph`, `node`, and `edge` by which it is possible to describe the topology of a graph.

The *GraphML-Attributes* extension allows the definition of node/edge attributes in the same XML document where the graph is defined.

In the CSTNU Tool library, it is assumed that a GraphML document describing a temporal constraint network uses the following attributes (we report here the most significant):

Table A.1
Summary of implemented checking algorithms.

Class	Checking algorithm
STN	Floyd–Warshall Bellman–Ford (single sink) Bannister–Eppstein Dijkstra Johnson Yen (single sink) BFCT
STNU	Morris14 RUL$^-$ RUL20
CSTN	HPC15 HP16 (for ϵ-reaction) HP18 (3 rules) HP20 (potential)
CSTNU	HP18 HP18 OnlyToZ CSTNU2CSTN
CSTNPSU (FTNU)	PC20

For the CSTN class, HP18 can be configured to work assuming different kind of reaction of the system with respect to the uncertainty of conditions, i.e., standard-/ ϵ-/ π- semantics.

```
<key id="Obs" for="node">
  <desc>Proposition Observed. Value specification:
    [a-zA-F]</desc>
</key>
<key id="Label" for="node">
  <desc>Label. Format: [¬[a-zA-F]|[a-zA-F]]+|⊡</desc>
  <default>⊡</default>
</key>
<key id="Potential" for="node">
  <desc>Labeled Potential Values. Format: {[('node name',
    'integer', 'label') ]+}|{}</desc>
</key>
<key id="Type" for="edge">
  <desc>Type: Possible values:
    contingent|requirement|derived|internal.</desc>
  <default>normal</default>
</key>
<key id="LowerCaseLabeledValues" for="edge">
  <desc>Labeled Lower-Case Values. Format: {[('node name',
    'integer', 'label') ]+}|{}</desc>
</key>
<key id="UpperCaseLabeledValues" for="edge">
  <desc>Labeled Upper-Case Values. Format: {[('node name',
    'integer', 'label') ]+}|{}</desc>
</key>
<key id="Value" for="edge">
  <desc>Value for STN edge. Format: 'integer'</desc>
</key>
<key id="LabeledValues" for="edge">
  <desc>Labeled Values. Format: {[('integer', 'label')
    ]+}|{}</desc>
</key>
```

Therefore, for example, to assign the labeled values $\langle 8, p\rangle$ and $\langle 6, q\rangle$ to the edge from node $C0$ to node X, it is sufficient to add the attribute "LabeledValues" as shown in the following listing:

```
<edge id="c0x" source="C0" target="X">
  <data key="LabeledValues">{(8, p), (6, q) }</data>
</edge>
```

References

[1] Barták R, Morris RA, Venable KB. An introduction to constraint-based temporal reasoning. Synthesis lectures on artificial intelligence and machine learning, vol. 8, Morgan & Claypool Publishers; 2014, p. 1–121. http://dx.doi.org/10.2200/S00557ED1V01Y201312AIM026.

[2] Dechter R, Meiri I, Pearl J. Temporal constraint networks. Artificial Intelligence 1991;49(1–3):61–95. http://dx.doi.org/10.1016/0004-3702(91)90006-6.

[3] Vidal T, Fargier H. Handling contingency in temporal constraint networks: from consistency to controllabilities. J Exp Theor Artif Intell 1999;11(1):23–45. http://dx.doi.org/10.1080/095281399146607.

[4] Morris PH, Muscettola N, Vidal T. Dynamic control of plans with temporal uncertainty. In: 17th international joint conference on artificial intelligence. 2001. p. 494–502.

[5] Karpas E, Levine SJ, Yu P, Williams BC. Robust execution of plans for human-robot teams. In: 25th international conference on automated planning and scheduling. 2015. p. 342–346.

[6] Franceschetti M, Eder J. Checking temporal service level agreements for web service compositions with temporal parameters. In: 2019 IEEE international conference on web services. IEEE; 2019, p. 443–5.

[7] Franceschetti M, Eder J. Designing decentralized business processes with temporal constraints. In: Advanced information systems engineering. 2020, p. 51–63. http://dx.doi.org/10.1007/978-3-030-58135-0_5.

[8] Franceschetti M, Eder J. Negotiating temporal commitments in cross-organizational business processes. In: 27th international symposium on temporal representation and reasoning. LIPIcs, vol. 178, Dagstuhl; 2020, p. 4:1–4:15. http://dx.doi.org/10.4230/LIPIcs.TIME.2020.4.

[9] Morris PH, Muscettola N. Temporal dynamic controllability revisited. In: 20th national conference on artificial intelligence. 2005. p. 1193–1198.

[10] Morris P. A structural characterization of temporal dynamic controllability. In: Principles and practice of constraint programming, Vol. 4204. 2006, p. 375–89. http://dx.doi.org/10.1007/11889205_28.

[11] Hunsberger L. A faster execution algorithm for dynamically controllable STNUs. In: 20th internationl symposium on temporal representation and reasoning. 2013.

[12] Hunsberger L. A faster algorithm for checking the dynamic controllability of simple temporal networks with uncertainty. In: 6th international conference on agents and artificial intelligence. 2014.

[13] Morris P. Dynamic controllability and dispatchability relationships. In: Integration of AI and OR techniques in constraint programming. CPAIOR 2014.. LNCS, vol. 8451, Springer; 2014, p. 464–79. http://dx.doi.org/10.1007/978-3-319-07046-9_33.

[14] Hunsberger L. New techniques for checking dynamic controllability of simple temporal networks with uncertainty. In: 6th international conference on agents and artificial intelligence, revised selected papers. Lecture notes in computer science, vol. 8946, 2015, p. 170–93.

[15] Morris P. The mathematics of dispatchability revisited. In: 26th int. conf. on automated planning and scheduling. 2016. p. 244–252.

[16] Cairo M, Rizzi R. Dynamic controllability made simple. In: 24th international symposium on temporal representation and reasoning. LIPIcs, vol. 90, 2017, p. 8:1–8:16. http://dx.doi.org/10.4230/LIPIcs.TIME.2017.8.

[17] Combi C, Gambini M, Migliorini S, Posenato R. 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 2014;44(9):1182–203. http://dx.doi.org/10.1109/TSMC.2014.2300055.

[18] Hunsberger L, Posenato R, Combi C. A sound-and-complete propagation-based algorithm for checking the dynamic consistency of conditional simple temporal networks. In: 22nd int. symp. on temporal representation and reasoning. 2015, p. 4–18. http://dx.doi.org/10.1109/TIME.2015.26.

[19] Conrad PR, Williams BC. Drake: An efficient executive for temporal plans with choice. J Artif Intell Res (JAIR) 2011;42:607–59, URL http://dx.doi.org/10.1613/jair.3478.

[20] Hunsberger L, Posenato R. Simpler and faster algorithm for checking the dynamic consistency of conditional simple temporal networks. In: 26th int. joint conf. on artificial intelligence. 2018, p. 1324–30. http://dx.doi.org/10.24963/ijcai.2018/184.

[21] Hunsberger L, Posenato R. Checking the dynamic consistency of conditional temporal networks with bounded reaction times. In: 26th int. conf. on automated planning and scheduling. 2016, p. 175–83, URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13108.

[22] Cairo M, Hunsberger L, Posenato R, Rizzi R. A streamlined model of conditional simple temporal networks - semantics and equivalence results. In: 24th int. symp. on temporal representation and reasoning. LIPIcs, vol. 90, 2017, p. 10:1–10:19. http://dx.doi.org/10.4230/LIPIcs.TIME.2017.10.

[23] Cairo M, Combi C, Comin C, Hunsberger L, Posenato R, Rizzi R, et al. Incorporating decision nodes into conditional simple temporal networks. In: 24th int. symp. on temporal representation and reasoning. LIPIcs, vol. 90, 2017, p. 9:1–9:18. http://dx.doi.org/10.4230/LIPIcs.TIME.2017.9.

[24] Hunsberger L, Posenato R. Reducing ϵ-DC checking for conditional simple temporal networks to DC checking. In: 25th int. symp. on temporal representation and reasoning. LIPIcs, vol. 120, 2018, p. 15:1–15:15. http://dx.doi.org/10.4230/LIPIcs.TIME.2018.15.

[25] Hunsberger L, Posenato R. Faster dynamic-consistency checking for conditional simple temporal networks. In: 30th int. conf. on automated planning and scheduling, Vol. 30. 2020, p. 152–60, URL https://www.aaai.org/ojs/index.php/ICAPS/article/view/6656.

[26] Combi C, Hunsberger L, Posenato R. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In: 5th int. conf. on agents and artificial intelligent, Vol. 2. 2013, p. 144–56. http://dx.doi.org/10.5220/0004256101440156.

[27] Combi C, Hunsberger L, Posenato R. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty - revisited. In: Agents and artificial intelligence. Communications in computer and information science (CCIS), vol. 449, Springer; 2014, p. 314–31. http://dx.doi.org/10.1007/978-3-662-44440-5_19.

[28] Cimatti A, Hunsberger L, Micheli A, Posenato R, Roveri M. Sound and complete algorithms for checking the dynamic controllability of temporal networks with uncertainty, disjunction and observation. In: 21st int. symp. on temporal representation and reasoning. 2014, p. 27–36. http://dx.doi.org/10.1109/TIME.2014.21.

[29] Cimatti A, Hunsberger L, Micheli A, Posenato R, Roveri M. Dynamic controllability via timed game automata. Acta Inform 2016;53(6–8):681–722. http://dx.doi.org/10.1007/s00236-016-0257-2.

[30] Hunsberger L, Posenato R. Sound-and-complete algorithms for checking the dynamic controllability of conditional simple temporal networks with uncertainty. In: 25th int. symp. on temporal representation and reasoning. LIPIcs, vol. 120, 2018, p. 14:1–14:17. http://dx.doi.org/10.4230/LIPIcs.TIME.2018.14.

[31] Combi C, Posenato R. Extending conditional simple temporal networks with partially shrinkable uncertainty. In: 25th international symposium on temporal representation and reasoning. LIPIcs, vol. 120, Dagstuhl; 2018, p. 9:1–9:15. http://dx.doi.org/10.4230/LIPIcs.TIME.2018.9.

[32] Posenato R, Lanz A, Combi C, Reichert M. Managing time-awareness in modularized processes. Soft Syst Model 2019;18(2):1135–54. http://dx.doi.org/10.1007/s10270-017-0643-4.

[33] Posenato R, Combi C. Adding flexibility to uncertainty: Flexible simple temporal networks with uncertainty (FTNU). Inform Sci 2022;584:784–807. http://dx.doi.org/10.1016/j.ins.2021.10.008.

[34] O'Madadhain Jea. JUNG: Java universal network/graph framework. 2016, https://github.com/jrtom/jung.

[35] Brandes U, Eiglsperger M, Herman I, Himsolt M, Marshall MS. Graphml progress report structural layer proposal. In: Graph drawing. 2002, p. 501–12. http://dx.doi.org/10.1007/3-540-45848-4_59, URL http://graphml.graphdrawing.org/.

[36] Muscettola N, Nayak P, Pell B, Williams B. Remote agent: To boldly go where no AI system has gone before. Artificial Intelligence 1998;103(1–2):5–47. http://dx.doi.org/10.1016/s0004-3702(98)00068-x.

[37] Frank J, Jónsson A. Constraint-based attribute and interval planning. Constraints 2003;8(4):339–64. http://dx.doi.org/10.1023/A:1025842019552.

[38] Benton J, Coles A, Coles A. Temporal planning with preferences and time-dependent continuous costs. In: Proceedings of the twenty-second international conference on automated planning and scheduling. 2012, p. 2–10, URL https://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4699/4708.

[39] McGann C, Py F, Rajan K, Thomas H, Henthorn R, et al. A deliberative architecture for AUV control. In: Proc. - ieee int. conf. robot. autom.. 2008, p. 1049–54. http://dx.doi.org/10.1109/ROBOT.2008.4543343.

[40] Nunes E, Gini M. Multi-robot auctions for allocation of tasks with temporal constraints. In: Proc. natl. conf. artif. intell., Vol. 3. AI Access Foundation; 2015, p. 2110–6.

[41] Maniadakis M, Hourdakis E, Trahanias P. Time-informed task planning in multi-agent collaboration. Cogn Syst Res 2016. http://dx.doi.org/10.1016/j.cogsys.2016.09.004.

[42] Hofmann AG, Williams BC. Temporally and spatially flexible plan execution for dynamic hybrid systems. Artificial Intelligence 2017;247:266–94. http://dx.doi.org/10.1016/j.artint.2015.02.007, Application of STN.

[43] Eder J, Panagos E, Rabinovich M. Workflow time management revisited. In: Seminal contributions to information systems engineering. Springer; 2013, p. 207–13. http://dx.doi.org/10.1007/978-3-642-36926-1_16.

[44] Cheikhrouhou S, Kallel S, Guermouche N, Jmaiel M. A survey on time-aware business process modeling. Tech. rep, Centre pour la communication scientifique directe; 2013, URL http://hal.archives-ouvertes.fr/hal-00800444/.

[45] Lanz A, Weber B, Reichert M. Time patterns for process-aware information systems. Requir Eng 2014;19(2):113–41. http://dx.doi.org/10.1007/s00766-012-0162-3.

[46] Lanz A, Posenato R, Combi C, Reichert M. Controlling time-awareness in modularized processes. In: Enterprise, business-process and information systems modeling, 17th international conference, BPMDS 2016, 21st international conference, EMMSAD 2016. 2016, p. 157–72. http://dx.doi.org/10.1007/978-3-319-39429-9_11.

[47] Posenato R, Zerbato F, Combi C. Managing decision tasks and events in time-aware business process models. In: Business process management - 16th international conference. LNCS, vol. 11080, Springer; 2018, p. 102–18. http://dx.doi.org/10.1007/978-3-319-98648-7_7.

[48] Liu D, Wang H, Qi C, Zhao P, Wang J. Hierarchical task network-based emergency task planning with incomplete information, concurrency and uncertain duration. Knowled-Based Syst 2016;112:67–79. http://dx.doi.org/10.1016/J.KNOSYS.2016.08.029.

[49] Eder J, Franceschetti M, Köpke J. Controllability of business processes with temporal variables. In: Proceedings of the 34th ACM/SIGAPP symposium on applied computing. ACM; 2019, p. 40–7. http://dx.doi.org/10.1145/3297280.3297286.

[50] Ocampo-Pineda M, Posenato R, Zerbato F. TimeAwareBPMN-Js: an editor and temporal verification tool for time-aware BPMN processes. SoftwareX 2022;???:???–???. http://dx.doi.org/10.1016/j.softx.2021.100939.

[51] Cairo M, Rizzi R. Dynamic controllability of conditional simple temporal networks is PSPACE-complete. In: 23rd int. symp. on temporal representation and reasoning. 2016, p. 90–9. http://dx.doi.org/10.1109/TIME.2016.17.