Converting Simple Temporal Networks with Uncertainty into Dispatchable Form—Faster

Luke Hunsberger
Vassar College
Poughkeepsie, NY USA

Roberto Posenato
Università di Verona
Verona, Italy

TIME–2023
Sept. 25, 2023

Acknowledgments

- ⋆ The extended abstract describes a recent journal article:

  A faster algorithm for converting simple temporal networks with uncertainty into dispatchable form, Luke Hunsberger and Roberto Posenato, Information and Computation, Vol. 293, 2023. `https://doi.org/10.1016/j.ic.2023.105063`.

- ⋆ The research was supported in part by:

  NSF Award # 1909739:
  RI: Small: RUI: Automated Reasoning about Time.

Introduction

Background

- Typically, building feasible real–world plans requires being able to do quantitative temporal reasoning.
- Competing formalisms have different expressiveness and computational complexity.
- Application developers have expressed the need for:
  1. representing actions with uncertain durations;
  2. efficient algorithms for checking whether plans with such actions are controllable; and
  3. efficient algorithms for converting such plans into a form that enables efficient real–time execution while preserving maximum flexibility.

To Meet These Needs

1. Simple Temporal Networks with Uncertainty (STNUs)
2. Polynomial–time dynamic–controllability (DC) checking algorithms
3. Polynomial–time algorithms for converting STNUs into *dispatchable* form

Simple Temporal Networks with Uncertainty (STNUs)

Simple Temporal Networks with Uncertainty (STNU)

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

$\mathcal{T}$: A set of *time–points*: $\{X, Y, Z, \ldots\}$
Real–valued variables representing events.

Simple Temporal Networks with Uncertainty (STNU)

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

$\mathcal{T}$: A set of *time-points*: $\{X, Y, Z, \ldots\}$
Real-valued variables representing events.

$\mathcal{C}$: A set of *simple temporal constraints*:
$Y - X \leq \delta$ or $Y - X \in [u, v]$.

Simple Temporal Networks with Uncertainty (STNU)

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

$\mathcal{T}$: A set of *time-points*: $\{X, Y, Z, \ldots\}$
Real-valued variables representing events.

$\mathcal{C}$: A set of *simple temporal constraints*:
$Y - X \leq \delta$ or $Y - X \in [u, v]$.

$\mathcal{L}$: A set of *contingent links*: (A, x, y, C)

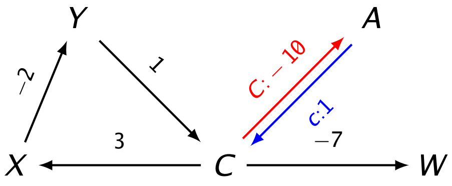
- A: *activation* time-point
- C: *contingent* time-point
- Uncertain duration: $C - A \in [x, y]$
- Executor only learns actual duration in real time.

Graphical Form of an STNU

Each STNU $\mathcal{S} = (\mathcal{T}, \mathcal{C}, \mathcal{L})$ can be represented by a graph $\mathcal{G} = (\mathcal{T}, \mathcal{E}, \mathcal{E}_L)$, where:

- Time–points serve as nodes in the graph
- Each constraint $(Y - X \leq \delta) \in \mathcal{C}$ corresponds to a labeled, directed edge in $\mathcal{E}$: $X \xrightarrow{\delta} Y$.
- Each contingent link $(A, x, y, C) \in \mathcal{L}$ corresponds to a pair of edges in $\mathcal{E}_L$ that represent uncontrollable possibilities:
  - Lower–case (LC) edge: $A \xrightarrow{c:x} C$
  - Upper–case (UC) edge: $C \xrightarrow{C:-y} A$

Sample STNU Graph



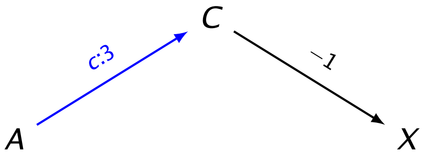
- The black (ordinary) edges represent constraints we want to satisfy.
- The red (upper-case) edge represents the uncontrollable possibility that the contingent duration $C - A$ might take on its maximum value 10.
- The blue (lower-case) edge represents the uncontrollable possibility that $C - A$ might take on its minimum value 1.

Dynamic Controllability

Dynamic Controllability

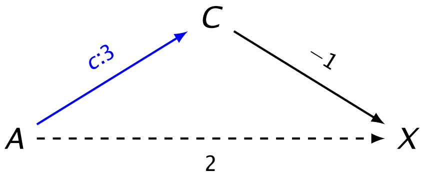
- An STNU is dynamically controllable (DC) if there exists a dynamic strategy for executing the non-contingent time-points such that all constraints in $\mathcal{C}$ will be satisfied no matter how the contingent durations turn out.
- A dynamic strategy can react to observations of contingent durations in real time, but its execution decisions cannot depend on durations that have not yet completed.
- Several DC-checking algorithms have been presented, each based on the propagation of constraints Morris [2006, 2014]; Morris and Muscettola [2005]; Nilsson et al. [2014].
- The fastest DC-checking algorithm is due to Cairo et al. [2018]. Its worst-case time-complexity is $O(mn + k^2 n + kn \log n)$, where $n = |\mathcal{T}|$, $m = |\mathcal{C}|$, $k = |\mathcal{L}|$.

Sample Constraint Propagation for STNUs



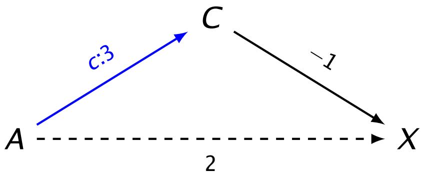
- The lower-case (LC) edge $(A, c{:}3, C)$ represents the uncontrollable possibility that $C - A$ might take on its min value 3.
- The ordinary edge $(C, -1, X)$ represents a constraint to be satisfied. (Requires X to be executed *before* C.)

Sample Constraint Propagation for STNUs



- The lower–case (LC) edge $(A, c{:}3, C)$ represents the uncontrollable possibility that $C - A$ might take on its min value 3.
- The ordinary edge $(C, -1, X)$ represents a constraint to be satisfied. (Requires X to be executed *before* C.)
- To guard against the possibility of $C - A = 3$, we must also satisfy the generated (dashed) edge $(A, 2, X)$.

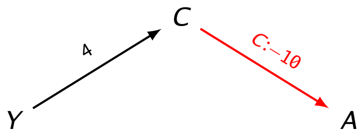
Sample Constraint Propagation for STNUs



- The lower–case (LC) edge $(A, c{:}3, C)$ represents the uncontrollable possibility that $C - A$ might take on its min value 3.
- The ordinary edge $(C, -1, X)$ represents a constraint to be satisfied. (Requires X to be executed *before* C.)
- To guard against the possibility of $C - A = 3$, we must also satisfy the generated (dashed) edge $(A, 2, X)$.

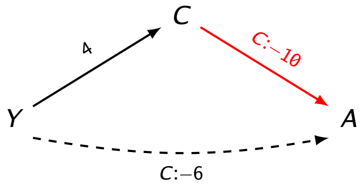
$\Rightarrow$ The generated edge *bypasses* the LC edge.

Sample Constraint Propagation for STNUs (ctd.)


$$\begin{tikzcd} & C \arrow[dr, "C:-10"] & \\ Y \arrow[ur, "4"] & & A \end{tikzcd}$$

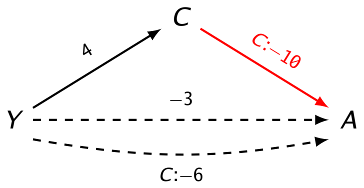
- Upper-case edge $(C, C{:}-10, A)$ represents the uncontrollable possibility that $C - A$ might take on its maximum value 10.
- The ordinary edge $(Y, 4, C)$ represents constraint to be satisified.

Sample Constraint Propagation for STNUs (ctd.)



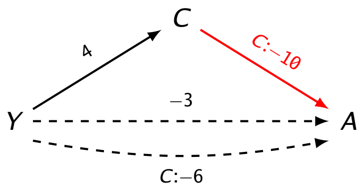
- Upper–case edge $(C, C{:}{-10}, A)$ represents the uncontrollable possibility that $C - A$ might take on its maximum value 10.
- The ordinary edge $(Y, 4, C)$ represents constraint to be satisified.
- To guard against the possibility of $C - A = 10$, we must also satisfy the conditional (dashed) wait constraint $(Y, C{:}{-6}, A)$ (i.e., as long as C not executed, Y must wait until at least 6 after A).

Sample Constraint Propagation for STNUs (ctd.)



- Upper–case edge $(C, C{:}{-10}, A)$ represents the uncontrollable possibility that $C - A$ might take on its maximum value 10.
- The ordinary edge $(Y, 4, C)$ represents constraint to be satisified.
- To guard against the possibility of $C - A = 10$, we must also satisfy the conditional (dashed) wait constraint $(Y, C{:}{-6}, A)$ (i.e., as long as C not executed, Y must wait until at least 6 after A).
- Since C *cannot* execute before $A + 3$ (where 3 is the min value of $C - A$), Y must unconditionally wait 3 after A.

Sample Constraint Propagation for STNUs (ctd.)



- Upper-case edge $(C, C{:}{-10}, A)$ represents the uncontrollable possibility that $C - A$ might take on its maximum value 10.
- The ordinary edge $(Y, 4, C)$ represents constraint to be satisified.
- To guard against the possibility of $C - A = 10$, we must also satisfy the conditional (dashed) wait constraint $(Y, C{:}{-6}, A)$ (i.e., as long as C not executed, Y must wait until at least 6 after A).
- Since C *cannot* execute before $A + 3$ (where 3 is the min value of $C - A$), Y must unconditionally wait 3 after A.
- Both of the generated edges *bypass* the UC edge.

DC–Checking Algorithms

- Morris [2006]: $O(n^4)$–time algorithm
- Morris [2014]: $O(n^3)$–time algorithm
- Cairo et al. [2018]: $O(mn + k^2 n + kn \log n)$–time algorithm

The above algorithms differ in:

- direction of propagation (forward or backward)
- the types of edges they aim to *bypass* (LC, UC, or any neg edges)
- whether they need to incrementally update a potential function
- the kinds of edges they generate
  (Ord+UC, only non–neg Ord, or any Ord)

STNU Dispatchability

Dispatchability

- The DC property only guarantees the existence of a dynamic execution strategy. It does not provide one.
- In any case, fully representing such a strategy would typically require exponential space.
- However, each DC STNU has an equivalent dispatchable form that enables (the relevant portion of) an execution strategy to be incrementally computed in real time, using only local propagation, while preserving maximum flexibility (Morris [2014]).
- The fastest previous algorithm for converting an STNU into dispatchable form is due to Morris [2014]. (It is a slight modification of his $O(n^3)$–time DC–checking algorithm.)

A Faster Algorithm: FD_{STNU}

⇒ This paper presents a faster $O(mn + kn^2 + n^2 \log n)$–time algorithm, called FD_{STNU}, for converting an STNU into an equivalent dispatchable form.

⇒ The new algorithm is particularly relevant for *sparse* graphs:

- If $m = O(n \log n)$ and $k = O(\log n))$, the running time reduces to $O(n^2 \log n) \ll O(n^3)$.
- If $m = O(n^{1.5})$ and $k = O(\sqrt{n})$, it reduces to $O(n^{2.5}) \ll O(n^3)$.

Background: Dispatchable STNs

- A *Simple Temporal Network* (STN) has no contingent links.
- STNs can be efficiently executed in real–time while requiring only minimal computation, as follows.
  - Maintain a time–window for each time–point.
  - Iteratively:
    - select a time–point to execute next;
    - then update the time–windows of only the *neighboring* time–points. (Local propagation.)

Background: Dispatchable STNs

- A *Simple Temporal Network* (STN) has no contingent links.
- STNs can be efficiently executed in real–time while requiring only minimal computation, as follows.
    - Maintain a time–window for each time–point.
    - Iteratively:
        - select a time–point to execute next;
        - then update the time–windows of only the *neighboring* time–points. (Local propagation.)
- Also requires keeping track when time–points become enabled.

Background: Dispatchable STNs

- A *Simple Temporal Network* (STN) has no contingent links.
- STNs can be efficiently executed in real–time while requiring only minimal computation, as follows.
  - Maintain a time–window for each time–point.
  - Iteratively:
    - select a time–point to execute next;
    - then update the time–windows of only the *neighboring* time–points. (Local propagation.)
- Also requires keeping track when time–points become enabled.

⇒ But not every consistent STN is *dispatchable*

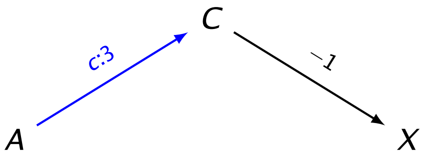
Converting STNs into Dispatchable Form

- Every consistent STN can be converted into an *equivalent* dispatchable STN (Muscettola et al. [1998]).
- Fast dispatchability algorithm for STNs runs in $O(mn + n^2 \log n)$ time (Tsamardinos et al. [1998]).
- Characteristic feature of *non-dispatchable* STNs is a *plus-minus* pair of edges (Morris [2016]):

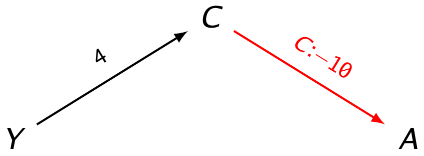
$$\begin{tikzcd}
 & V \arrow[dr, "-8"] & \\
U \arrow[ur, "5"] & & W
\end{tikzcd}$$

Recall the LC and UC rules

Most of the constraint–propagation rules used by the DC–checking algorithms bypass plus–minus pairs of edges!



$$\begin{tikzcd} & C \arrow[dr, "-1"] & \\ A \arrow[ur, "c:3"] & & X \end{tikzcd}$$



$$\begin{tikzcd} & C \arrow[dr, "C:-10"] & \\ Y \arrow[ur, "4"] & & A \end{tikzcd}$$

Recall the LC and UC rules

Most of the constraint–propagation rules used by the DC–checking algorithms bypass plus–minus pairs of edges!

$$\begin{tikzcd}
& C \arrow[dr, "-1"] & \\
A \arrow[ur, "c{:}3"] \arrow[rr, dashed, "2"'] & & X
\end{tikzcd}$$

$$\begin{tikzcd}
& C \arrow[dr, "C{:}-10"] & \\
Y \arrow[ur, "4"] \arrow[rr, dashed, "-3"] \arrow[rr, dashed, bend right, "C{:}-6"'] & & A
\end{tikzcd}$$

The FD$_{\mathrm{STNU}}$ Algorithm

Details of the $\mathrm{FD}_{\mathrm{STNU}}$ Algorithm

The $\mathrm{FD}_{\mathrm{STNU}}$ algorithm has three phases:

Phase 1: A back-propagation phase that is a modified version of the RUL^{-} DC-checking algorithm of Cairo et al. [2018]

Details of the FD_{STNU} Algorithm

The FD_{STNU} algorithm has three phases:

Phase 1: A back-propagation phase that is a modified version of the RUL^- DC-checking algorithm of Cairo et al. [2018]
$\Rightarrow$ Bypasses plus-minus pairs involving UC
(and possibly *some* LC) edges.

Details of the FD_{STNU} Algorithm

The FD_{STNU} algorithm has three phases:

Phase 1: A back-propagation phase that is a modified version of the RUL^{-} DC-checking algorithm of Cairo et al. [2018]
$\Rightarrow$ Bypasses plus-minus pairs involving UC (and possibly *some* LC) edges.

Phase 2: A forward-propagation phase that is a restricted version of the DC-checking algorithm of Morris [2006]

Details of the $\mathrm{FD}_{\mathrm{STNU}}$ Algorithm

The $\mathrm{FD}_{\mathrm{STNU}}$ algorithm has three phases:

Phase 1: A back–propagation phase that is a modified version of the RUL^{-} DC–checking algorithm of Cairo et al. [2018]
$\Rightarrow$ Bypasses plus–minus pairs involving UC
(and possibly *some* LC) edges.

Phase 2: A forward–propagation phase that is a restricted version of the DC–checking algorithm of Morris [2006]
$\Rightarrow$ Bypasses plus–minus pairs involving LC (but not UC) edges.

Details of the FD_{STNU} Algorithm

The FD_{STNU} algorithm has three phases:

Phase 1: A back-propagation phase that is a modified version of the RUL^{-} DC-checking algorithm of Cairo et al. [2018]
$\Rightarrow$ Bypasses plus-minus pairs involving UC (and possibly *some* LC) edges.

Phase 2: A forward-propagation phase that is a restricted version of the DC-checking algorithm of Morris [2006]
$\Rightarrow$ Bypasses plus-minus pairs involving LC (but not UC) edges.

Phase 3: An application of the dispatchability algorithm for STNs due to Tsamardinos et al. [1998].

Details of the FD_{STNU} Algorithm

The FD_{STNU} algorithm has three phases:

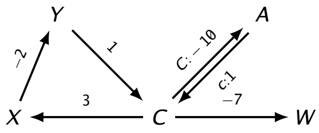
Phase 1: A back-propagation phase that is a modified version of the RUL^- DC-checking algorithm of Cairo et al. [2018]
$\Rightarrow$ Bypasses plus-minus pairs involving UC (and possibly *some* LC) edges.

Phase 2: A forward-propagation phase that is a restricted version of the DC-checking algorithm of Morris [2006]
$\Rightarrow$ Bypasses plus-minus pairs involving LC (but not UC) edges.

Phase 3: An application of the dispatchability algorithm for STNs due to Tsamardinos et al. [1998].
$\Rightarrow$ Bypasses plus-minus pairs of ordinary edges.

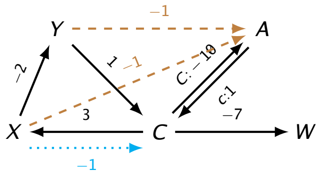
Phase 1: Backward Propagation

Like RUL$^-$, our Phase 1 propagates backward along lower-case and ordinary edges to generate edges that bypass upper-case edges.

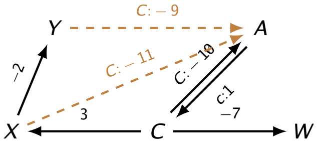


Sample DC STNU

Unlike RUL$^-$, our Phase 1 generates new upper-case edges, and many fewer ordinary edges.



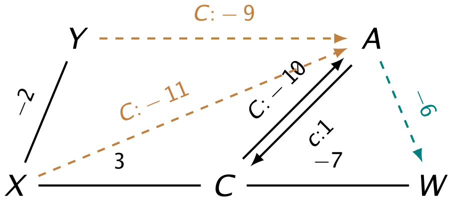
RUL$^-$ Output



Phase 1 Output

Phase 2: Forward Propagation

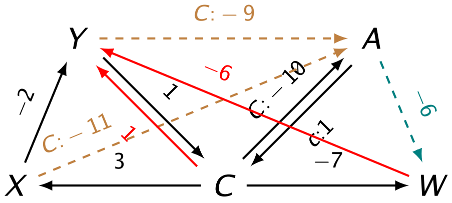
Like the DC-checking algorithm of Morris [2006], our Phase 2 propagates forward to to generate edges, like $(A, -6, W)$ below, that bypass lower-case edges.



Unlike Morris' algorithm, our Phase 2 does not propagate along upper-case edges and, thus, does not need to iteratively update a potential function. It also generates many fewer edges.

Phase 3: Using STN Dispatchability Algorithm

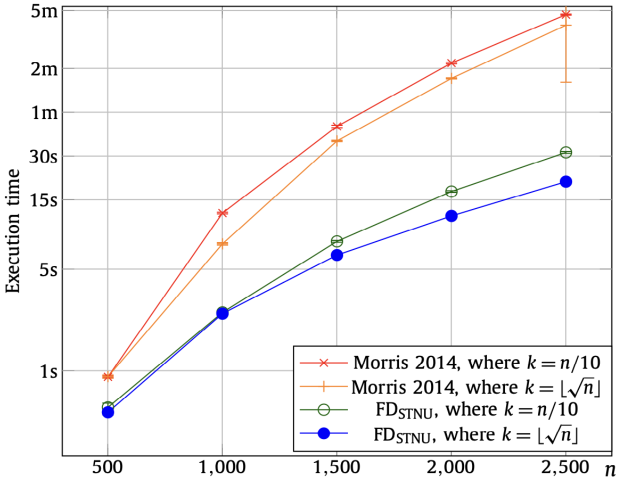
Our Phase 3 applies the STN dispatchability algorithm of Tsamardinos et al. [1998] to convert the subgraph of ordinary edges into a dispatchable STN subgraph.



After completing Phase 3, the resulting STNU graph is guaranteed to be dispatchable (i.e., able to be executed efficiently in real time, using only local propagation, while preserving maximum flexibility).

Empirical Evaluation

Empirical Evaluation



Empirical Evaluation (ctd.)

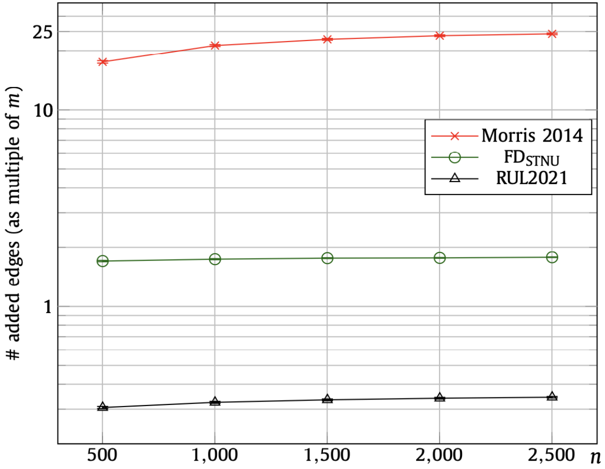


Fig. 12. Number of added edges vs number of nodes, n.

Empirical Evaluation (ctd.)

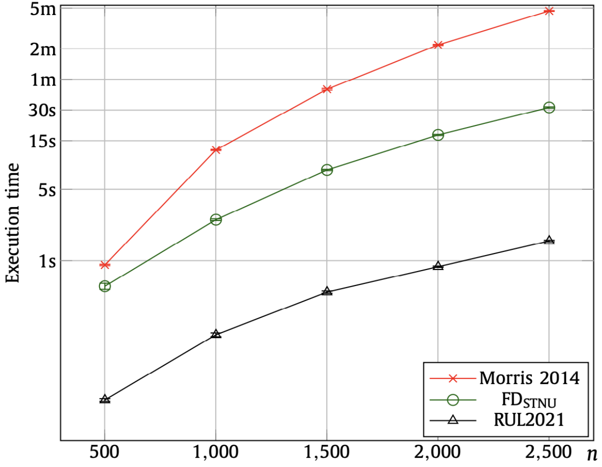


Fig. 11. Execution time vs number of nodes, n.

Future Work

- Compute *minimal* dispatchable STNU.
- Extend dispatchability theory/algorithms to more expressive networks (e.g., Conditional STNs and Conditional STNUs) (Tsamardinos et al. [2003], Combi et al. [2013]).

References I

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.

Carlo Combi, Luke Hunsberger, and Roberto Posenato. An algorithm for checking the dynamic controllability of a conditional simple temporal network with uncertainty. In *5th Int. Conf. on Agents and Artificial Intelligent (ICAART-2013)*, volume 2, pages 144-156, 2013. ISBN 9789898565389. doi: 10.5220/0004256101440156.

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. ISSN 0890-5401. doi: 10.1016/j.ic.2023.105063.

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.

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.

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.

References II

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`.

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.

Mikael Nilsson, Jonas Kvarnstrom, and Patrick Doherty. Incremental dynamic controllability in cubic worst–case time. In *21st International Symposium on Temporal Representation and Reasoning (TIME–2014)*, 2014.

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.

Ioannis Tsamardinos, Thierry Vidal, and Martha E. Pollack. CTP: A new constraint–based formalism for conditional, temporal planning. *Constraints*, 8:365-388, 2003. doi: 10.1023/A:1025894003623.