An Algorithm for Finding Negative Cycles in Simple Temporal Networks with Uncertainty

Luke Hunsberger[1] **Roberto Posenato**[2]

[1]Department of Computer Science, Vassar College, Poughkeepsie, NY USA

[2]Department of Computer Science, University of Verona, Verona, Italy

TIME 2024
October 28–30, 2024
Montpellier, France

Motivation

Simple Temporal Networks with Uncertainty (STNUs)

- Simple Temporal Network with Uncertainty (STNU)
  a data structure for representing and reasoning about events and actions with uncertain, but bounded durations [11].
- STNUs [11] are attractive for planning and scheduling applications in different fields [4, 13, 5, 8, 16].
- Dynamic Controllability (DC)
  a property that ensures the existence of a dynamic strategy for executing the controllable events such that all relevant constraints will be satisfied no matter how the uncertain durations turn out [11].
- The current fastest algorithm for solving the DC–checking decision problem (RUL2021) runs in $O(mn + k^2 n + 2n \log n)$ time [7, 3].

---

n = #events, m = #constraints, k = #uncertain durations

Motivation (continued)

Finding Semi–Reducible Negative (SRN) Cycles in Non–DC STNUs

- What about non–DC STNUs?
- Non–DC STNUs necessarily have one or more semi–reducible negative (SRN) cycles [9].
  - SRN cycles can have exponentially many (repeated) edges [6].
- Finding SRN cycles helps to restore the DC property, but existing algorithms are based on less efficient DC–checking algorithms and give scant attention to repeated edges [18, 19, 17, 1, 2].
- This paper presents a faster algorithm for solving the search version of the DC problem. For non–DC networks, it returns:
  - one SRN cycle (in a compact form)
  - in time $O(mn + k^2 n + kn \log n)$
  - and space $O(mk + k^2 n)$.

Simple Temporal Networks with Uncertainty

Definition [11]

An STNU is a triple, $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$, where:

- $\mathcal{T} = \mathcal{T}_X \cup \mathcal{T}_C$ is a set of real–valued variables called *timepoints* (*instantaneous events*);
- $\mathcal{C}$ is a set of *ordinary constraints*, each of the form $Y - X \leq \delta$, where $X, Y \in \mathcal{T}$ and $\delta \in \mathbb{R}$
- $\mathcal{L}$ is a set of *contingent links* (*actions with uncertain durations*), each of the form (A, x, y, C):
  - $A \in \mathcal{T}_X$ is the *activation* timepoint.
  - $C \in \mathcal{T}_C$ is the *contingent* timepoint.
  - $C - A \in [x, y]$ but *uncontrollable*

Executing an STNU: Assign values to the *non–contingent* timepoints (i.e., those in $\mathcal{T}_X$) aiming to satisfy all constraints in $\mathcal{C}$.

- Contingent links: Agent executes A, but only observes the *execution* of C in real time, knowing that $C - A \in [x, y]$.

Simple Temporal Networks with Uncertainty

STNU Graph [12]

Each STNU has a graphical form [12] where:

- Timepoints $\iff$ nodes
- Ordinary constraints $\iff$ ordinary edges:

  $Y - X \in [3, 7] \iff X \underset{-3}{\overset{7}{\rightleftarrows}} Y$

- Contingent Links $\iff$ Labeled Edges

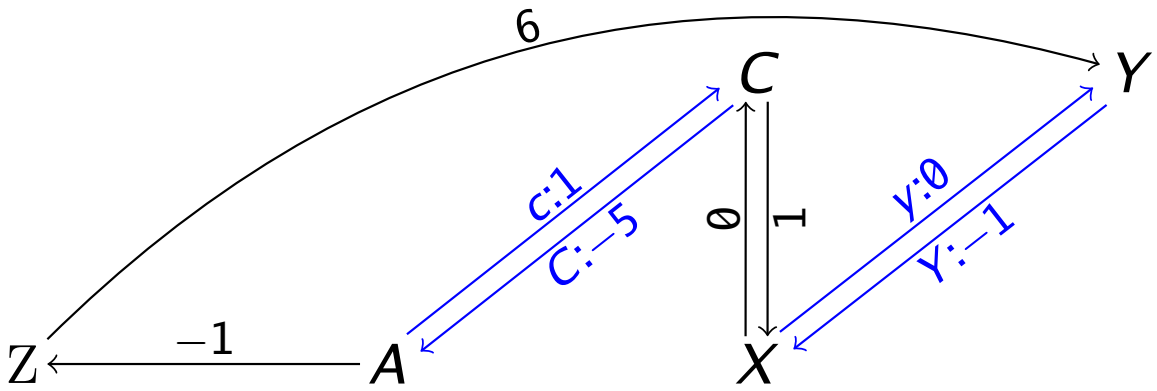
  $(A, 3, 7, C) \iff A \underset{C:-7}{\overset{c:3}{\rightleftarrows}} C$

  - The *lower–case edge*, $A \xrightarrow{c:3} C$, represents the *uncontrollable possibility* $C - A$ might equal 3 (minimum).
  - The *upper–case edge*, $A \xleftarrow{C:-7} C$, represents the *uncontrollable possibility* $C - A$ might equal 7 (maximum).

- In general, any path (or cycle) from X to Y is semi–reducible (SR) if it entails a path of the same length from X to Y that contains no lower–case edges.

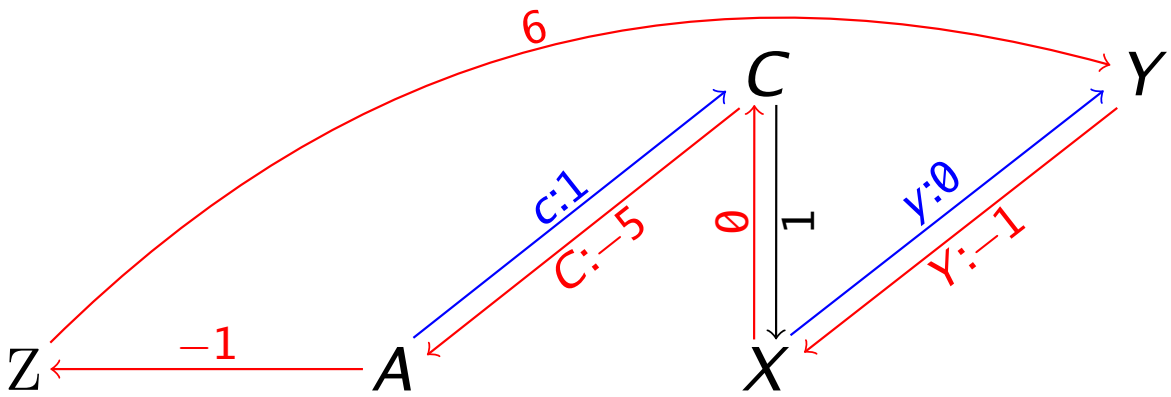
Simple Temporal Networks with Uncertainty

Sample STNU Graph



- Z is the blood sample extraction event: $\mathrm{Z} = 0$.
- Blood test must start at least one day after the extraction (i.e., $\mathrm{Z} - A \leq -1$; equivalently, $A \geq 1$).
- Blood test may one to five days: contingent link $(A, 1, 5, C)$.
- The test result must be reported to a doctor within 1 day: $X - C \in [0, 1]$.
- The validation/communication of test result takes up to one day: contingent link $(X, 0, 1, Y)$.
- The entire process must take at most 6 days: $Y - \mathrm{Z} \leq 6$ (i.e., $Y \leq 6$).

The Sample STNU is not DC!



It contains the (red) SRN cycle:

$(Z, 6, Y, Y{:}-1, X, 0, C, C{:}-5, A, -1, Z)$

The RUL2021 DC–Checking Algorithm [7]

Key Features

- Key idea: propagate backward from upper–case edges, along ordinary and lower–case edges, aiming to generate bypass edges.
- Back–propagation is guided by a priority queue and a potential function to re–weight the edges to non–negative values, as in Johnson's algorithm.
- The bypass edges are the only new edges inserted into the graph.
- Use a separate, Dijkstra–like back–propagation to update the potential function after the insertion of bypass edges.

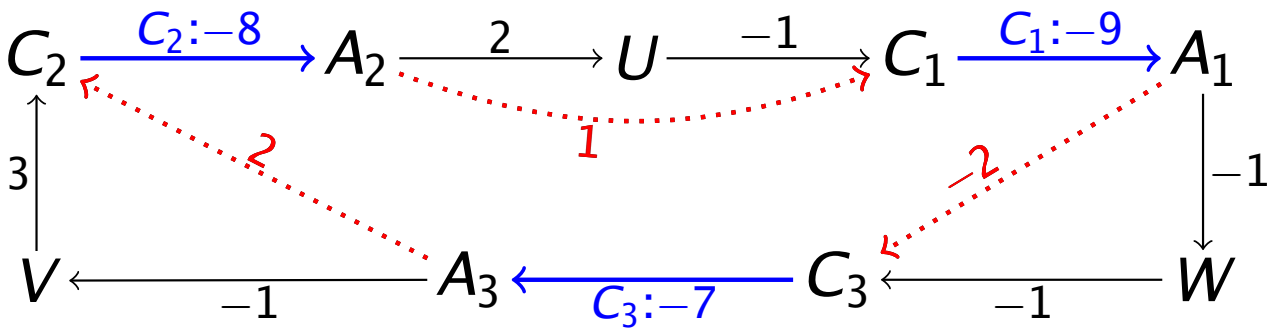
Inserting bypass edges is sufficient for DC checking, but not for finding SRN cycles (in the case of non–DC STNUs).

The RUL2021 DC–Checking algorithm

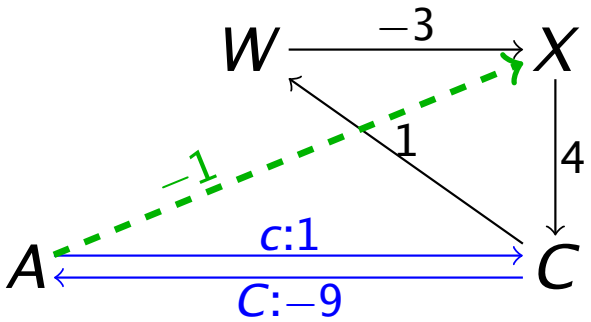
Key Observations

Three different ways that RUL2021 detects that an STNU is not DC:

1. Failure to update the potential function
2. Cycle of interruptions



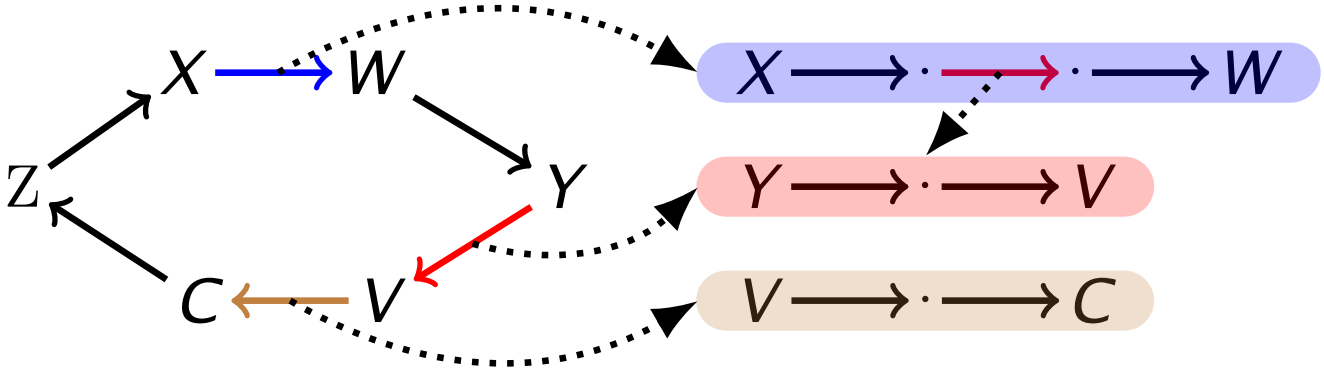
3. CC loops



The FindSRNC (Find SRN Cycle) Algorithm

Key features

- Preserves the general structure of the RUL2021 algorithm.
- If the input is a non–DC STNU, it outputs a compact representation of an SRN cycle `(negCycle, edgeAnnotation)`:
  - `negCycle`: a negative cycle of (labeled) edges;
  - `edgeAnnotation`: a hash table *(bypassEdge, path)* entries, where: *bypassEdge* is an edge generated by the algorithm, and *path* is the path used to generate that edge.



- This polynomial–sized representation is the most convenient since, in the worst case, unpacking all the edges in the SRN cycle could result in an exponential number of repeated edges.

The FindSRNC (Find SRN Cycle) Algorithm

Key features

- We enriched the RUL2021 algorithm to temporarily store paths during back propagation (in `loc.path`) and permanently store paths associated with bypass edges (in `edgeAnnotation`).
  - $\approx$ 25% new instructions sufficient for obtaining an SRN cycle in case of "Failure to update the potential function" or "CC loops".
  - A new procedure `AddNegCycle` builds the relevant SRN cycle in case of a "Cycle of interruptions".
- Correctness follows from the correctness of RUL2021:
  - Just added instructions to store the relevant paths in the appropriate hash tables.

The FindSRNC (Find SRN Cycle) Algorithm

Time and Space Complexity

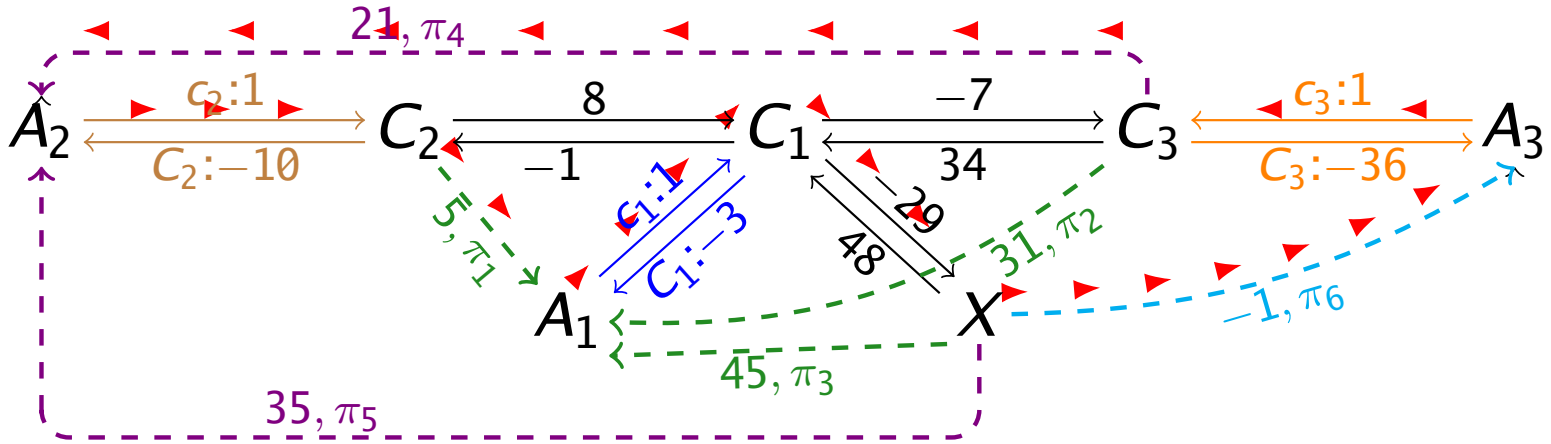
- **Most time–consuming operation**: Prepending an edge to an existing path. Achievable in constant time. (This occurs at most once per visited edge.)
- **Worst–case time complexity**: $O(mn + k^2 n + kn \log n)$ (= RUL2021)
- **Worst–case space complexity**: $O(mk + k^2 n)$
  - `TryBackProp` requires space for accumulating path information. It is called $\leq 2k$ times; each time it requires $O(m + nk)$ space. Total space across $O(k)$ iterations: $O(mk + k^2 n)$.
  - The `edgeAnnotation` hash table has $O(nk)$ entries (one for each bypass edge).
  - The `AddNegCycle` procedure generates a compact SRN cycle joining at most k paths. Each path can have at most n edges $\Rightarrow O(nk)$.

The pair `(negCycle, edgeAnnotation)` avoids redundantly storing repeated structures.

Magic Loops

- There exists a family of STNUs, each containing an indivisible SRN cycle, called a magic loop, that has an exponential number of edges [6].
- Since each STNU has at most $n^2 + 2k$ edges, magic loops necessarily contain many repeated edges!
- A magic loop of order k contains
  - k contingent links, and
  - $3(2^k) - 2$ edges.

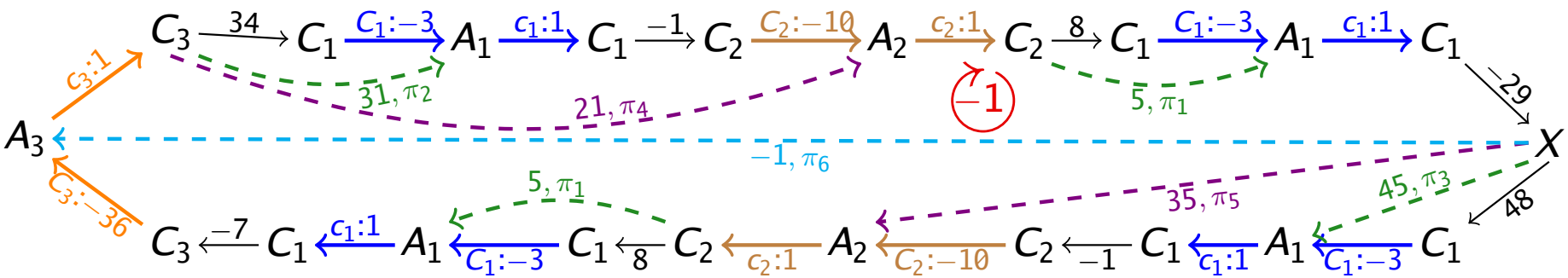
Sample Magic Loop of Order 3



- 3 contingent links; 6 constraints across contingent timepoints; and a non–contingent timepoint X that must be scheduled before C_1.
- Bypass edges are dashed.
- The SRN cycle is $(A_3, c_3{:}1, C_3, 21, A_2, c_2{:}1, C_2, 5, A_1, c_1{:}1, C_1, -29, X, -1, A_3)$, and its length (one traversal) is -1.
- It occurs when all contingent links assume their min value.
- But it is not the only thing...

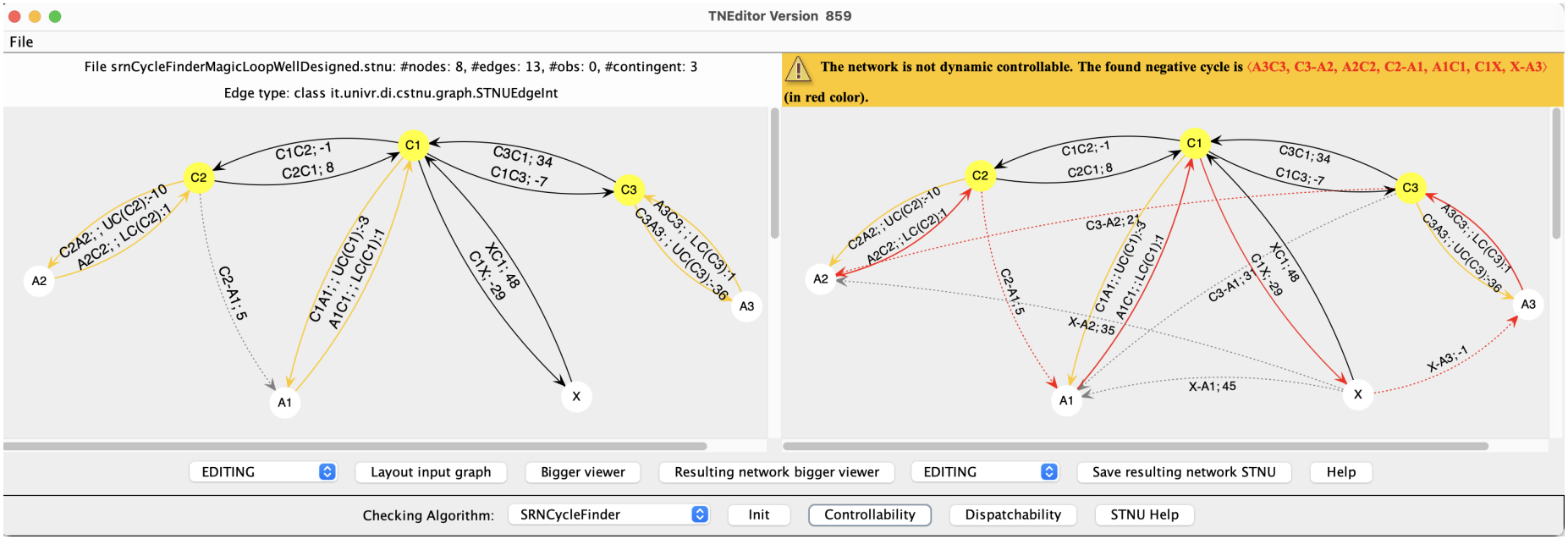
Sample Magic Loop of Order 3 (continued)

- Representing all bypass edges in terms of original edges, #edges = $3(2^3) - 2 = 22$, exponential w.r.t. #contingent links.
- Edges of contingent link $(A_1, 1, 3, C_1)$ are present 4 times each.



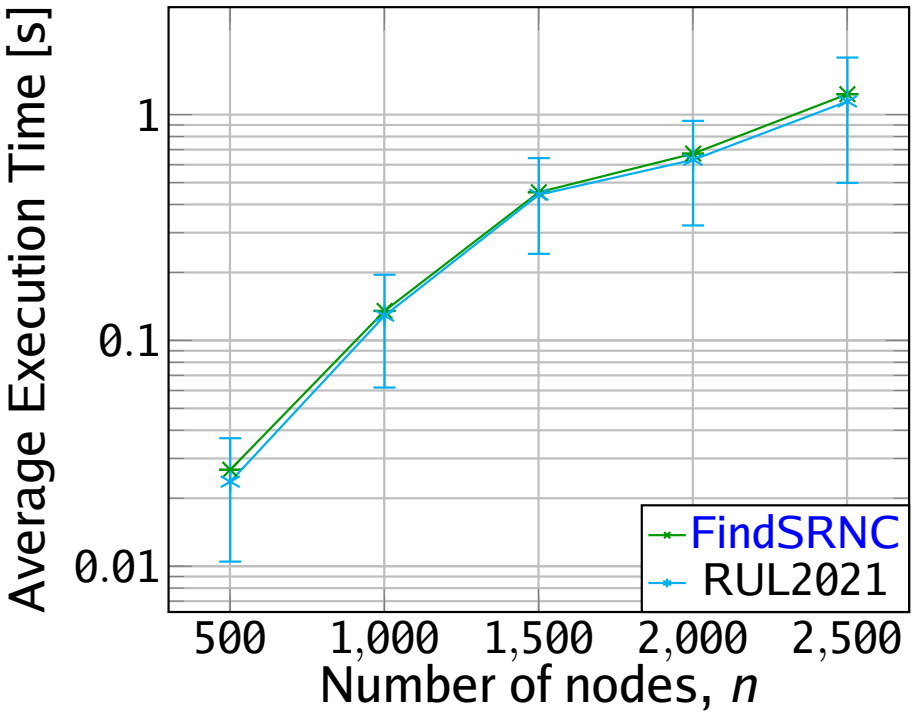
Proof of Concept

- The FindSRNC algorithm is implemented in the CSTNU Tool [15].
- The CSTNU Tool enables users to manage different kinds of temporal constraint networks and to verify automatically certain properties (e.g., dynamic controllability, dispatchability).
- For STNUs: it allows DC checking and, in case the network is not DC, the determination of an SRN cycle.



Proof of Concept

- We empirically evaluated FindSRNC on a published benchmark [14] to confirm that the execution times of FindSRNC and RUL2021 are equivalent.



n	Avg. SRN cycle length	% Simple SRN cycles	Avg. SRN expanded cycle length
500	8.0	63%	14.1
1000	7.9	64%	14.4
1500	7.6	74%	15.3
2000	8.5	72%	14.0
2500	8.6	63%	14.4

Conclusion

- We presented the FindSRNC algorithm which extends the fastest DC–checking algorithm for STNUs to determine and compactly represent an SRN cycle when the input is not DC.
- We showed that FindSRNC does not require more time than the fastest DC–checking algorithm to find the SRN cycle.
- We provided a Java implementation within the CSTNU Tool library.
- FindSRNC can be used to identify constraints to relax or contingent durations to tighten when seeking to restore the DC property.

References I

[1] Shyan Akmal, Savanna Ammons, Hemeng Li, and James C. Boerkoel, Jr.
Quantifying degrees of controllability in temporal networks with uncertainty.
In *29th International Conference on Automated Planning and Scheduling (ICAPS 2019)*, pages 22–30, 2019.
doi:10.1609/icaps.v29i1.3456.

[2] Nikhil Bhargava, Tiago Vaquero, and Brian C. Williams.
Faster conflict generation for dynamic controllability.
In *Proceedings of the Twenty–Sixth International Joint Conference on Artificial Intelligence (IJCAI–17)*, pages 4280–4286, 2017.
doi:10.24963/ijcai.2017/598.

References II

[3] Massimo Cairo, Luke Hunsberger, and Romeo Rizzi.
Faster Dynamic Controllablity 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.
doi:10.4230/LIPIcs.TIME.2018.8.

[4] Johann Eder, Marco Franceschetti, and Josef Lubas.
Dynamic Controllability of Processes without Surprises.
Applied Sciences, 12(3):1461, January 2022.
doi:10.3390/app12031461.

[5] Cheng Fang, Andrew J. Wang, and Brian C. Williams.
Chance–constrained Static Schedules for Temporally Probabilistic Plans.
J. Artificial Intelligence Research, 75:1323–1372, 2022.
doi:10.1613/jair.1.13636.

References III

[6] Luke Hunsberger.
Magic Loops in Simple Temporal Networks with Uncertainty–Exploiting Structure to Speed Up Dynamic Controllability Checking.
In *5th International Conference on Agents and Artificial Intelligence (ICAART–2013)*, volume 2, pages 157–170, 2013.
doi:10.5220/0004260501570170.

[7] Luke Hunsberger and Roberto Posenato.
Speeding up the RUL$^-$ Dynamic–Controllability–Checking Algorithm for Simple Temporal Networks with Uncertainty.
In *36th AAAI Conference on Artificial Intelligence (AAAI–22)*, volume 36–9, pages 9776–9785. AAAI Pres, 2022.
doi:10.1609/aaai.v36i9.21213.

[8] Erez Karpas, Steven J. Levine, Peng Yu, and Brian C. Williams.
Robust Execution of Plans for Human–Robot Teams.
In *25th Int. Conf. on Automated Planning and Scheduling (ICAPS–15)*, volume 25, pages 342–346, 2015.
doi:10.1609/icaps.v25i1.13698.

References IV

[9] Paul Morris.
A Structural Characterization of Temporal Dynamic Controllability.
In *Principles and Practice of Constraint Programming (CP–2006)*, volume 4204, pages 375–389, 2006.
doi:10.1007/11889205_28.

[10] Paul Morris.
Dynamic controllability and dispatchability relationships.
In *Int. Conf. on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR–2014)*, volume 8451 of *LNCS*, pages 464–479. Springer, 2014.
doi:10.1007/978-3-319-07046-9_33.

[11] Paul Morris, Nicola Muscettola, and Thierry Vidal.
Dynamic control of plans with temporal uncertainty.
In *17th Int. Joint Conf. on Artificial Intelligence (IJCAI–2001)*, volume 1, pages 494–499, 2001.
URL: https://www.ijcai.org/Proceedings/01/IJCAI-2001-e.pdf.

References V

[12] Paul H. Morris and Nicola Muscettola.
Temporal dynamic controllability revisited.
In *20th National Conference on Artificial Intelligence (AAAI–2005)*, pages 1193–1198, 2005.
URL: https://www.aaai.org/Papers/AAAI/2005/AAAI05-189.pdf.

[13] Jun Peng, Jingwei Zhu, and Liang Zhang.
Generalizing STNU to Model Non–functional Constraints for Business Processes.
In *2022 International Conference on Service Science (ICSS)*, pages 104–111. IEEE, May 2022.
doi:10.1109/ICSS55994.2022.00024.

[14] Roberto Posenato.
STNU Benchmark version 2020, 2020.
https://profs.scienze.univr.it/~posenato/software/cstnu/benchmarkWrapper.

References VI

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

[16] Christoph Strassmair and Nicholas Kenelm Taylor.
Human Robot Collaboration in Production Environments.
In *23rd IEEE International Symposium on Robot and Human Interactive Communication 2014*, 2014.
URL: https://researchportal.hw.ac.uk/en/publications/human-robot-collaboration-in-production-environments.

[17] Andrew J. Wang.
Risk–bounded Dynamic Scheduling of Temporal Plans.
PhD thesis, Massachusetts Institute of Technology, 2022.
URL: https://hdl.handle.net/1721.1/147542.

References VII

[18] Peng Yu, Cheng Fang, and Brian Charles Williams.
Resolving uncontrollable conditional temporal problems using continuous relaxations.
In *24th International Conference on Automated Planning and Scheduling, ICAPS 2014*. AAAI, 2014.
doi:10.1609/icaps.v24i1.13623.

[19] Peng Yu, Brian C. Williams, Cheng Fang, Jing Cui, and Patrick Haslum.
Resolving over–constrained temporal problems with uncertainty through conflict–directed relaxation.
J. Artificial Intelligence Research, 60:425–490, 2017.
doi:10.1613/jair.5431.

Simple Temporal Networks with Uncertainty

Dynamic Controllability [11]

An STNU is *dynamically controllable* (DC) if:

- there exists a *dynamic strategy* ...
- for executing the *non–contingent* timepoints ...
- such that *all* the constraints will be satisfied ...
- *no matter how the contingent durations turn out.*

On–line vs Off–line Scheduler

A dynamic (on–line) scheduler for a DC STNU must be able to react to any possible combination of contingent durations.

In contrast, an off–line scheduler can require exponential space to store all possible combinations of contingent durations.

Simple Temporal Networks with Uncertainty

DC–Checking Algorithms for STNUs

Algorithm	Complexity
Morris06	$O(n^4)$
Morris14	$O(n^3)$
RUL$^-$ [3]	$O(mn+k^2n+kn\log n)$
RUL2021 [7]	$O(mn+k^2n+kn\log n)$

– All algorithms are based on *propagation rules* that are used to make some implicit constraints explicit.

Rule	Graphical Representation	Conditions
(NC)	$X \xrightarrow{v} Y \xrightarrow{w} W \;\Rightarrow\; X \xrightarrow[v+w]{} W$	(none)
(UC)	$X \xrightarrow{v} Y \xrightarrow{C:w} A \;\Rightarrow\; X \xrightarrow[C:v+w]{} A$	(none)
(LC)	$A \xrightarrow{c:x} C \xrightarrow{w} X \;\Rightarrow\; A \xrightarrow[x+w]{} X$	$w < 0$
(CC)	$A \xrightarrow{c:x} C \xrightarrow{K:w} B \;\Rightarrow\; A \xrightarrow[K:x+w]{} B$	$K \not\equiv C$, $w < 0$
(LR)	$X \xrightarrow{C:w} A \xrightarrow{c:x} C \;\Rightarrow\; X \xrightarrow[w]{} A$	$w \geq -x$

Morris & Muscettola Rules [12, 10]

Rule	Graphical representation	Applicability Conditions
(R)	$P \xrightarrow{v} Q \xrightarrow{w} C \;\Rightarrow\; P \xrightarrow[v+w]{} C$	$Q \in \mathcal{T}_X$, $w < \Delta_C$, $C \in \mathcal{T}_C$
(L)	$A_j \xrightarrow{c':x'} C' \xrightarrow{w} C \;\Rightarrow\; A_j \xrightarrow[x_j+w]{} C$	$C' \not\equiv C$, $w < \Delta_C$, $C \in \mathcal{T}_C$
(U_{lp})	$P \xrightarrow{v} C \xrightarrow{C:-y} A \;\Rightarrow\; P \xrightarrow[v-y]{} A$	

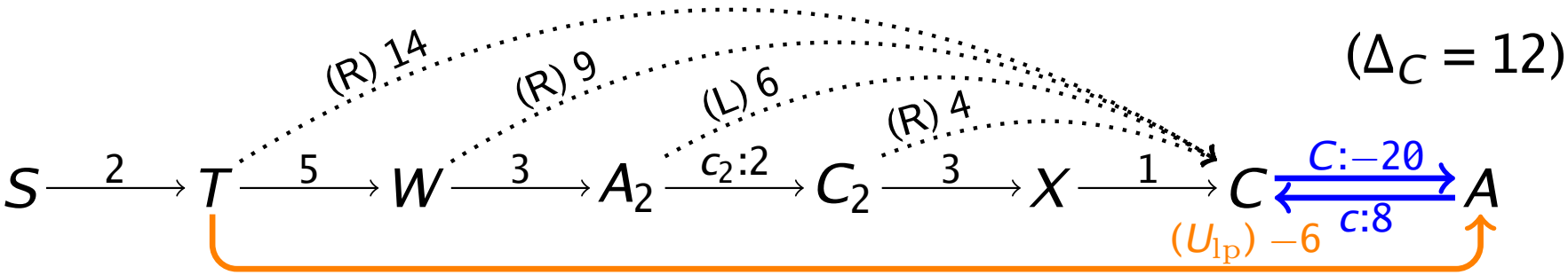
RUL2021 Rules [7]

Theorem 1 (Morris [9])

An STNU is Dynamically Controllable if and only if it does not have a semi–reducible negative (SRN) cycle.

The FindSRNC algorithm

Example of a bypass edge and determination of its generation path



`loc.path` hash table erased

Node	Path	Compact Representation
X	$(X, 1, C, C{:}-20, A)$	
C_2	$(C_2, 3, X, 1, C, C{:}-20, A)$	$(C_2, 3, X)$ + `loc.path`(X)
A_2	$(A_2, c_2{:}2, C_2, 3, X, 1, C, C{:}-20, A)$	$(A_2, c_2{:}2, C_2)$ + `loc.path`(C_2)
W	$(W, 3, A_2, c_2{:}2, C_2, 3, X, 1, C, C{:}-20, A)$	$(W, 3, A_2)$ + `loc.path`(A_2)
T	$(T, 5, W, 3, A_2, c_2{:}2, C_2, 3, X, 1, C, C{:}-20, A)$	$(T, 5, W)$ + `loc.path`(W)

`edgeAnnotation` hash table

BypassEdge	Path
(T, A)	$(T, 5, W, 3, A_2, c_2{:}2, C_2, 3, X, 1, C, C{:}-20, A)$