Faster Algorithm for Converting an STNU into Minimal Dispatchable Form

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 Network with Uncertainty (STNU):
  - A data structure for representing and reasoning about actions with *uncertain, but bounded durations* [7].
- Dynamic Controllability (DC) property:
  - Ensures existence of a dynamic execution strategy satisfying all the constraints no matter how the uncertain durations turn out [7].
- Dispatchability property (for DC STNUs):
  - Ensures STNU can be successfully executed in real time, maintaining max flexibility but requiring only min computation [5].
  - A DC STNU can be converted into an equivalent dispatchable network having a minimal number of edges in $O(kn^3)$ time [3].
- We wanted to reduce the computational time of this conversion.
- In this work we present a faster $O(n^3)$–time algorithm for converting a DC STNU into an equivalent dispatchable network having a minimal number of edges.

Overview

In this presentation, we:

- present the foundations of dispatchability for STNs and STNUs;
- an example of the critical phase of the previous algorithm for converting a DC STNU into a minimal dispatchable form;
- the idea how we simplified the critical phase obtaining the new algorithm;

DC–Checking Algorithms for STNUs

- Different DC–checking algorithms use different sets of rules to propagate constraints (equiv., generate new edges from old) [7, 8, 4, 5, 1, 2]
- Most DC–checking algorithms generate a new kind of constraint/edge called a *wait:* [8]

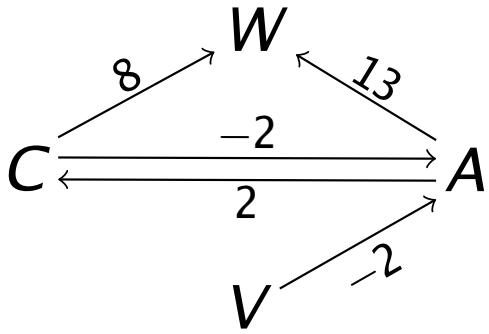
  "While (contingent) C unexecuted, Y must wait *at least* 5 *after A."*

  $$Y \xrightarrow{C:-5} A$$

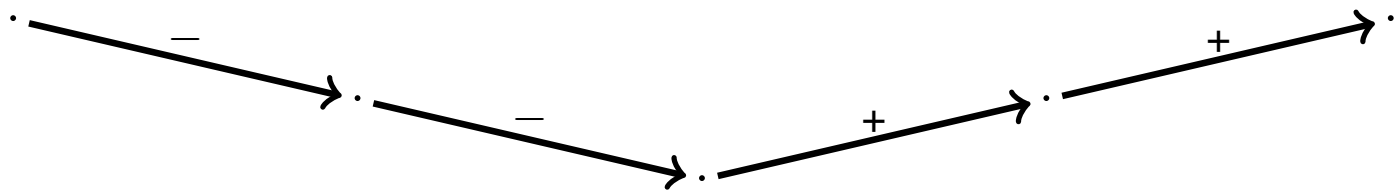
- An STNU, together with generated waits, is called an extended STNU (ESTNU) [5].
- Wait constraints are necessary for the dispatchability. So, here we consider the dispatchability on ESTNUs.

Graphical Characterization of STN Dispatchability

- A Simple Temporal Network (STN) is an STNU without any contingent links.



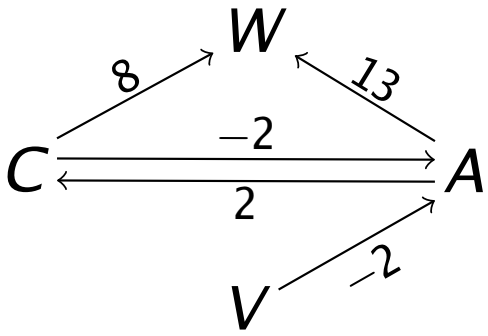
- Morris [6]: A Simple Temporal Network (STN) $\mathcal{S}$ is dispatchable iff for each pair of timepoints X and Y, if there is a path from X to Y in $\mathcal{S}$, then there is a shortest path from X to Y that is a vee–path.
  - A vee–path (V–path) is a path comprising zero or more negative edges followed by zero or more non–negative edges.



  - A vee–path that is a shortest path is called a shortest vee–path.

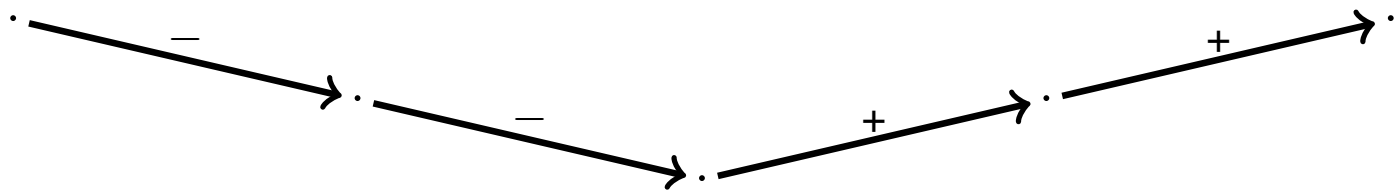
Graphical Characterization of STN Dispatchability

- A Simple Temporal Network (STN) is an STNU without any contingent links.



Dispatchable!
There is a shortest vee–path for each pair of timepoints connected by a path!

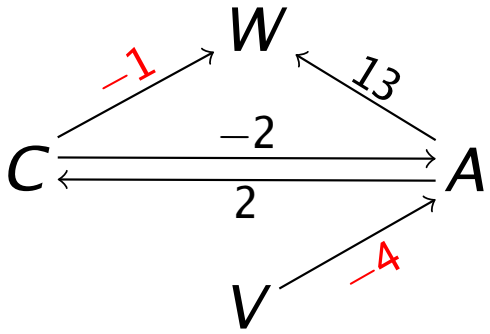
- Morris [6]: A Simple Temporal Network (STN) $\mathcal{S}$ is dispatchable iff for each pair of timepoints X and Y, if there is a path from X to Y in $\mathcal{S}$, then there is a shortest path from X to Y that is a vee–path.
  - A vee–path (V–path) is a path comprising zero or more negative edges followed by zero or more non–negative edges.



  - A vee–path that is a shortest path is called a shortest vee–path.

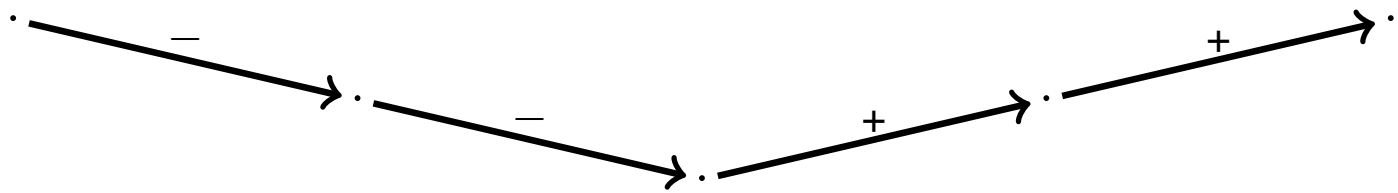
Graphical Characterization of STN Dispatchability

- A Simple Temporal Network (STN) is an STNU without any contingent links.



Not Dispatchable!
Shortest path from V to W has length -3, but no shortest vee–path from V to W.

- Morris [6]: A Simple Temporal Network (STN) $\mathcal{S}$ is dispatchable iff for each pair of timepoints X and Y, if there is a path from X to Y in $\mathcal{S}$, then there is a shortest path from X to Y that is a vee–path.
  - A vee–path (V–path) is a path comprising zero or more negative edges followed by zero or more non–negative edges.



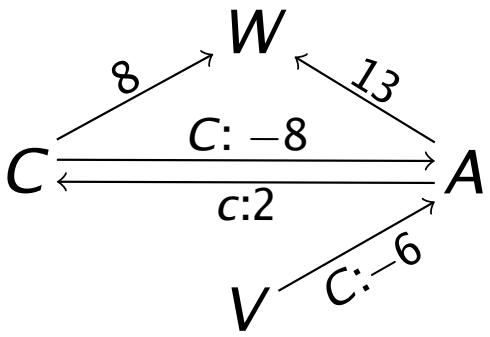
  - A vee–path that is a shortest path is called a shortest vee–path.

Graphical Characterization of ESTNU Dispatchability

Stand–in Edges

Morris [5]: **An ESTNU $\mathcal{S}$ is dispatchable iff each of its STN projections is dispatchable (as an STN).**

- An STN projection is an STN obtained from an ESTNU by constraining all contingent links to fixed values.

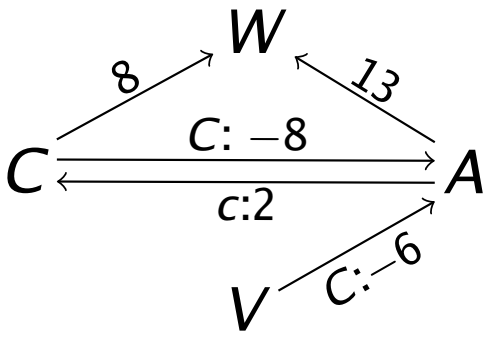


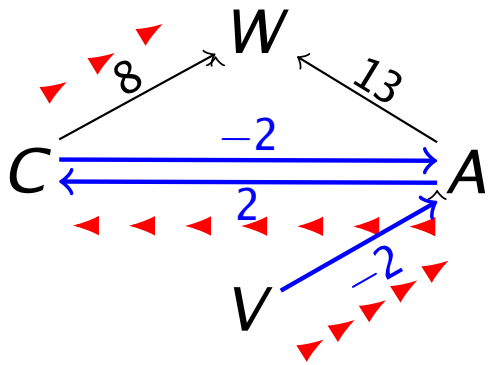
Graphical Characterization of ESTNU Dispatchability

Stand–in Edges

Morris [5]: **An ESTNU $\mathcal{S}$ is dispatchable iff each of its STN projections is dispatchable (as an STN).**

- An STN projection is an STN obtained from an ESTNU by constraining all contingent links to fixed values.





$C - A = 2$

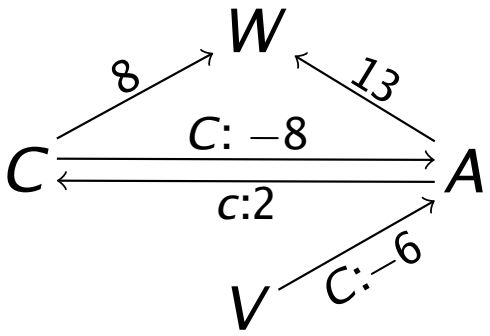
$|VACW| = 8$

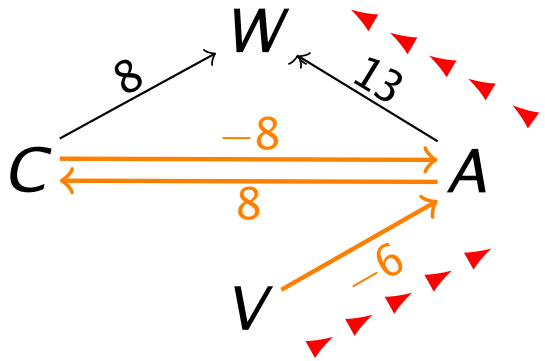
Graphical Characterization of ESTNU Dispatchability

Stand–in Edges

Morris [5]: **An ESTNU $\mathcal{S}$ is dispatchable iff each of its STN projections is dispatchable (as an STN).**

- An STN projection is an STN obtained from an ESTNU by constraining all contingent links to fixed values.





$C - A = 8$

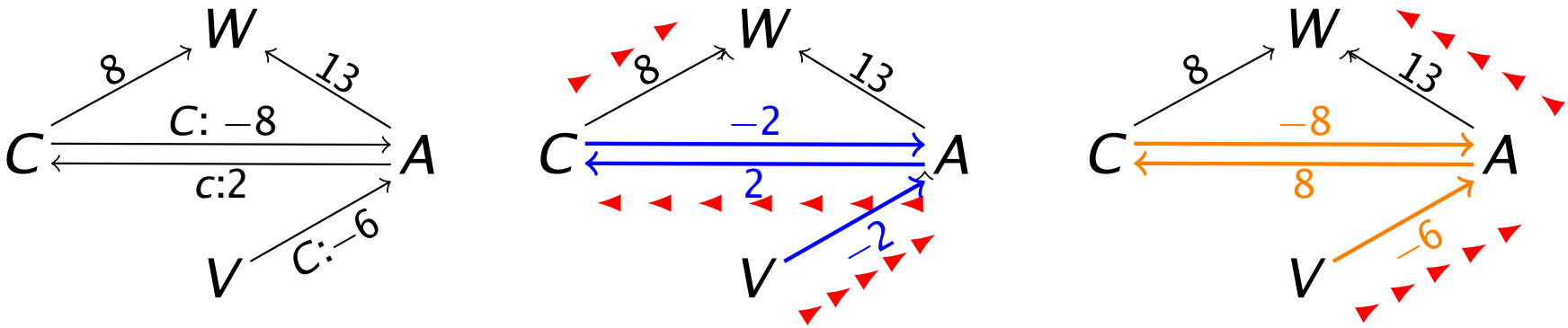
$|VAW| = 7$

Graphical Characterization of ESTNU Dispatchability

Stand–in Edges

Morris [5]: **An ESTNU $\mathcal{S}$ is dispatchable iff each of its STN projections is dispatchable (as an STN).**

- An STN projection is an STN obtained from an ESTNU by constraining all contingent links to fixed values.



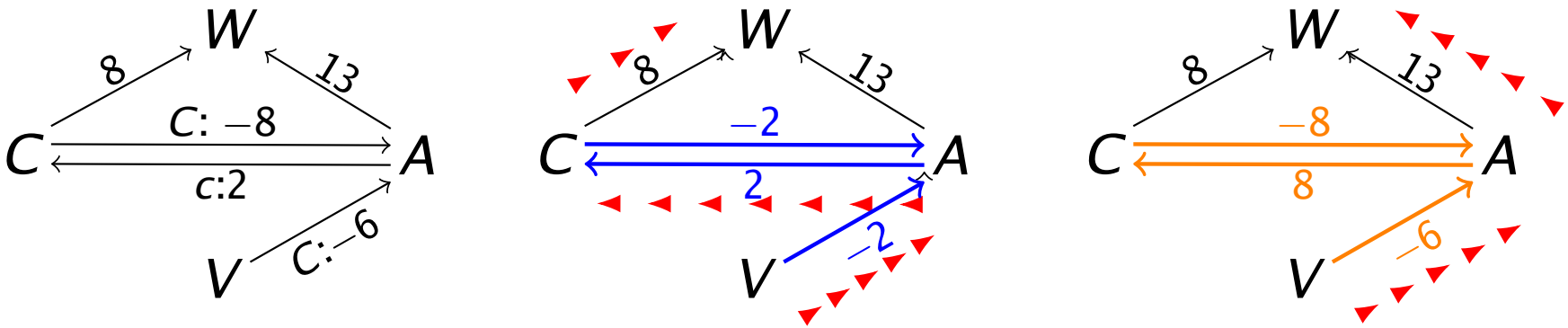
The above ESTNU is dispatchable because each of its STN projections is.

Graphical Characterization of ESTNU Dispatchability

Stand–in Edges

Morris [5]: **An ESTNU $\mathcal{S}$ is dispatchable iff each of its STN projections is dispatchable (as an STN).**

- An STN projection is an STN obtained from an ESTNU by constraining all contingent links to fixed values.



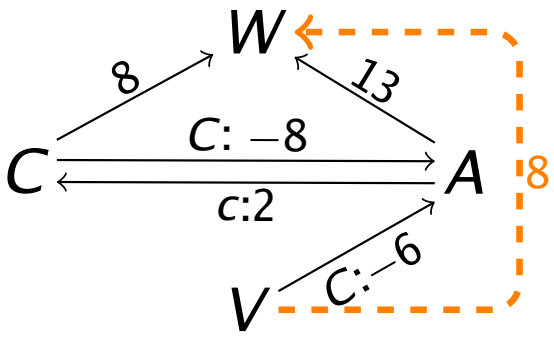
The above ESTNU is dispatchable because each of its STN projections is.

Key observation: Shortest vee–path from V to W can take a different route in different projections and the minimization depends on shortest vee–paths!

Graphical Characterization of ESTNU Dispatchability

Stand–in Edges

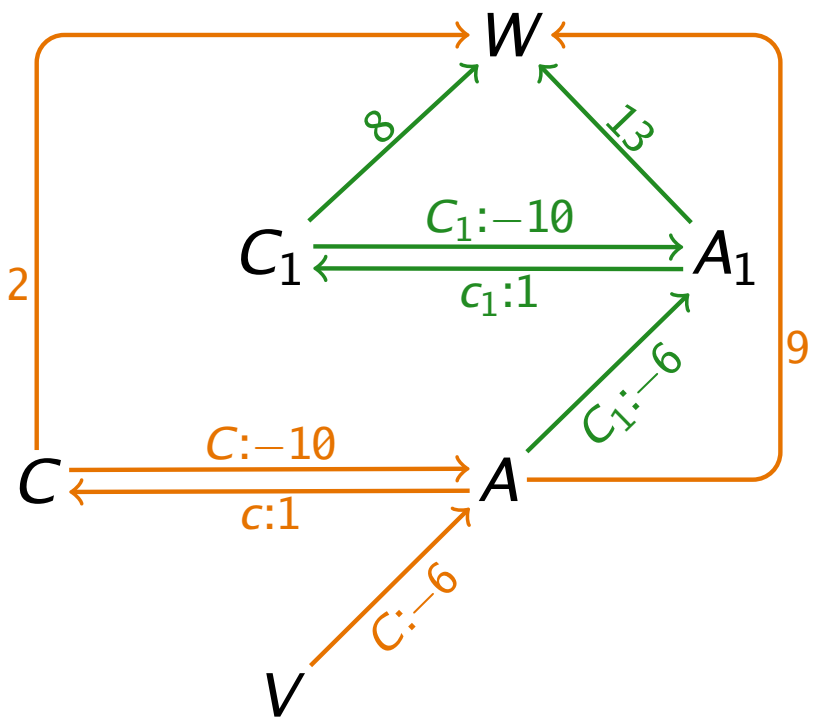
We proved it is possible to represent different shortest vee–paths in the different STN projections as a **single temporary ordinary constraint**, called "stand–in" constraint. [3]



- "stand–in" edge is the maximum shortest vee–path from V to W.
- Given a pair (V, W), it can be determined in constant time.

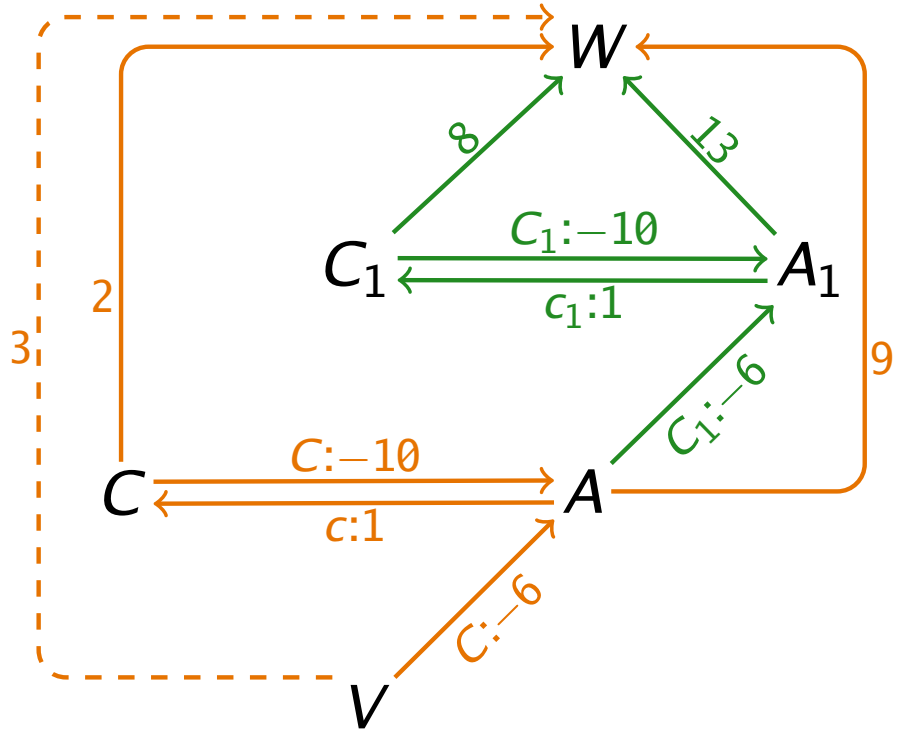
- They are removed at the end.

Edges Entailed by Nested Diamond Structures



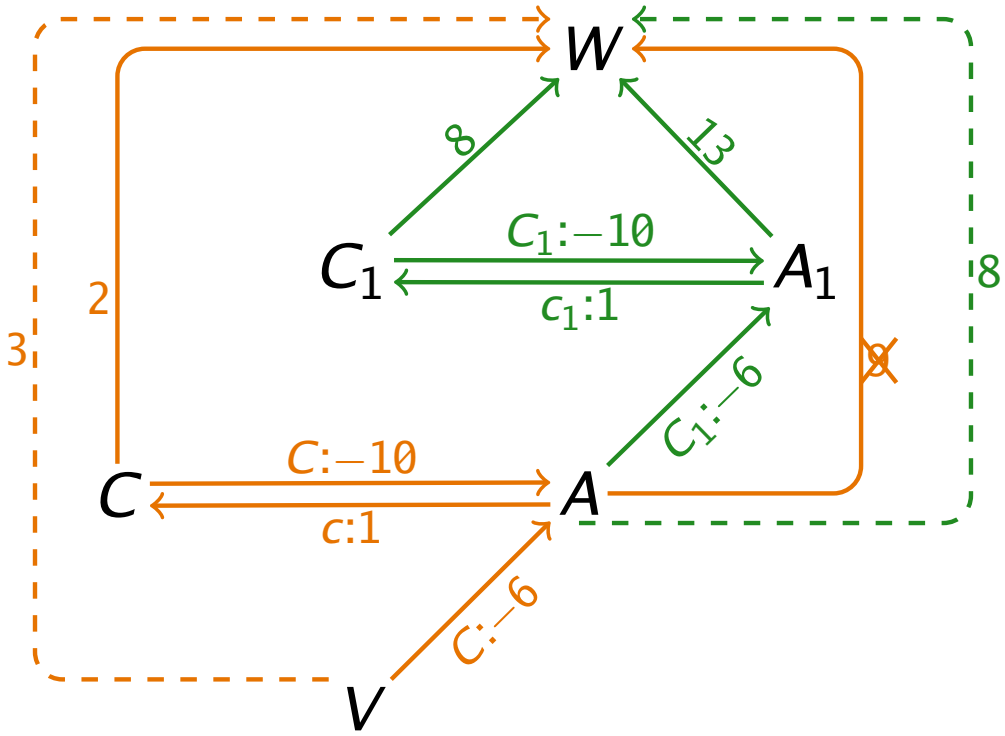
"Stand–in" edges are very usefull, but their determination can be **expensive** when the network becomes more complex.

Edges Entailed by Nested Diamond Structures



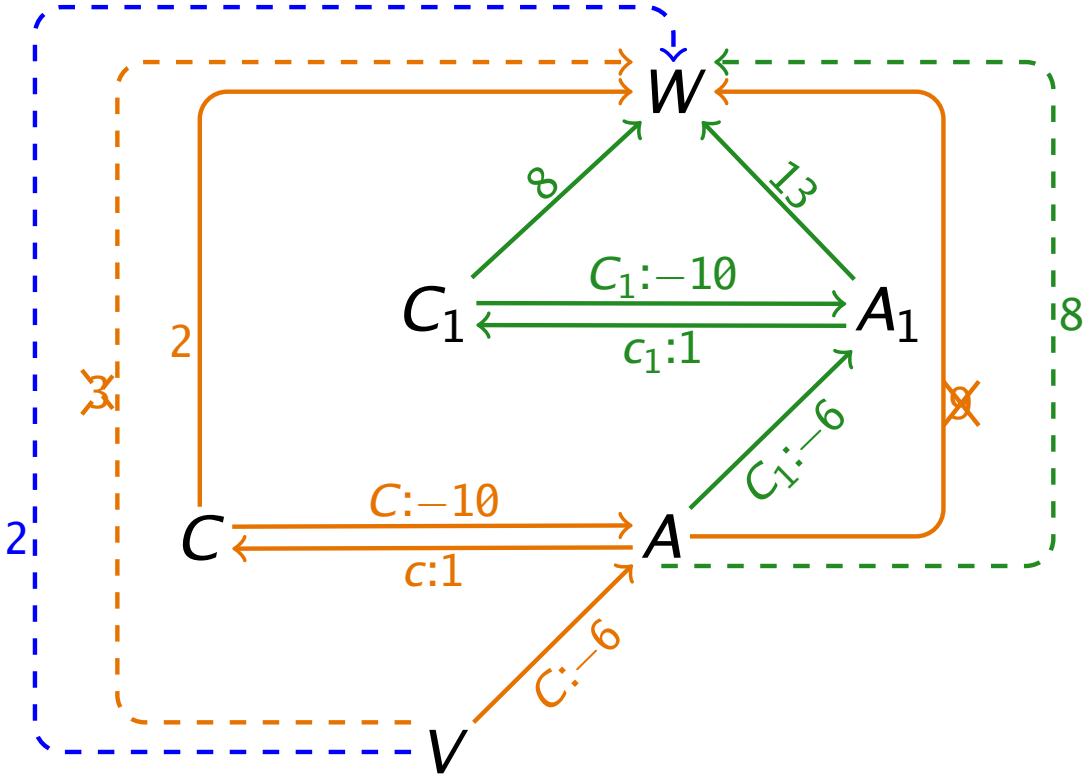
The orange edges entail the dashed orange "stand–in" edge in every projection.

Edges Entailed by Nested Diamond Structures



The green edges entail the dashed green "stand–in" edge in every projection.

Edges Entailed by Nested Diamond Structures



The orange edges, together with the dashed green edge, entail the dashed blue "stand–in" edge in every projection.

Complexity of Previous Algorithm [3]

- To generate all "stand–in" edges, it suffices to explore nested diamond structures up to a maximum depth of k [3].
- After each iteration, to update the distances of **all** ordinary paths affected by inserting the new "stand–in" edges, the algorithm calls Johnson's algorithm yielding an overall time–complexity of $O(kn^3)$ [3].

New Algorithm

Key Insights

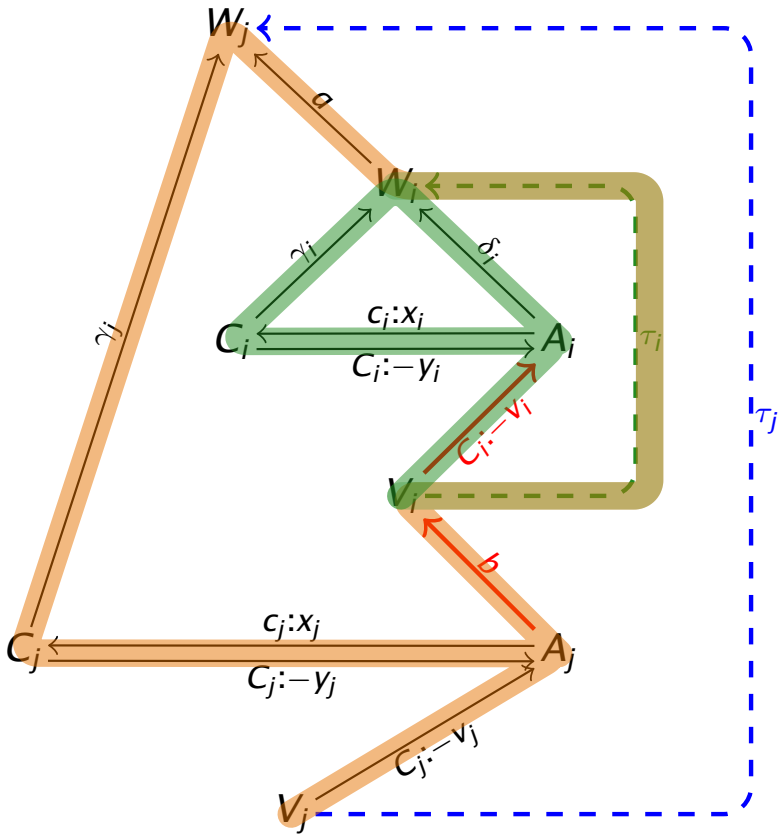
- Previous algorithm:
  - generates more dashed edges than necessary on each iteration; and
  - so cannot obtain any savings from determining an optimal order in which to process nested diamond structures.
- New algorithm:
  - determines an optimal partial order for processing nested diamonds;
  - still does k iterations;
  - after each iteration, updates a more restrictive (but still sufficient) set of distances affected by the new "stand–in" edges.
  - does one final iteration to compute complete set of ordinary distances.

New Algorithm

Key Insight RE: Order of Nesting of Diamond Structures

We formally proved that:

- when exploring the diamond $\langle V_i, A_i, C_i, W_i \rangle$, nested *inside* the diamond $\langle V_j, A_j, C_j, W_j \rangle$ ...
- relevant stand–in edges (e.g., (V_i, τ_i, W_i) and (V_j, τ_j, W_j)) are possible only if the path from A_j to A_i comprises zero or more negative ordinary edges followed by a single wait edge.
  We call such paths negOrdWait paths.
- negOrdWait paths determine a strict partial order on relevant nestings.



New Algorithm

`getPCInfo`: Get Parent/Child Info for Nesting Order

`getPCInfo` Algorithm:

- Does k iterations.
- On each iteration, starts from an activation timepoint A_i, back–propagating along negOrdWait paths.
  - Uses potential function to re–weight ordinary and upper–case edges, thereby allowing Dijkstra–like traversal of edges.
  - Records instances of any negOrdWait paths that it discovers from any activation timepoint A_j to A_i. (A_j is the parent, A_i is the child.)
- Populates vectors of (redundant) parent/child hash–tables:
  - For each A_i, $\texttt{parent}[A_i]$ = list of parent activation timepoints.
  - For each A_j, $\texttt{child}[A_j]$ = list of child activation timepoints.
- Overall complexity of k iterations of Dijkstra–like traversal is $O(mk + kn \log n) \in O(n^3)$.

`newGenStandIns`: (New) Generate Stand–In Edges

- Initialize a potential function for the ordinary edges.
- Process diamond structures according to a topological sort based on the strict partial order computed by `getPCinfo`: do not process diamonds involving (A_j, x_j, y_j, C_j) until all of A_j's children have already been processed.
- Instead of computing all ordinary distances affected by stand–in edges entailed (A_j, x_j, y_j, C_j), only need to update ordinary distances of paths emanating from A_j (i.e., single source).
- Use Dijkstra–like traversal of paths emanating from A_j. Then update the potential function.
- After all nested diamonds have been processed, one iteration of the original `genStandIns` algorithm generates all remaining stand–in edges, followed by a single call to Johnson's algorithm to update ordinary distances.

Complexity of New Algorithm

- The k iterations of `newGenStandIns` take a total of $O(mk + nk^2 + kn \log n)$ time (instead of $O(kn^3)$).
- The single call to Johnson's algorithm after `newGenStandIns` costs $O(n^3)$ time.
- Therefore, the complexity of the new algorithm is dominated by the final call to Johnson's algorithm: $O(n^3)$.

Conclusions

- Presented a new, faster $O(n^3)$–time algorithm that, given a DC STNU, computes an equivalent dispatchable ESTNU having a minimal number of edges.
- The previous algorithm had a worst–case time–complexity of $O(kn^3)$.
- The new algorithm achieves its better performance by:
  - Computing a strict partial order among the activation timepoints that it uses to order its processing of the corresponding nested diamond structures; and
  - On each iteration, only updating distances of paths emanating from a single source (the activation timepoint).

References I

[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.
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.

References II

[3] Luke Hunsberger and Roberto Posenato.
Converting Simple Temporal Networks with Uncertainty into Minimal Equivalent Dispatchable Form.
In *Proceedings of the Thirty–Fourth International Conference on Automated Planning and Scheduling (ICAPS 2024)*, volume 34, pages 290–300, 2024.
doi:10.1609/icaps.v34i1.31487.

[4] 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.

References III

[5] 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.

[6] 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.

[7] 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 IV

[8] 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.