Converting Simple Temporal Networks with Uncertainty into Dispatchable Form – Faster

Luke Hunsberger
Vassar College, Poughkeepsie, NY, USA

Roberto Posenato
University of Verona, Italy

Abstract

In many sectors of real-world industry, it is necessary to plan and schedule tasks allocated to agents participating in complex processes. Temporal planning aims to schedule tasks while respecting temporal constraints such as release times, maximum durations, and deadlines, which requires quantitative temporal reasoning. Over the years, several major application developers have highlighted the need for the explicit representation of actions with uncertain durations; efficient algorithms for determining whether plans involving such actions are controllable; and efficient algorithms for converting such plans into forms that enable them to be executed in real time with minimal computation, while preserving maximum flexibility.

A Simple Temporal Network with Uncertainty (STNU) is a data structure for reasoning about time constraints on actions that may have uncertain durations. An STNU is a triple $(\mathcal{T}, \mathcal{C}, \mathcal{L})$ where $\mathcal{T}$ is a set of real-valued variables called *timepoints*, $\mathcal{C}$ is a set of constraints of the form $Y - X \leq \delta$, where $X, Y \in \mathcal{T}$ and $\delta \in \mathbf{R}$, and $\mathcal{L}$ is a set of *contingent links* of the form (A, x, y, C), where $A, C \in \mathcal{T}$ and $0 < x < y < \infty$. A contingent link (A, x, y, C) represents an uncertain duration where A is the *activation timepoint*, C is the *contingent timepoint*, and $y - x$ is the uncertainty in the duration $C - A$. Typically, an executor controls the execution of A, but only observes the execution of C in real time. Although uncontrollable, the duration is guaranteed to satisfy $C - A \in [x, y]$. We let $n = |\mathcal{T}|$, $m = |\mathcal{C}|$ and $k = |\mathcal{L}|$.

An STNU graph is a pair $(\mathcal{T}, \mathcal{E})$, where the timepoints in $\mathcal{T}$ serve as nodes in the graph, and the edges in $\mathcal{E}$ correspond to the constraints in $\mathcal{C}$ and contingent links in $\mathcal{L}$. For each $Y - X \leq \delta$ in $\mathcal{C}$, $\mathcal{E}$ contains an *ordinary* edge $X \xrightarrow{\delta} Y$. For each $(A, x, y, C) \in \mathcal{L}$, $\mathcal{E}$ contains a *lower-case* (LC) edge, $A \xrightarrow{c:x} C$, and an *upper-case* (UC) edge, $C \xrightarrow{C:-y} A$, representing the respective possibilities that $C - A$ might take its minimum or maximum value. The *LO-edges* are the LC or ordinary edges; the *OU-edges* are the ordinary or UC edges.

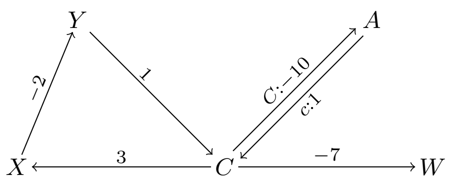
For any STNU, it is important to determine whether it is *dynamically controllable* (DC) (i.e., whether it is possible, in real time, to schedule its non-contingent timepoints such that all constraints will necessarily be satisfied no matter what durations turn out for the contingent links). Polynomial-time algorithms are available to solve this *DC-checking* problem. Each uses rules to generate new edges aiming to bypass certain kinds of edges in the STNU graph. Morris' $O(n^4)$-time DC-checking algorithm [3] starts from LC edges, propagating forward along OU-edges, looking for opportunities to generate new OU-edges that bypass the LC edges. Morris' $O(n^3)$-time algorithm [4] starts from negative OU-edges, propagating backward along LO-edges, aiming to bypass negative edges with non-negative edges. The $O(mn + k^2n + kn \log n)$-time RUL$^-$ algorithm [1] starts from UC edges, propagating backward along LO-edges, aiming to bypass UC edges with ordinary edges. After propagating, each algorithm checks for certain kinds of negative cycles to decide DC-vs.-non-DC.

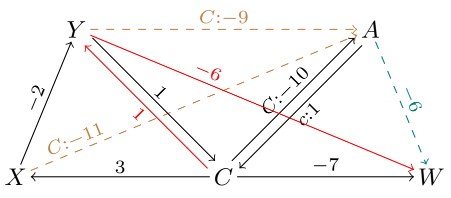
However, being DC only asserts the *existence* of a dynamic scheduler. It is also crucial to be able to *execute* a DC STNU efficiently in real time. For maximum flexibility and minimal space and time requirements, a dynamic scheduler for an STNU is typically computed *incrementally,* in real time, so that it can react to observations of contingent executions as they occur. An efficient dynamic scheduler can be realized by first transforming an STNU into an equivalent *dispatchable* form [6, 8]. Then, to execute the dispatchable STNU, it suffices to maintain *time-windows* for each timepoint and, as each timepoint X is executed, only updating time-windows for neighbors of X in the graph. Dispatchable STNUs are very important in applications that demand quick responses to observations of contingent events.

Of the existing DC-checking algorithms, only Morris' $O(n^3)$-time algorithm necessarily generates a dispatchable STNU for DC inputs. This abstract describes a faster, $O(mn + kn^2 + n^2 \log n)$-time algorithm for converting DC STNUs into dispatchable form. (The full journal article is available elsewhere [2].) This improvement is significant for applications (e.g., modeling business processes) where networks are typically sparse. For example, if $m = O(n \log n)$ and $k = O(\log n)$, then our algorithm runs in $O(n^2 \log n) \ll O(n^3)$ time.

Our new *Fast Dispatch* algorithm, $\mathrm{FD_{STNU}}$, has three phases. The first phase is similar to the RUL^- DC-checking algorithm, but generates an order-of-magnitude fewer edges overall, while also generating new UC edges that correspond to *wait constraints*. The second phase is a version of Morris' 2006 algorithm that propagates forward from LC edges, but only along LO-edges, aiming to generate *ordinary* bypass edges. The third phase focuses on the subgraph of *ordinary* edges, which comprise a Simple Temporal Network (STN). It uses an existing dispatchability algorithm for STNs [8] to convert that ordinary subgraph into a dispatchable STN. After completing the three phases, the STNU is guaranteed to be dispatchable.

The left-hand graph below is the graph for a sample STNU that happens to be DC.





The right-hand graph shows the edges inserted by our $\mathrm{FD_{STNU}}$ algorithm. The first phase applies the modified RUL^- algorithm, propagating backward from the original UC edge $(C, C{:}{-10}, A)$ to generate the (brown, dashed) wait edges $(Y, C{:}{-9}, A)$ and $(X, C{:}{-11}, A)$. (The original RUL^- algorithm only generates ordinary edges.) The second phase propagates *forward* from the LC edge $(A, c{:}1, C)$ to generate the (teal, dashed) edge $(A, -6, W)$. The third phase runs the pre-existing STN-dispatchability algorithm on the *ordinary* STN subgraph, inserting the (red) edges $(C, 1, Y)$ and $(Y, -6, W)$. Using the mathematical analysis of dispatchability due to Morris [5], it is not hard to confirm that this STNU is dispatchable.

We provide the source code of a Java implementation of the considered algorithms (Morris, RUL^-, and $\mathrm{FD_{STNU}}$) [7] and the benchmarks used to compare their performances.

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

Keywords and phrases Temporal constraint networks, contingent durations, dispatchable network

Digital Object Identifier 10.4230/LIPIcs.TIME.2023.20

Category Extended Abstract

Related Version *Full Version*: `https://doi.org/10.1016/j.ic.2023.105063` [2]

Supplementary Material *Software (Source code)*: `https://profs.scienze.univr.it/~posenato/software/cstnu/` [7]

Funding *Luke Hunsberger*: National Science Foundation [Grant No. 1909739].

References

1 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.

2 Luke Hunsberger and Roberto Posenato. A Faster Algorithm for Converting Simple Temporal Networks with Uncertainty into Dispatchable Form. *Information and Computation*, 293(105063):1–21, 2023. `doi:10.1016/j.ic.2023.105063`.

3 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`.

4 Paul Morris. Dynamic controllability and dispatchability relationships. In *CPAIOR 2014*, volume 8451 of *LNCS*, pages 464–479. Springer, 2014. `doi:10.1007/978-3-319-07046-9_33`.

5 Paul Morris. The Mathematics of Dispatchability Revisited. In *26th International Conference on Automated Planning and Scheduling (ICAPS-2016)*, pages 244–252, 2016. `doi:10.1609/icaps.v26i1.13739`.

6 Nicola Muscettola, Paul H. Morris, and Ioannis Tsamardinos. Reformulating Temporal Plans for Efficient Execution. In *6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-1998)*, pages 444–452, 1998.

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

8 Ioannis Tsamardinos, Nicola Muscettola, and Paul Morris. Fast Transformation of Temporal Plans for Efficient Execution. In *15th National Conf. on Artificial Intelligence (AAAI-1998)*, pages 254–261, 1998.