Time-based Dynamic Controllability of Disjunctive Temporal Networks with Uncertainty: A Tree Search Approach with Graph Neural Network Guidance

Kevin Osanlou[1,2,3,4], Jeremy Frank[1], J. Benton[1], Andrei Bursuc[5], Christophe Guettier[2], Eric Jacopin[6] and Tristan Cazenave[3]

[1] NASA Ames Research Center [2] Safran Electronics & Defense [3]LAMSADE, Paris-Dauphine
[4] Universities Space Research Association [5]valeo.ai [6]CREC Saint-Cyr Coetquidan

{kevin.osanlou, jeremy.d.frank, j.benton}@nasa.gov
{kevin.osanlou, christophe.guettier}@safrangroup.com andrei.bursuc@valeo.com
eric.jacopin@st-cyr.terre-net.defense.gouv.fr tristan.cazenave@lamsade.dauphine.fr

Abstract

Scheduling in the presence of uncertainty is an area of interest in artificial intelligence due to the large number of applications. We study the problem of dynamic controllability (DC) of disjunctive temporal networks with uncertainty (DTNU), which seeks a strategy to satisfy all constraints in response to uncontrollable action durations. We introduce a more restricted, stronger form of controllability than DC for DTNUs, time-based dynamic controllability (TDC), and present a tree search approach to determine whether or not a DTNU is TDC. Moreover, we leverage the learning capability of a message passing neural network (MPNN) as a heuristic for tree search guidance. Finally, we conduct experiments for which the tree search shows superior results to state-of-the-art timed-game automata (TGA) based approaches, effectively solving fifty percent more DTNU problems on a known benchmark. We also observe that MPNN tree search guidance leads to substantial performance gains on benchmarks of more complex DTNUs, with up to eleven times more problems solved than the baseline with the same time budget.

1 Introduction

Temporal Networks (TN) are a common formalism to represent temporal constraints over a set of time points (*e.g.* start/end of activities in a scheduling problem). The Simple Temporal Networks with Uncertainty (STNUs) (Tsamardinos 2002) (Vidal and Fargier 1999) explicitly incorporate qualitative uncertainty into temporal networks. Considerable work has resulted in algorithms to determine whether or not all timepoints can be scheduled, either up-front or reactively, in order to account for uncertainty (*e.g.* (Morris and Muscettola 2005), (Morris 2014)). In particular, an STNU is *dynamically controllable* (DC) if there is a reactive strategy in which controllable timepoints can be executed either at a specific time, or after observing the occurrence of an uncontrollable timepoint. Cimatti et al. (Cimatti, Micheli, and Roveri 2016) investigate the problem of DC for Disjunctive Temporal Networks with Uncertainty (DTNUs), which generalize STNUs. Figure 1a shows two DTNUs γ and γ' on the left side; a_i are controllable timepoints, u_j are uncontrollable timepoints. Timepoints are variables which can take on any value in $\mathbb{R}$. Constraints between timepoints characterize a minimum and maximum time distance separating them, likewise valued in $\mathbb{R}$. The key difference between STNUs and DTNUs lies in the *disjunctions* that yield more choice points for consistent scheduling, especially reactively.

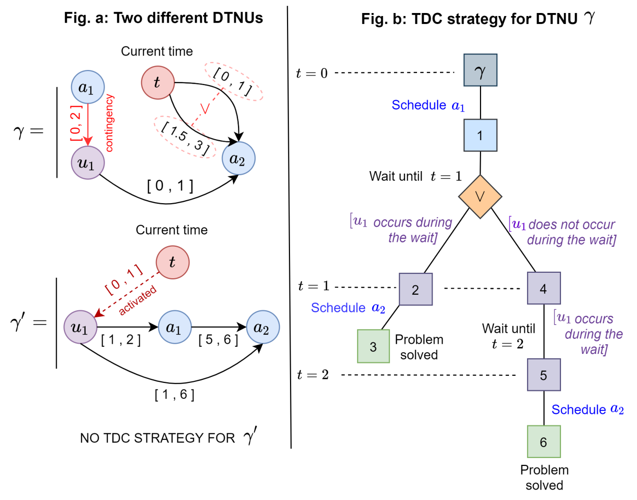


Figure 1: **Two example DTNUs γ and γ'.** In both examples, timepoints a_1 and a_2 are controllable; u_1 is uncontrollable. Black arrows and their intervals represent time constraints between timepoints; the light red arrow and its interval contingency links. The dashed dark red arrow in γ' implies u_1 has already been activated and will occur in the specified interval. A TDC strategy is displayed for γ. Nodes below γ are sub-DTNUs except the $\vee$ node which lists transitional possibilities. DTNU γ', on the other hand, is an example of a DTNU which is DC but not TDC.

The complexity of DC checking for DTNUs is $PSPACE$-complete (Bhargava and Williams 2019), making this a highly challenging problem. The difficulty in proving or disproving DC arises from the need to check all possible combinations of disjuncts in order to handle all possible occurrence outcomes of the uncontrollable timepoints. The best previous approaches for this problem use timed-game automata (TGAs) and Satisfiability Modulo Theories (SMTs), described in (Cimatti, Micheli, and Roveri 2016).

A new emerging trend of neural networks, graph-based neural networks (GNNs), have been proposed as an extension of convolutional neural networks (CNNs) (Krizhevsky, Sutskever, and Hinton 2012) to graph-structured data. Recent variants based on spectral graph theory include (Defferrard, Bresson, and Vandergheynst 2016), (Li et al. 2016), (Kipf and Welling 2017). They take advantage of relational properties between nodes for classification, but do not take into account potential edge weights. In newer approaches, Message Passing Neural Networks (MPNNs) with architectures such as in (Battaglia et al. 2016), (Gilmer et al. 2017) and (Kipf et al. 2018) use embeddings comprising edge weights within each computational layer. We focus our interest on these architecture types as DTNUs can be formalized as graphs with edge distances representing time constraints.

In this work, we study DC checking of DTNUs as a search problem, express states as graphs, and use MPNNs to learn heuristics based on previously solved DTNUs to guide search. The key contributions of our approach are the following. **(1)** We introduce a time-based form of dynamic controllability (TDC) and a tree search approach to identify TDC strategies. We informally show that TDC implies DC, but the opposite is not generally true. **(2)** We describe an MPNN architecture for handling DTNU scheduling problems and use it as heuristic for guidance in the tree search. Moreover, we define a self-supervised learning scheme to train the MPNN to solve randomly generated DTNUs with short timeouts to limit search duration. **(3)** We introduce constraint propagation rules which enable us to enforce time domain restrictions for variables in order to ensure soundness of strategies found. We carry out experiments showing that the tree search algorithm improves performance and scalability over the best previous DC-solving approach in (Cimatti, Micheli, and Roveri 2016), PYDC-SMT, with 50% more DTNU instances solved. Moreover, we expose that the learned MPNN heuristic considerably improves the tree search on harder DTNUs: performance gains go up to 11 times more instances solved within the same time frame. Results also highlight that the MPNN, which is trained on a set of solved DTNUs, is able to generalize to larger DTNUs.

2 Time-based Dynamic Controllability

A DC *strategy* for a DTNU either executes controllable timepoints at a specific time, or reacts to the occurrence of an uncontrollable timepoint. We present our TDC formalism here. A TDC strategy executes controllable timepoints at specific times under the assumption that some uncontrollable timepoints may occur or not in a given time interval. Each interval in a TDC strategy can have an arbitrary duration. Controllable timepoints are usually executed at the start or the end of an interval, while uncontrollable timepoints may occur inside the interval. TDC also makes it possible to execute a controllable timepoint at the exact same time as the occurrence time of an uncontrollable timepoint inside the interval, with a *reactive execution*.

TDC is less flexible than a DC strategy which can wait for an uncontrollable timepoint to occur before making a new decision. Conversely TDC does not allow, for instance, a delayed reactive execution of a controllable timepoint in response to an uncontrollable one. TDC is a subset of DC, and a stronger form of controllability: TDC implies DC.

DTNU γ' in Figure 1a shows an example of an STNU which is not TDC but DC. In this example, uncontrollable timepoint u_1 is activated, *i.e.* the controllable timepoint associated to u_1 in the contingency links has been executed. Moreover, it is known that u_1 occurs between t and $t+1$, where t is the current time. The interval $[t, t+1]$ is referred to as the *activation time interval* for u_1. Controllable timepoint a_1 must be executed at least 1 time unit after u_1, and controllable timepoint a_2 at least 5 time units after a_1. However, controllable timepoint a_2 cannot be executed later than 6 time units after u_1. A valid DC strategy waits for u_1 to occur, then schedules a_1 exactly 1 time unit later, and a_2 5 time units after a_1. However, for any TDC strategy, there is no wait duration small enough while waiting for u_1 to happen that does not violate these constraints. There will always be some strictly positive lapse of time between the moment u_1 occurs and the end of the wait. The exact execution time of u_1 during the wait is unknown: a TDC strategy therefore assumes u_1 happened at the end of the wait when trying to schedule a_1 at the earliest. Therefore, the earliest time a_1 can be scheduled in a TDC strategy is 1 time unit after the end of the wait, which is too late.

3 Tree Search Preliminaries

We introduce here the tree search algorithm. The approach discretizes uncontrollable durations, *i.e.* durations when one or several uncontrollable timepoints can occur, into reduced intervals. These are in turn used to account for possible outcomes of uncontrollable timepoints and adapt the scheduling strategy accordingly. The root of the search tree built by the algorithm is a DTNU, and other tree nodes are either sub-DTNUs or logical nodes (*OR, AND*) which respectively represent decisions that can be made and how uncontrollable timepoints can unfold. At a given DTNU tree node, decisions such as executing a controllable timepoint or waiting for a period of time develop children DTNU nodes for which these decisions are propagated to constraints. The TDC controllability of a *leaf* DTNU, *i.e.* a sub-DTNU for which all controllable timepoints have been executed and uncontrollable timepoints are assumed to have occurred in specific intervals, indicates whether or not this sub-DTNU has been solved at the end of the scheduling process. We also refer to the TDC controllability of a DTNU node in the search tree as its *truth attribute*. Lastly, the search logically combines TDC controllability of children DTNUs to determine TDC controllability for parent nodes. We give a simple example of a TDC strategy for a DTNU γ in Figure 1.

Let $\Gamma = \{A, U, C, L\}$ be a DTNU. A is the list of controllable timepoints, U the list of uncontrollable timepoints, C the list of constraints and L the list of contingency links. The root node of the search tree is Γ. There are four different types of nodes in the tree and each node has a *truth* attribute (see §4.4) which is initialized to *unknown* and can be set to either *true* or *false*. The different types of tree nodes are listed below and shown in Figure 2.

***DTNU* nodes.** Any DTNU node other than the original problem Γ corresponds to a sub-problem of Γ at a given point in time t, for which some controllable timepoints may have already been scheduled in upper branches of the tree, some amount of time may have passed, and some uncontrollable timepoints are assumed to have occurred. A DTNU node is made of the same timepoints A and U, constraints C and contingency links L as DTNU Γ. It also carries a schedule memory S of what exact time, or during what time interval, scheduled timepoints were executed during previous decisions in the tree. Lastly, the node also keeps track of the activation time intervals of activated uncontrollable timepoints B. The schedule memory S is used to create an updated list of constraints C' resulting from the propagation of the execution time or execution time interval of timepoints in constraints C as described in §4.5. A non-terminal DTNU node, *i.e.* a DTNU node for which all timepoints have not been scheduled, has exactly one child node: a *d-OR* node.

***OR* nodes.** When a choice can be made at time t, this transition control is represented by an *OR* node. We distinguish two types of such nodes, *d-OR* and *w-OR* . For *d-OR* nodes, the first type of choice available is which controllable timepoint a_i to execute. This leads to a DTNU node. The other type of choice is to wait a period of time (§4.1) which leads to a *WAIT* node. *w-OR* nodes can be used for *reactive wait strategies*, *i.e.* to stipulate that some controllable timepoints will be scheduled reactively during waits (§4.3). The parent of a *w-OR* node is therefore a *WAIT* node and its children are *AND* nodes, described below.

***WAIT* nodes.** These nodes are used after a decision to wait a certain period of time Δ_t. The parent of a *WAIT* node is a *d-OR* node. A *WAIT* node has exactly one child: a *w-OR* node, which has the purpose of exploring different reactive wait strategies. The uncertainty management related to uncontrollable timepoints is handled by *AND* nodes.

***AND* nodes.** Such nodes are used after a wait decision is taken and a reactive wait strategy is decided, represented respectively by a *WAIT* and *w-OR* node. Each child node of the *AND* node is a DTNU node at time $t + \Delta_t$, t being the time before the wait and Δ_t the wait duration. Each child node represents an outcome of how uncontrollable timepoints may unfold and is built from the set of *activated* uncontrollable timepoints (uncontrollable timepoints that have been triggered by the execution of their controllable timepoint) whose occurrence time interval overlaps the wait. If there are l activated uncontrollable timepoints, then there are at most 2^l *AND* node children, representing each element of the power set of activated uncontrollable timepoints (§4.1).

Figure 2 illustrates how a sub-problem of Γ, referred to as $DTNU_{O,P,t}$, is developed. Here, $O \subset A$ is the set of controllable timepoints that have already been executed, $P \subset U$ the set of uncontrollable timepoints which have occurred, and t the time. This root node transitions into a *d-OR* node. The *d-OR* node in turn is developed into several children nodes $DTNU_{O\cup\{a_i\},P,t}$ and a *WAIT* node. Each node $DTNU_{O\cup\{a_i\},P,t}$ corresponds to a sub-problem which is obtained from the execution of controllable timepoint a_i at time t. The *WAIT* node refers to the process of waiting a given period of time, Δ_t in the figure, before making the next decision. The *WAIT* node leads directly to a

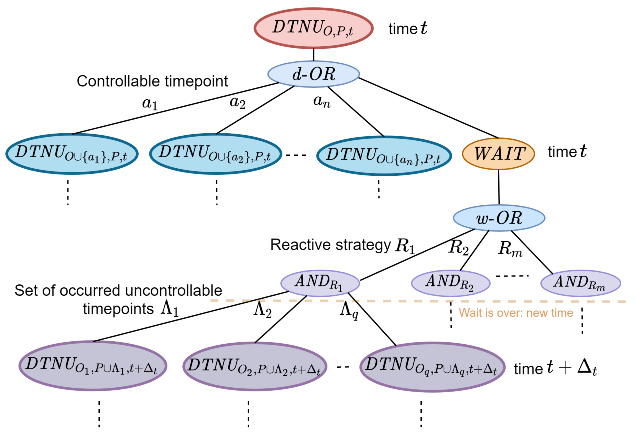

$DTNU_{O,P,t}$
time t
d-OR
Controllable timepoint
a_1
a_2
a_n
$DTNU_{O\cup\{a_1\},P,t}$
$DTNU_{O\cup\{a_2\},P,t}$
$DTNU_{O\cup\{a_n\},P,t}$
$WAIT$
time t
w-OR
Reactive strategy R_1
R_2
R_m
AND_{R_1}
AND_{R_2}
AND_{R_m}
Set of occurred uncontrollable timepoints Λ_1
Λ_2
Λ_q
Wait is over: new time
$DTNU_{O_1,P\cup\Lambda_1,t+\Delta_t}$
$DTNU_{O_2,P\cup\Lambda_2,t+\Delta_t}$
$DTNU_{O_q,P\cup\Lambda_q,t+\Delta_t}$
time $t+\Delta_t$

Figure 2: **Basic structure of the search tree describing how a DTNU node $\boldsymbol{DTNU_{O,P,t}}$ is developed.** $DTNU_{O,P,t}$ (placed at the root of the tree) refers to a DTNU where O is the set of controllable timepoints that have already been executed, P the set of uncontrollable timepoints that have occurred, and t the time. Each branch a_i refers to a controllable timepoint a_i, R_i to a reactive strategy during the wait, and Λ_i to a combination of uncontrollable timepoints which can occur during the wait.

w-OR node which lists different wait strategies R_i. If there are l activated uncontrollable timepoints, there are 2^l subsets of uncontrollable timepoints Λ_i that could occur. Each AND_{R_j} node has one sub-problem DTNU for each Λ_i. Each sub-problem $DTNU_{O_i,P\cup\Lambda_i,t+\Delta_t}$ of the node AND_{R_j} is a DTNU at time $t+\Delta_t$ for which all uncontrollable timepoints in Λ_i are assumed to have happened during the wait period, *i.e.* in the time interval $[t, t + \Delta_t]$. Additionally, some controllable timepoints may have been reactively executed during the wait and may now be included in the set of scheduled controllable timepoints O_i. Otherwise, $O_i = O$.

Two types of leaf nodes exist in the tree. The first type is a node $DTNU_{A,U,t}$ for which all controllable timepoints $a_i \in A$ have been scheduled and all uncontrollable timepoints $u_i \in U$ have occurred. The second type is a node $DTNU_{A\setminus A',U,t}$ for which all uncontrollable timepoints $u_i \in U$ have occurred, but some controllable timepoints $a_i \in A'$ have not been executed. The constraint satisfiability test of the former type of leaf node is straightforward: all execution times of all timepoints are propagated to constraints in the same fashion as in §4.5. The leaf node's truth attribute is set to *true* if all constraints are satisfied, *false* otherwise. For the latter type, we propagate the execution times of all uncontrollable timepoints as well as all scheduled controllable timepoints in the same way, and obtain an updated set of constraint C'. This leaf node, $DTNU_{A\setminus A',U,t}$, is therefore characterized as $\{A', \emptyset, C', \emptyset\}$ and is a DTN. We add the constraints $a'_i \geq t, \forall a'_i \in A'$ and use a mixed integer linear programming solver (Cplex 2009) to solve the DTN. If a solution is found, the execution time values for each $a'_i \in A'$ are stored and the leaf node's truth value is set to *true.* Otherwise, it is set to *false.* After a truth value is assigned to the leaf node, the truth propagation function defined in §4.4 is called to logically infer truth value properties for parent nodes. Lastly, the search algorithm explores the tree in a depth-first manner. At each *d-OR* , *w-OR* and *AND* node, children nodes are visited in the order they are created. Once a child node is selected, its entire subtree will be processed by the algorithm before the other children are explored. Some simplifications made in the exploration are detailed in §11.6 in the appendix.

4 Tree Search Characteristics

4.1 Wait action

When a wait decision of duration Δ_t is taken at time t for a DTNU node, two categories of uncontrollable timepoints are considered to account for all transitional possibilities:

- $Z = \{\zeta_1, \zeta_2, ..., \zeta_l\}$ is a set of timepoints that could either happen during the wait, or afterwards, *i.e.* the end of the activation time interval for each ζ_i is greater than $t + \Delta_t$.
- $H = \{\eta_1, \eta_2, ..., \eta_m\}$ is a set of timepoints that are certain to happen during the wait, *i.e.* the end of the activation time interval for each η_i is less than or equal to $t + \Delta_t$.

There are $q = 2^l$ number of different possible combinations (empty set included) $\Upsilon_1, \Upsilon_2, ..., \Upsilon_q$ of elements taken from Z. For each combination Υ_i, the set $\Lambda_i = H \cup \Upsilon_i$ is created. The union $\bigcup\limits_{i=1}^{q} \Lambda_i$ refers to all possible combinations of uncontrollable timepoints which can occur by $t + \Delta_t$. In Figure 2, for each *AND* node, the combination Λ_i leads to a DTNU sub-problem $DTNU_{O_i, P\cup\Lambda_i, t+\Delta_t}$ for which the uncontrollable timepoints in Λ_i are considered to have occurred between t and $t + \Delta_t$ in the schedule memory S. In addition, any potential controllable timepoint ϕ planned to be instantly scheduled in a reactive wait strategy R_i in response to an uncontrollable timepoint u in Λ_i will also be considered to have been scheduled between t and $t + \Delta_t$ in S. The only exception is when checking constraint satisfiability for the conjunct $u - \phi \in [0, y]$ which required the reactive scheduling, for which we assume ϕ executed at the same time as u, thus the conjunct is considered satisfied.

4.2 Wait Eligibility and Period

The way time is discretized holds direct implications on the search space explored and the capability of the algorithm to find TDC strategies. Longer waits make the search space smaller, but carry the risk of missing key moments where a decision is needed. On the other hand, smaller waits can make the search space too large to explore. We explain when the wait action is eligible, and how its duration is computed.

Eligibility At least one of these two criteria has to be met for a *WAIT* node to be added as child of a *d-OR* node. **(1)** There is at least one activated uncontrollable timepoint for the parent DTNU node. **(2)** There is at least one conjunct of the form $v \in [x, y]$, where v is a timepoint, in the constraints of the parent DTNU node. These criteria ensure that the search tree will not develop branches below *WAIT* nodes when waiting is not relevant, *i.e.* when a controllable timepoint necessarily needs to be scheduled. It also prevents the tree search from getting stuck in infinite *WAIT* loop cycles.

Wait Period We define the wait duration Δ_t at a given *d-OR* node eligible for a wait dynamically by examining the updated constraint list C' of the parent DTNU and the activation time intervals B of its activated uncontrollable timepoints. Let t be the current time for this DTNU node. The wait duration is defined by comparing t to elements in C' and B to look for a minimum positive value defined by the following three rules. **(1)** For each activated time interval $u \in [x, y]$ in B, we select $x - t$ or $y - t$, whichever is smaller and positive, and we keep the smallest value δ_1 found over all activated time intervals. **(2)** For each conjunct $v \in [x, y]$ in C', where v is a timepoint, we select $x - t$ or $y - t$, whichever is smaller and positive, and we keep the smallest value δ_2 found over all conjuncts. **(3)** We determine timepoints which need to be scheduled ahead of time by chaining constraints together. Intuitively, when a conjunct $v \in [x, y]$ is in C', it means v has to be executed when $t \in [x, y]$ to satisfy this conjunct. However, v could be linked to other timepoints by constraints which require them to happen before v. These timepoints could in turn be linked to yet other timepoints in the same way, and so on. The third rule consists in chaining backwards to identify potential timepoints which start this chain and potential time intervals in which they need to be executed. The following mechanism is used: for each conjunct $v \in [x, y]$ in C' found in (2), we apply a recursive backward chain function to both (v, x) and (v, y). We detail here how it is applied to (v, x), the process being the same for (v, y). Conjuncts of the form $v - v' \in [x', y'], x' \geq 0$ in C' are searched for. For each conjunct found, we add to a list two elements, $(v', x - x')$ and $(v', x - y')$. We select $x - x' - t$ or $x - y' - t$, whichever is smaller and positive, as potential minimum candidate. The backward chain function is called recursively on each element of the list, proceeding the same way. We keep the smallest candidate δ_3. Figure 8 in the appendix illustrates an application of this process. Finally, we set $\Delta_t = \min(\delta_1, \delta_2, \delta_3)$ as the wait duration. This duration is stored inside the *WAIT* node.

4.3 Reactive scheduling during waits

Execution of a controllable timepoint may be necessary in some situations at the exact same time as when an uncontrollable timepoint occurs to satisfy a constraint. Therefore, different reactive wait strategies are considered and listed as children of a *w-OR* node after a wait decision, before the actual start of the wait itself. We designate as a *conjunct* a constraint relationship of the form $v_i - v_j \in [x, y]$ or $v_i \in [x, y]$, where v_i, v_j are timepoints and $x, y, \in \mathbb{R}$. We refer to a constraint where several conjuncts are linked by $\vee$ operators as a *disjunct*. If at any given DTNU node in the tree there is an activated uncontrollable timepoint u with the potential to occur during the next wait and there is at least one unscheduled controllable timepoint a such that a conjunct of the form $u - a \in [0, y], y \geq 0$ is present in the constraints, a reactive wait strategy is available that will schedule a as soon as u occurs.

If there are s controllable timepoints that may be reactively scheduled, there are 2^s different reactive wait strategies R_i, each of which is embedded in an *AND* child of the *w-OR* node. Let $\Phi = \{\phi_1, \phi_2, ..., \phi_s\} \subset A$ be the complete set of unscheduled controllable timepoints for which there are conjunct clauses $u - \phi_i \in [0, y]$. We denote as $R_1, R_2, ..., R_m$ all possible combinations of elements taken from Φ, including the empty set. The child node AND_{R_i} of the *w-OR* node resulting from the combination R_i has a reactive wait strategy for which all controllable timepoints in R_i will be immediately executed at the moment u occurs during the wait, if it does. If u doesn't occur, no controllable timepoint is reactively scheduled during the wait.

4.4 Truth Value Propagation

In this section, we describe how truth attributes of nodes are related to each other. The truth attribute of a tree node represents its TDC controllability, and the relationships shared between nodes make it possible to define sound strategies. When a leaf node is assigned a truth attribute β, the tree search is momentarily stopped and β is propagated onto upper parent nodes. To this end, a parent node $\boldsymbol{\omega}$ is selected recursively and we distinguish the following cases:

- The parent $\boldsymbol{\omega}$ is a DTNU or *WAIT* node: $\boldsymbol{\omega}$ is assigned β.
- The parent $\boldsymbol{\omega}$ is a *d-OR* or *w-OR* node: If $\beta = true$, then $\boldsymbol{\omega}$ is assigned *true* . If $\beta = false$ and all children nodes of $\boldsymbol{\omega}$ have *false* attributes, $\boldsymbol{\omega}$ is assigned *false* . Otherwise, the propagation stops.
- The parent $\boldsymbol{\omega}$ is an *AND* node: If $\beta = false$, then $\boldsymbol{\omega}$ is assigned *false* . If $\beta = true$ and all children nodes of $\boldsymbol{\omega}$ have *true* attributes, $\boldsymbol{\omega}$ is assigned *true* . Otherwise, the propagation stops.

After the propagation finishes, the tree search algorithm resumes where it was stopped. A *true* attribute reaching the root node of the tree means a TDC strategy has been found. A *false* attribute means none could be found. The pseudocode for the propagation algorithm is given in Algorithm 1 in the appendix.

4.5 Constraint Propagation

Decisions taken in the tree define when controllable timepoints are executed and also bear consequences on the execution time of uncontrollable timepoints. We explain here how these decisions are propagated into constraints, as well as the concept of '*tight bound*'. Let C' be the list of updated constraints for a DTNU node $\boldsymbol{\psi}$ for which the parent node is $\boldsymbol{\omega}$. We distinguish two cases. Either $\boldsymbol{\omega}$ is a *d-OR* node and $\boldsymbol{\psi}$ results from the execution of a controllable timepoint a_i, or $\boldsymbol{\omega}$ is an *AND* node and $\boldsymbol{\psi}$ results from a wait of Δ_t time units. In the first case, let t be the execution time of a_i. The updated list C' is built from the constraints of the parent DTNU of $\boldsymbol{\psi}$ in the tree. If a conjunct contains a_i and is of the form $a_i \in [x, y]$, this conjunct is replaced with *true* if $t \in [x, y]$, *false* otherwise. If the conjunct is of the form $v_j - a_i \in [x, y]$, we replace the conjunct with $v_j \in [t + x, t + y]$. The other possibility is that $\boldsymbol{\psi}$ results from a wait of Δ_t time at time t, with a reactive wait strategy R_j. In this case, the new time is $t + \Delta_t$ for $\boldsymbol{\psi}$. As a result of the wait, some uncontrollable timepoints $u_i \in \Lambda_i$ are assumed to have occured, and some controllable timepoints $a_i \in R_j$ may be executed reactively during the wait. Let $v_i \in \Lambda_i \cup R_j$ be these timepoints occurring during the wait. The execution time of these timepoints is assumed to be in $[t,\ t + \Delta_t]$. For uncontrollable timepoints $u'_i \in \Lambda'_i \subset \Lambda_i$ for which the activation time ends at $t + \Delta'_{t_i} < t + \Delta_t$, and potential controllable timepoints a'_i instantly reacting to these uncontrollable timepoints, the execution time is further reduced and considered to be in $[t, t + \Delta'_{t_i}]$. We define a concept of *tight bound* to update constraints which restricts time intervals in order to account for all possible values v_i can take between t and $t + \Delta_t$. For all conjuncts $v_j - v_i \in [x, y]$, we replace the conjunct with $v_j \in [t + \Delta_t + x, t + y]$. Intuitively, this means that since v_i can happen at the latest at $t + \Delta_t$, v_j can not be allowed to happen before $t + \Delta_t + x$. Likewise, since v_i can happen at the earliest at t, v_j can not be allowed to happen after $t + y$. Finally, if $t + \Delta_t + x > t + y$, the conjunct is replaced with *false* . Also, the process can be applied recursively in the event that v_j is also a timepoint that occurred during the wait, in which case the conjunct would be replaced by *true* or *false*. In any case, any conjunct obtained of the form $a_j \in [x', y']$ is replaced with *false* if $t + \Delta_t > y'$. Finally, if all conjuncts inside a disjunct are set to *false* by this process, the constraint is violated and the DTNU is no longer satisfiable.

5 Learning-based Heuristic

We present here our learning model and explain how it provides tree search heuristic guidance. Our learning architecture originates from (Gilmer et al. 2017). It uses message passing rules allowing neural networks to process graph-structured inputs where both vertices and edges possess features. This architecture was originally designed for node classification in quantum chemistry and achieved state-of-the-art results on a molecular property prediction benchmark. Here, we first define a way of converting DTNUs into graph data. Then, we process the graph data with our MPNN and use the output to guide the tree search.

Let $\Gamma = \{A, U, C, L\}$ be a DTNU. We explain how we turn Γ into a graph $\mathcal{G} = (\mathcal{K}, \mathcal{E})$. First, we convert all time values from absolute to relative with the assumption the current time for Γ is $t = 0$. We search all converted time intervals $[x_i, y_i]$ in C and L for the highest interval bound value d_{max}, *i.e.* the farthest point in time. We proceed to normalize every time value in C and L by dividing them by d_{max}. As a result, every time value becomes a real number between 0 and 1. Next, we convert each controllable timepoint $a \in A$ and uncontrollable timepoint $u \in U$ into graph nodes with corresponding *controllable* or *uncontrollable* node features. The time constraints in C and contingency links in L are expressed as edges between nodes with 10 different edge distance classes ($0 : [0, 0.1)$, $1 : [0.1, 0.2)$, ..., $9 : [0.9, 1]$). We also use additional edge features to account for edge types (constraint, disjunction, contingency link, direction sign for lower and upper bounds). Moreover, intermediary nodes are used with a distinct node feature in order to map possible disjunctions in constraints and contingency links. We add a *WAIT* node with a distinct node feature which implicitly designates the act of waiting a period of time. The graph conversion of DTNU γ is characterized by three elements:

the matrix of all node features X_κ, the adjacency matrix of the graph X_ϵ and the matrix of all edge features X_ρ.

These features are processed by several consecutive *message passing* layers from (Gilmer et al. 2017). Each layer takes an input graph, consists of a phase during which messages are passed between nodes, and returns the same graph with new node features. The overall process for a layer is the following. For each node κ_i in the input graph, a *feature aggregation* phase is applied and creates new features for κ_i from current features of neighboring nodes and edges. In detail, for each neighbor node κ_j, a small neural network (termed multi-layer perceptron, or MLP) takes as input the features of the edge connecting κ_i and κ_j and returns a matrix which is then multiplied by the features of κ_j to obtain a feature vector. The sum of these vectors for the entire neighborhood defines the new features for κ_i. The output of the message passing layer consists of the graph updated with the new node features. The feature aggregation process being the same for any node, it can be applied to input graphs of any size, *i.e.* it enables our MPNN architecture to take as input DTNUs of any size. Moreover, each message passing layer contains a different MLP and can thus be trained to learn a different feature aggregation scheme.

Let f be the mathematical function for our MPNN and θ its set of parameters. Our function f stacks 5 message passing layers coupled with the $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$ piecewise activation function (Glorot, Bordes, and Bengio 2011). The `sigmoid` function $\sigma(\cdot) = \frac{1}{1+\exp(-\cdot)}$ is then used to obtain a list of probabilities π over all nodes in $\mathcal{G}$: $f_\theta(X_\kappa, X_\epsilon, X_\rho) = \pi$. The probability of each node κ in π corresponds to the likelihood of transitioning into a TDC DTNU from the original DTNU Γ by taking the action corresponding to κ. If κ represents a controllable timepoint a in Γ, its corresponding probability in π is the likelihood of the sub-DTNU resulting from the execution of a being TDC. If κ represents a *WAIT* decision, its probability refers to the likelihood of the *WAIT* node having a *true* attribute, *i.e.* the likelihood of all children DTNUs resulting from the wait being TDC (with the wait duration rules set in §4.2). We call these two types of nodes *active* nodes. Otherwise, if κ is another type of node, its probability is not relevant to the problem and ignored. Our MPNN is trained on DTNUs generated and solved in §6 only on active nodes by minimizing the cross-entropy loss:

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{q} -Y_{ij}\log(f_\theta(X_i)_j) - (1-Y_{ij})\log(1-f_\theta(X_i)_j)$$

Here $X_i = (X_{i_\kappa}, X_{i_\epsilon}, X_{i_\rho})$ is DTNU number i among a training set of m examples, Y_{ij} is the TDC controllability (1 or 0) of active node number j for DTNU number i.

Lastly, the MPNN heuristic is used in the following way in the tree search. Once a *d-OR* node is reached, the parent DTNU node is converted into a graph and the MPNN f is called upon the corresponding graph elements $X_\kappa, X_\epsilon, X_\rho$. Active nodes in output probabilities π are then ordered by highest values first, and the tree search visits the corresponding children tree nodes in the suggested order, preferring children with higher likelihood of being TDC first.

6 Randomized Simulations for Heuristic Training

We leverage a learning-based heuristic to guide the tree search. A key component in learning-based methods is the annotated training data. We generate such data in automatic manner by using a DTNU generator to create random DTNU problems and solving them with a modified version of the tree search. We store results and use them for training the MPNN. We detail now our data generation strategy.

We create DTNUs with a number of controllable timepoints ranging from 10 to 20 and uncontrollable timepoints ranging from 1 to 3. The generation process is the following. For interval bounds of constraint conjuncts or contingency links, we randomly generate real numbers within $[0, 100]$. We restrict the number of conjuncts inside a disjunct to 5 at most. A random number $n_1 \in [10, 20]$ of controllable timepoints and $n_2 \in [1, 3]$ of uncontrollable timepoints are selected. Each uncontrollable timepoint is randomly linked to a different controllable timepoint with a contingency link. Next, we iterate over the list of timepoints, and for each timepoint v_i not appearing in constraints or contingency links, we add in the constraints a disjunct for which at least one conjunct constrains v_i. The type of conjunct is selected randomly from either a *distance* conjunct $v_i - v_j \in [x, y]$ or a *bounded* conjunct $v_i \in [x, y]$. On the other hand, if v_i was already present in the constraints or contingency links, we add a disjunct constraining v_i with only a 20% probability.

In order to solve these DTNUs, we modify the tree search as follows. For a DTNU Γ, the first *d-OR* child node is developed as well as its children $\psi_1, \psi_2, ..., \psi_n \in \Psi$. The modified tree search explores each ψ_i multiple times (ν times at most), each time with a timeout of τ seconds. We set $\nu = 25$ and $\tau = 3$. For each exploration of ψ_i, children nodes of any *d-OR* node encountered in the corresponding subtree are explored randomly each time. If ψ_i is proved to be either TDC or non-TDC during an exploration, the next explorations of the same child ψ_i are called off and the truth attribute β_i of ψ_i is updated accordingly. The active node number k, corresponding to the decision leading to ψ_i from DTNU Γ's *d-OR* node, is updated with the same value, *i.e.* $Y_k = \beta_i$ (1 for *true*, 0 for *false*). If every exploration times out, ψ_i is assumed non-TDC and Y_k is set to *false*. Once each ψ_i has been explored, the pair $\langle G(\Gamma), (Y_1, Y_2, ..., Y_n)\rangle$ is stored in the training set, where $G(\Gamma)$ is the graph conversion of Γ described in §5. Data related to solved sub-DTNUs of Γ are not stored in the training set as it was found to cause bias issues and overall decrease generalization in MPNN predictions.

The assumption of non-TDC controllability for children nodes for which all explorations time out is acceptable in the sense that the heuristic used is not admissible and does not need to be. The output of the MPNN is a probability for each child node of the *d-OR* node, creating a preferential order of visit by highest probabilities first. Even in the event the suggested order first recommends visiting children nodes which will be found to be non-TDC, the algorithm will continue to explore the remaining children nodes until one is found to be TDC. Nevertheless, such a scenario will rarely occur as the trained MPNN will give higher probabilities for children nodes for which explorations would tend to find a TDC strategy before timeout, and lower probabilities for ones where explorations would tend to result in a timeout.

7 Strategy Execution

A strategy found by the tree search for a DTNU Γ is sound and guarantees constraint satisfiability if executed in the following manner. Let $\mathcal{Q}$ be the system interacting with the environment, executing controllable timepoints and observing how uncontrollable timepoints unfold. At each DTNU node in the tree, $\mathcal{Q}$ moves on to the child *d-OR* node. The child node ψ_i of the *d-OR* node which was found by the strategy to have a *true* attribute is selected. If ψ_i is a DTNU node, $\mathcal{Q}$ executes the corresponding controllable timepoint a_i and moves on to ψ_i. If ψ_i is a *WAIT* node, $\mathcal{Q}$ moves on to ψ_i, reads the wait duration Δ_t stored in ψ_i and moves on to the child *w-OR* node. The child node AND_{R_j} of the *w-OR* node which has a *true* attribute is selected, and $\mathcal{Q}$ will wait Δ_t time units with the reactive wait strategy R_j. After the wait is over, $\mathcal{Q}$ observes the list of all uncontrollable timepoints Λ_i which occurred, deduces which DTNU child node of the AND_{R_j} node it transitioned into, and moves on to that node.

By following these guidelines, the final tree node $\mathcal{Q}$ transitions into is necessarily a leaf node with a *true* attribute, *i.e.* a node for which all constraints are satisfied. This is due to the fact that for *d-OR* and *w-OR* nodes $\mathcal{Q}$ visits, $\mathcal{Q}$ chooses to transition into a child node with a *true* attribute. For *AND* nodes $\mathcal{Q}$ visits, all children DTNU nodes have a *true* attribute, so $\mathcal{Q}$ transitions into a child node with a *true* attribute regardless of how uncontrollable timepoints unfold.

8 Experiments

We evaluate experimentally the efficiency of the tree search approach and the effect of the MPNN's guidance. We also compare them with a DC solver from (Cimatti, Micheli, and Roveri 2016). TDC is a subset of DC and a more restrictive form of controllability: non-TDC controllability does not imply non-DC controllability. A TDC solver can thus be expected to offer better performance than a DC one while potentially being unable to find a strategy when a DC algorithm would. In this section, we refer to the tree search algorithm as TS, the tree search algorithm guided by the trained MPNN up to the 15^{th} (respectively X^{th}) *d-OR* node depthwise in the tree as MPNN-TS (respectively MPNN-TS-X) and the most efficient DC solver from (Cimatti, Micheli, and Roveri 2016) as PYDC-SMT ordered.

First, we use the benchmark in the experiments of (Cimatti, Micheli, and Roveri 2016) from which we remove DTNs and STNs. We compare TS, MPNN-TS and PYDC-SMT on the resulting benchmark which is comprised of 290 DTNUs and 1042 STNUs. Here, Limiting the maximum depth use of the MPNN to 15 offers a good trade off between guidance gain and cost of calling the heuristic. Results are given in Figure 3. We observe that TS solves roughly 50% more problem instances than PYDC-SMT within the allocated time (20 seconds). In addition, TS solves 56% of all instances while the remaining ones time out. Among solved instances, a strategy is found for 89% and the remaining

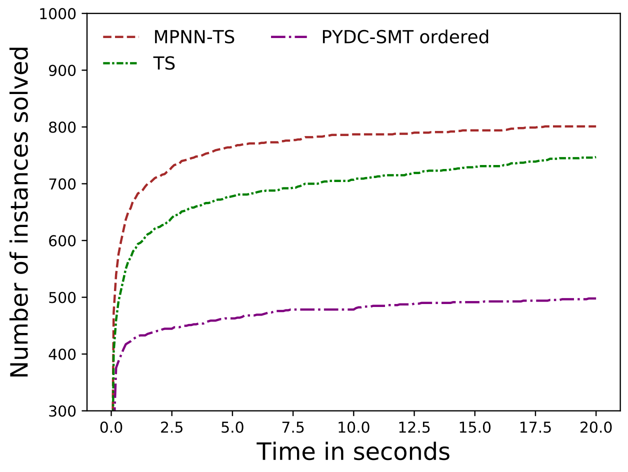


Figure 3: **Experiments on (Cimatti, Micheli, and Roveri 2016)'s benchmark from which DTNs and STNs have been removed.** The X-axis represents the allocated time in seconds and the Y-axis the number of instances in the benchmark each solver can solve within the corresponding allocated time. Timeout is set to 20 seconds per instance.

11% are proved non-TDC. On the other hand, PYDC-SMT solves 37% of all instances. A strategy is found for 85% of PYDC-SMT's solved instances while the remaining 15% are proved non-DC. Finally, out of all instances PYDC-SMT solves, TS solves 97% accurately with the same conclusion, *i.e.* TDC when DC and non-TDC when non-DC. The use of the heuristic leads to an additional +6% problems solved within the allocated time. We argue this small increase is essentially due to the fact that most problems solved in the benchmark are small-sized problems with few timepoints which are solved quickly. Despite this fact, the heuristic still provides performance boost on a benchmark generated with another DTNU generator, suggesting the bias introduced by our DTNU generator remains limited and the MPNN is able to generalize to DTNUs created with a different approach.

For further evaluation of the heuristic, we create new benchmarks using the DTNU generator from §6 with varying number of timepoints. These benchmarks have fewer quick to solve DTNUs and harder ones instead. Each benchmark contains 500 randomly generated DTNUs which have 1 to 3 uncontrollable timepoints. Moreover, each DTNU has 10 to 20 controllable timepoints in the first benchmark B_1, 20 to 25 in the second benchmark B_2 and 25 to 30 in the last benchmark B_3. Each disjunct in the constraints of any DTNU contains up to 5 conjuncts. Experiments on B_1, B_2 and B_3 are respectively shown in Figure 4, 6c (in the appendix) and 5. We note that for all three benchmarks no solver ever proves non-TDC or non-DC controllability before timing out due to the larger size of these problems.

PYDC-SMT performs poorly on B_1 and cannot solve any instance on B_2 and B_3. TS does not perform well on B_2 and only solves 2 instances on B_3. However, we see a significantly higher gain from the use of the MPNN, varying with the maximum depth use. At best depth use, the gain is

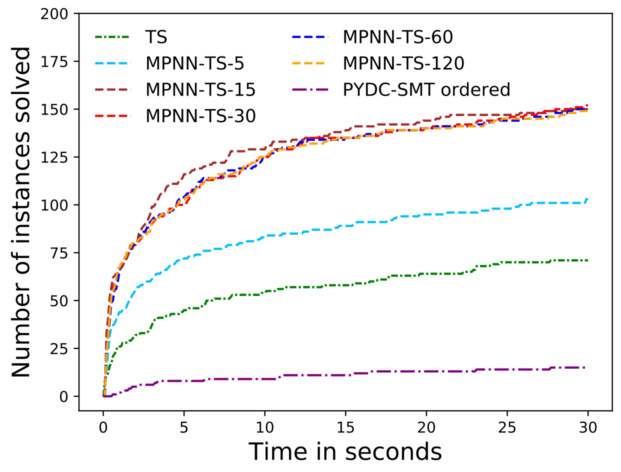


Figure 4: **Experiments on benchmark $\boldsymbol{B_1}$.** Axes are the same as in Figure 3. Timeout is set to 30 seconds per instance.

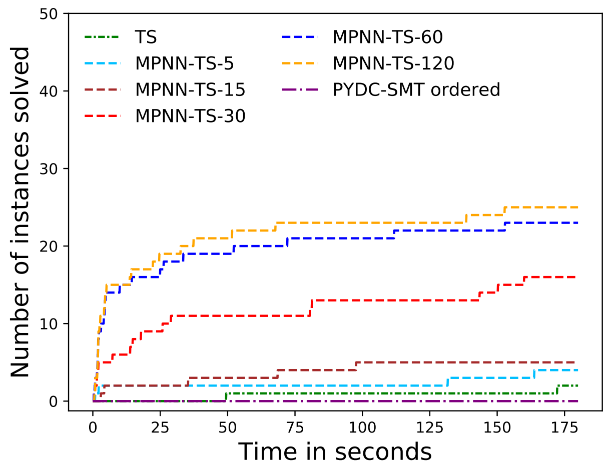


Figure 5: **Experiments on benchmark $\boldsymbol{B_3}$.** Axes are the same as in Figure 3. Timeout is set to 180 seconds.

$+91\%$ instances solved for B_1, $+980\%$ for B_2 and $+1150\%$ for B_3. The more timepoints instances have, the more worthwhile heuristic guidance appears to be. Indeed, the optimal maximum depth use of the MPNN in the tree increases with the problem size: 15 for B_1, 60 for B_2 and 120 for B_3. We argue this is due to the fact that more timepoints results in a wider search tree overall, including in deeper sections where heuristic use was not necessarily worth its cost for smaller problems. Furthermore, the MPNN is trained on randomly generated DTNUs which have 10 to 20 controllable timepoints. The promising gains shown by experiments on B_2 and B_3 suggest generalization of the MPNN to bigger problems than it is trained on.

The proposed tree search approach presents a good trade off between search completeness and effectiveness: almost all examples solved by PYDC-SMT from (Cimatti, Micheli, and Roveri 2016)'s benchmark are solved with the same conclusion, and many more which could not be solved are. Moreover, the TDC approach scales up better to problems with more timepoints, and the tree structure allows the use of learning-based heuristics. Although these heuristics are not key to solving problems of big scales, our experiments suggest they can still provide a high increase in efficiency.

9 Related Work

Learning-based heuristics have become increasingly popular for planning, combinatorial and network modeling problems. Recent works applied to network modeling and routing problems include (Rusek et al. 2019), (Chen et al. 2018), (Xu et al. 2018), (Kool and Welling 2018). Recently, GNNs have become a popular extension of CNNs. Essentially, their ability to represent problems with a graph structure and the resulting node permutation invariance makes them convenient for some applications. We refer the reader to (Wu et al. 2019) for a complete survey on GNNs. In combinatorial optimization, GNNs can benefit both approximate and exact solvers. In (Li, Chen, and Koltun 2018), authors combine tree search, GNNs and a local search algorithm to achieve state-of-the-art results for approximate solving of NP-hard problems such as the maximum independent set problem. On the other hand, (Gasse et al. 2019) use a GNN for branch and bound variable selection for exact solving of NP-hard problems and achieve superior results to previous learning approaches. In path-planning problems with NP-hard constraints, (Osanlou et al. 2019) use a GNN to predict an upper bound for a branch and bound solver and outperform an A*-based planner coupled with a problem-suited handcrafted heuristic. (Ma et al. 2018) leverage a GNN for the selection of a planner inside a portfolio for STRIPS planning problems and outperform the previous leading learning-based approach based on a CNN (Sievers et al. 2019). In most works, GNNs seem to offer generalization to bigger problems than they are trained on. Results from our experiments are in line with this observation.

10 Conclusion

We introduced a new type of controllability, time-based dynamic controllability (TDC), and a tree search approach for solving disjunctive temporal networks with uncertainty (DTNU) in TDC. Strategies are built by discretizing time and exploring different decisions which can be taken at different key points, as well as anticipating how uncontrollable timepoints can unfold. We defined constraint propagation rules which ensure soundness of strategies found. We showed that the tree search approach is able to solve DTNUs in TDC more efficiently than the state-of-the-art dynamic controllability (DC) solver, PYDC-SMT, with almost always the same conclusion. Lastly, we created MPNN-TS, a solver which combines the tree search with a heuristic function based on message passing neural networks (MPNN) for guidance. The MPNN is trained with a self-supervised strategy and enables steady improvements of the tree search on harder DTNU problems, notably on DTNUs of bigger size than those used for training the MPNN.

References

Battaglia, P.; Pascanu, R.; Lai, M.; Rezende, D. J.; et al. 2016. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, 4502–4510.

Bhargava, N., and Williams, B. C. 2019. Complexity bounds for the controllability of temporal networks with conditions, disjunctions, and uncertainty. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 6353 – 6357.

Chen, X.; Guo, J.; Zhu, Z.; Proietti, R.; Castro, A.; and Yoo, S. 2018. Deep-rmsa: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks. In *2018 Optical Fiber Communications Conference and Exposition (OFC)*, 1–3. IEEE.

Cimatti, A.; Micheli, A.; and Roveri, M. 2016. Dynamic controllability of disjunctive temporal networks: Validation and synthesis of executable strategies. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Cplex, I. I. 2009. V12. 1: User's manual for cplex. *International Business Machines Corporation* 46(53):157.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 3844–3852.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12(Jul):2121–2159.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, 15554–15566.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 1263–1272. JMLR. org.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 315–323.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Kipf, T.; Fetaya, E.; Wang, K.-C.; Welling, M.; and Zemel, R. 2018. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*.

Kool, W., and Welling, M. 2018. Attention solves your tsp. *arXiv preprint arXiv:1803.08475*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. S. 2016. Gated graph sequence neural networks. In *International Conference on Learning Representations*.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, 536—-545.

Ma, T.; Ferber, P.; Huo, S.; Chen, J.; and Katz, M. 2018. Adaptive planner scheduling with graph neural networks. *CoRR* abs/1811.00210.

Morris, P., and Muscettola, N. 2005. Temporal dynamic controllability revisited. In *Proceedings of the* 22^{nd} *National Conference on Artificial Intelligence*.

Morris, P. 2014. Dynamic controllability and dispatchability relationships. In *Proceedings of the IInternational Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 464 – 479.

Osanlou, K.; Bursuc, A.; Guettier, C.; Cazenave, T.; and Jacopin, E. 2019. Optimal solving of constrained path-planning problems with graph convolutional networks and optimized tree search. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3519–3525. IEEE.

Rusek, K.; Suárez-Varela, J.; Mestres, A.; Barlet-Ros, P.; and Cabellos-Aparicio, A. 2019. Unveiling the potential of graph neural networks for network modeling and optimization in sdn. In *Proceedings of the 2019 ACM Symposium on SDN Research*, 140–151.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019. Deep learning for cost-optimal planning: Task-dependent planner selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7715–7723.

Tsamardinos, I. 2002. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence*, 97 – 108.

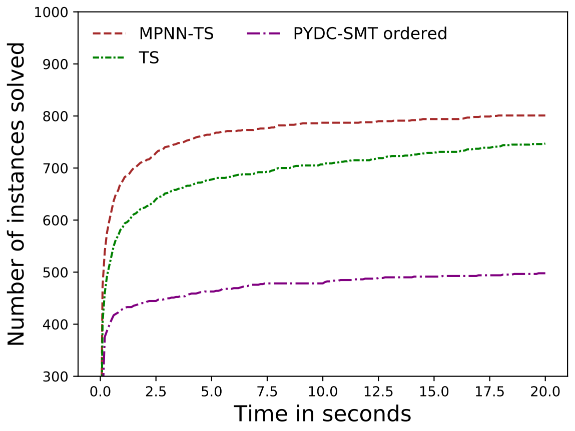
Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence* 11(1):23 – 45.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2019. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

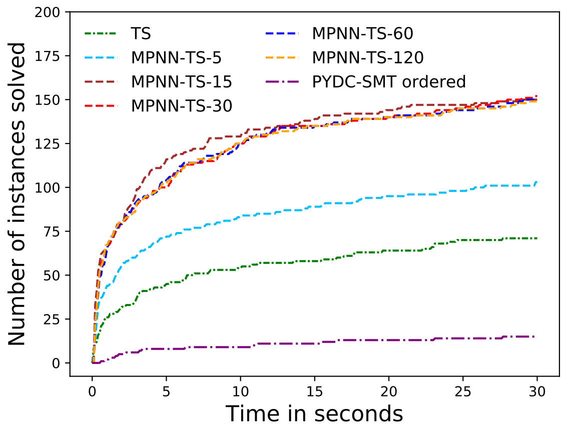
Xu, Z.; Tang, J.; Meng, J.; Zhang, W.; Wang, Y.; Liu, C. H.; and Yang, D. 2018. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 1871–1879. IEEE.

11 Appendix

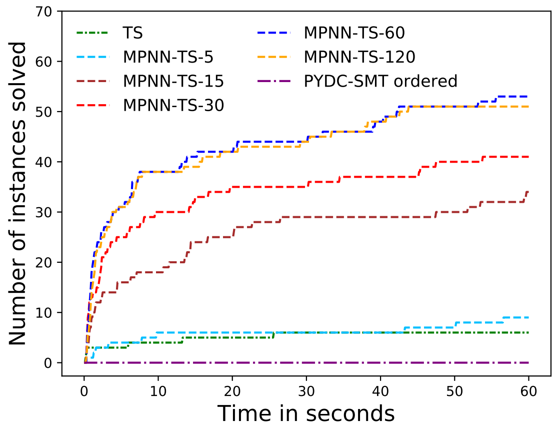
11.1 Plots



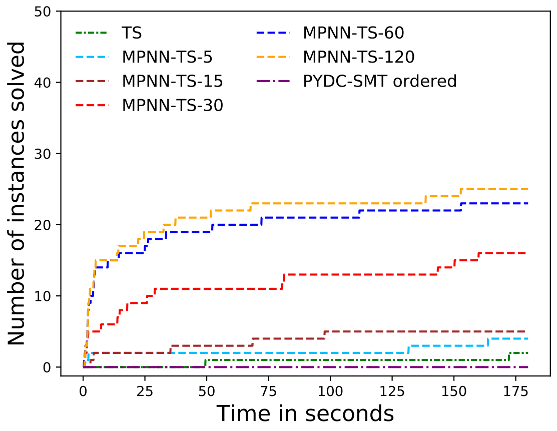
(a) Experiments on (Cimatti, Micheli, and Roveri 2016)'s benchmark from which the DTNs and STNs have been removed. The X-axis represents the allocated time in seconds and the Y-axis the total number of instances that each solver can solve within the corresponding allocated time. Timeout is set to 20 seconds per instance.



(b) Experiments on benchmark B_1. Axes are the same as in figure 6a. Timeout is set to 30 seconds per instance.



(c) Experiments on benchmark B_2. Axes are the same as in figure 6a. Timeout is set to 60 seconds per instance.



(d) Experiments on benchmark B_3. Axes are the same as in figure 6a. Timeout is set to 180 seconds per instance.

Figure 6: **Summary of experiments on benchmarks**

11.2 Simplified Example

Figure 7 is a simplified example of a TDC strategy of the example DTNU from (Cimatti, Micheli, and Roveri 2016).

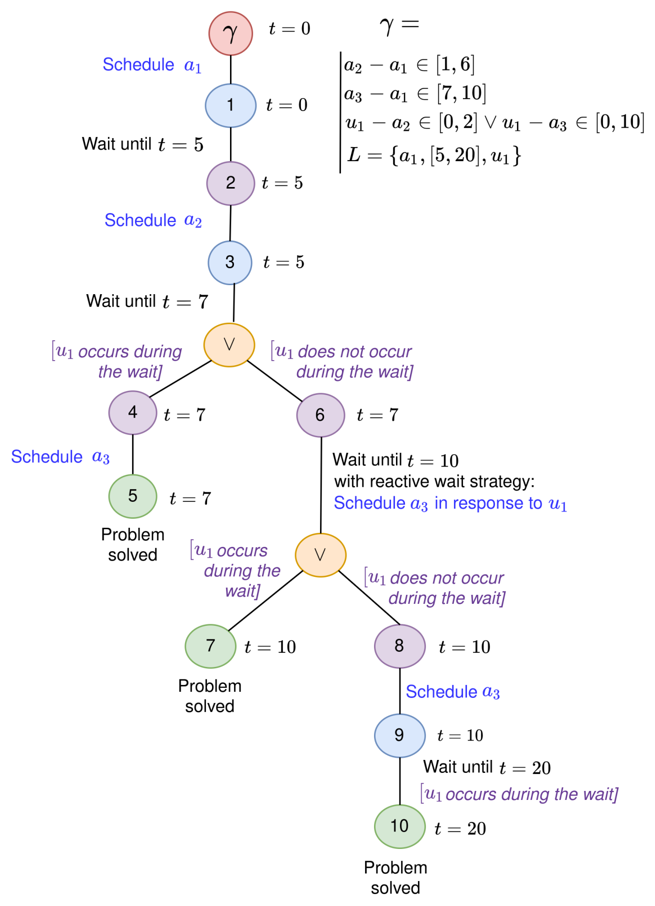

γ
$t = 0$
$\gamma =$
$a_2 - a_1 \in [1, 6]$
$a_3 - a_1 \in [7, 10]$
$u_1 - a_2 \in [0, 2] \vee u_1 - a_3 \in [0, 10]$
$L = \{a_1, [5, 20], u_1\}$
Schedule a_1
1
$t = 0$
Wait until $t = 5$
2
$t = 5$
Schedule a_2
3
$t = 5$
Wait until $t = 7$
$\vee$
[u_1 occurs during the wait]
[u_1 does not occur during the wait]
4
$t = 7$
6
$t = 7$
Schedule a_3
5
$t = 7$
Problem solved
Wait until $t = 10$ with reactive wait strategy: Schedule a_3 in response to u_1
$\vee$
[u_1 occurs during the wait]
[u_1 does not occur during the wait]
7
$t = 10$
Problem solved
8
$t = 10$
Schedule a_3
9
$t = 10$
Wait until $t = 20$
[u_1 occurs during the wait]
10
$t = 20$
Problem solved

Figure 7: **Simplified TDC strategy of a DTNU Γ.** For space reasons, we only give a summarized copy of the strategy found. Branches leading to unsolved cases are excluded, and we do not include *d-OR* , *w-OR* , *AND* and *WAIT* nodes. The node γ is the original DTNU. Other nodes are sub-DTNUs, except the $\vee$ node which aims to list transitional possibilities, and should be interpreted in the figure as an *AND* node.

11.3 Truth Value Propagation Algorithm

We present in this section Algorithm 1. This algorithm is called to propagate a truth value in the tree. The propagation is done in an ascending way: truth values are inferred from the leaves of the tree towards the root.

Algorithm 1 Truth Value Propagation

```
1:  function PROPAGATETRUTH(TREENODE $\psi$)
2:      $\omega \leftarrow$ parent($\psi$)                                   ▷ $^1*$
3:      if $\omega = null$ then
4:          return
5:      if isDTNU($\omega$) or isWAIT($\omega$) then                        ▷ $^2*$
6:          $\omega.truth \leftarrow \psi.truth$
7:          propagateTruth($\omega$)
8:      else if isOR($\omega$) then                                      ▷ $^3*$
9:          if $\psi.truth = True$ then
10:             $\omega.truth \leftarrow True$
11:             propagateTruth($\omega$)
12:         else
13:             if $\forall \sigma_i, \sigma_i.truth = False$ then               ▷ $^4*$
14:                 $\omega.truth \leftarrow False$
15:                 propagateTruth($\omega$)
16:     else if isAND($\omega$) then                                     ▷ $^5*$
17:         if $\psi.truth = False$ then
18:             $\omega.truth \leftarrow False$
19:             propagateTruth($\omega$)
20:         else
21:             if $\forall \sigma_i, \sigma_i.truth = True$ then                ▷ $^4*$
22:                 $\omega.truth \leftarrow True$
23:                 propagateTruth($\omega$)
```

^{1*} parent(x): Returns the parent node of x, $null$ if none.
^{2*} isDTNU(x): Returns *True* if x is a DTNU node, *False* otherwise; isWait(x): Returns *True* if x is a *WAIT* node, *False* otherwise.
^{3*} isOR(x): Returns *True* if x is an *d-OR* or *w-OR* node, *False* otherwise.
^{4*} σ_i: Child number i of ω. For a *d-OR* or *w-OR* node, in the case where ψ is *false* but not all other children of ω are *false* the propagation stops. Likewise, for an *AND* node and in the case where ψ is *true* but not all other children of ω are *true* , the propagation stops.
^{5*} isAnd(x): Returns *True* if x is an *AND* node, *False* otherwise.

11.4 Tree Search Algorithm

We give the simplified pseudocode for the tree search in Algorithm 2.

Algorithm 2 Tree Search

```
1:  function EXPLORE(TREENODE $\psi$)
2:      if $parent(\psi).truth \neq unknown$ then
3:          return
4:      if isDTNU($\psi$) then
5:          updateConstraints($\psi$)                        ▷ [6]*
6:          if IsLeaf($\psi$) then                            ▷ [7]*
7:              propagateTruth($\psi$)
8:              return
9:          Create d-OR child $\psi'$
10:         explore($\psi'$)
11:     if isOR($\psi$) then
12:         Create list of all children $\Psi'$               ▷ [8]*
13:         for $\psi' \in \Psi'$ do
14:             explore($\psi'$)
15:     if isAND($\psi$) then
16:         Create list of all children $\Psi'$               ▷ [9]*
17:         for $\psi' \in \Psi'$ do
18:             explore($\psi'$)
19:     if isWAIT($\psi$) then
20:         create w-OR child $\psi'$
21:         explore ($\psi'$)
22: function MAIN(DTNU $\gamma$)
23:     explore($\gamma$)
24:     if $\gamma.truth = True$ then
25:         return $True$
26:     else
27:         return $False$
```

[6]* updateConstraints(x): Updates the constraints of DTNU node x.
[7]* isLeaf(x): Sets the truth value of x to *true* and returns *true* if all constraints are satisfied. Sets the truth value to *false* and returns *true* if a constraint is violated. If no truth value can be inferred at this stage with the updated constraints, a second check is run to determine if all uncontrollable timepoints have occurred. If so, the corresponding DTN is solved, the truth value of x is updated accordingly, and the function returns *true* . Otherwise, no logical outcome can be inferred for the current state of the constraints because there remains at least one uncontrollable timepoint and this function returns *false* .
[8]* If this is a *d-OR* node, the list Ψ' contains all the children DTNU nodes resulting from either the decision of scheduling a controllable timepoint, or the *WAIT* node resulting from a wait if available. If this is a *w-OR* node, Ψ' contains all AND_{R_j} nodes, each of which possess a reactive wait strategy R_j
[9]* Here, the list Ψ' contains all DTNUs resulting from all possible combinations $\Lambda_1, \Lambda_2, ..., \Lambda_q$ of uncontrollable timepoints which have the potential to occur during the current wait.

11.5 Wait Period

Figure 8 gives an example of the third rule used to compute a wait duration.

11.6 Optimization Rules

The following rules are added to make branch cuts when possible.

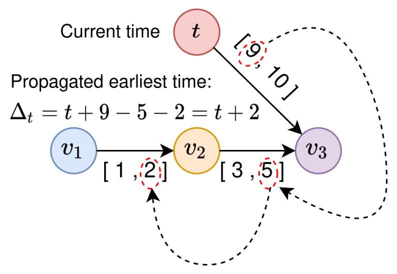


Figure 8: **Application of the 3^{rd} rule to determine a wait duration.** Current time is t. Variables v_1, v_2 and v_3 are timepoints. Here, v_2 is constrained to execute in the time interval $[1, 2]$ after v_1, v_3 in $[3, 5]$ after v_2 as well as in $[t + 9, t + 10]$. The rule suggests not to wait longer than 2 units of time at t: an execution of v_1 at $t + 2$, followed by an execution of v_2 at $t + 4$ opens a window of opportunity for v_3 to execute at $t + 9$.

Constraint Check. When a DTNU node is explored and the updated list of constraints C' is built according to §4.5, if a disjunct is found to be *false* , C' will no longer be satisfiable. All the subtree which can be developed from the DTNU will only have leaf nodes for which this is the case as well. Therefore, the search algorithm will not develop this subtree.

Symmetrical subtrees. Some situations can lead to the development of the exact same subtrees. A trivial example, for a given DTNU node at a time t, is the order in which a given combination of controllable timepoints $a_1, a_2, ..., a_k$ is taken before taking a wait decision. Regardless of what order these timepoints are explored in the tree before moving to a *WAIT* node, they will be considered executed at time t. Therefore, when taking a wait decision, it is checked that all preceding controllable timepoints executed before the previous wait are a combination of timepoints that has not been tested yet.

Truth Checks. Before exploring a new node for which the truth attribute is set to *unknown*, the truth attribute of the parent node is also checked. The node is only developed if the parent node's truth attribute is set to *unknown*. In this manner, when children of a tree node are being explored (depth-first) and the exploration of a child node leads to the assignment of a truth value to the tree node, the remaining unexplored children can be left unexplored.

11.7 Message Passing Layer

We use message passing layers that take as input a graph where nodes and edges possess features and return the graph with new node features. We detail pseudocode of a message passing layer applied to a graph $\mathcal{G} = (\mathcal{K}, \mathcal{E})$ in Algorithm 3.

11.8 Learning Implementation Details

Our MPNN architecture is made of 5 graph convolutional layers from (Gilmer et al. 2017). Each layer has a residual skip connection to the preceding layer (He et al. 2016), 32 abstract node features and a different two-layer MLP (multilayer perceptron) which has 128 neurons in its hidden layer. In addition, we use batch normalization after each graph

Algorithm 3 Message Passing Layer

```
1:  function MsgPass(Graph $\langle(\mathcal{K},\mathcal{E}),(H_\kappa,X_\epsilon,X_\rho)\rangle$)    ▷ [10]∗
2:      $H'_\kappa(\cdot,\cdot) \leftarrow 0$ // Initialize new node features matrix
3:      for all $\kappa_i \in \mathcal{K}$ do
4:          $h'_i \leftarrow 0$ // Initialize new features for $\kappa_i$
5:          for all $\kappa_j \in \mathcal{K}$ do
6:              if $X_\epsilon(\kappa_i,\kappa_j) = 1$ then
7:                  $\alpha \leftarrow X_\rho(\kappa_i,\kappa_j)$
8:                  $h \leftarrow H_\kappa(\kappa_j,\cdot)$
9:                  $h'_i \leftarrow h'_i + MLP(\alpha)h$    ▷ [11]∗
10:         $H'_\kappa(\kappa_i,\cdot) \leftarrow h'_i$ // Assign new features for $\kappa_i$
11:     return $\langle(\mathcal{K},\mathcal{E}),(H'_\kappa,X_\epsilon,X_\rho)\rangle$
```

[10]∗ $H_\kappa(\kappa_i,\cdot)$ returns a vector of current features for node κ_i; $X_\epsilon(\kappa_i,\kappa_j)$ returns 1 if $(\kappa_i,\kappa_j) \in \mathcal{E}$, 0 otherwise; $X_\rho(\kappa_i,\kappa_j)$ returns a vector of current features for edge (κ_i,κ_j).

[11]∗ MLP represents a multi-layer perceptron mapping input edge features to a matrix of dimension num-output-node-features x num-input-node-features. Moreover, h is of dimension num-input-node-features x 1. The matrix multiplication therefore results in a vector of size num-output-node-features.

layer and apply the $\mathrm{ReLU}(\cdot) = \max(0,\cdot)$ activation function. The input of the MPNN is the graph conversion of a DTNU. Figure 9 illustrates an example of graph conversion. We use 10 different edge distance classes: $0 : [0, 0.1)$, $1 : [0.1, 0.2)$, ..., $9 : [0.9, 1]$. Training is done with the *adagrad* optimizer (Duchi, Hazan, and Singer 2011) and an initial learning rate 10^{-4} on a dataset comprised of $30K$ instances generated as described in §6. We split the data into a training set comprised of $25K$ instances and a cross-validation set comprised of $5K$ instances. We add a dropout regularization layer with a *keep rate* 0.9 before the output layer to reduce overfitting.

11.9 Architecture Comparison

We study the impact of the design choices of the MPNN architecture on performance. To this end we compare different architectures of MPNN by varying depth and width (number of abstract node features per layer) and train them on the training set created in §6. We also assess the added value of residual skip connections to preceding layers. We create a benchmark of 400 DTNU instances, each of which has 20 to 25 controllable timepoints and up to 3 uncontrollable timepoints. We solve them using the tree search guided by each of these MPNN architectures. We limit the use of the MPNN architectures to a maximal depth of 50 (*d-OR* node-wise). Results are shown in Figure 10. We note the smallest network is too small to learn efficiently and performs poorly. Three-layer networks perform better. Wider networks perform slightly better for the same depth, black network 32 vs. green network 16. Overall, medium-depth networks of 5 layers work best. Residual connections lead to slight but steady gains. Interestingly, deeper networks (8+ layers) display lower scores compared to more shallower variants (5 layers), suggesting depth performance saturation. The quantity of training data can however be a limiting factor: we assume the optimal architecture to be actually deeper.

$$\gamma = \left|\begin{array}{l} a_2 - u_1 \in [0,1] \\ a_2 \in [0,1] \vee a_2 \in [1.5,3] \\ L = \{a_1,[0,2],u_1\} \end{array}\right.$$

$$\gamma' = \left|\begin{array}{l} a_2 - u_1 \in [0,0.33] \\ a_2 \in [0,0.33] \vee a_2 \in [0.5,1] \\ L = \{a_1,[0,0.66],u_1\} \end{array}\right.$$

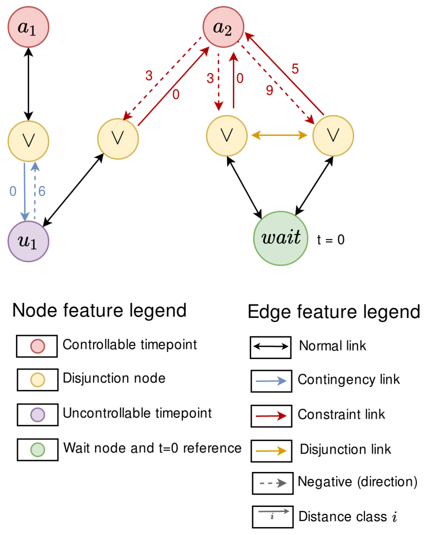


Figure 9: **Conversion of a DTNU γ into a graph.** γ' is the normalized DTNU. Edge distances are expressed as distance classes. To distinguish between lower and upper bounds in intervals, we introduce an additional *negative directional sign* feature.

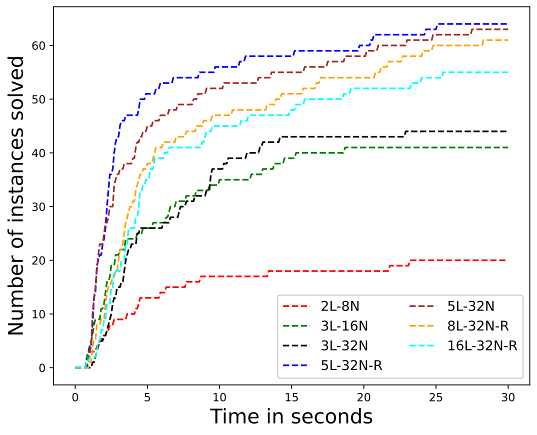


Figure 10: **Comparison of different MPNN architectures.** The notation XL-YN refers to an MPNN with X layers and Y abstract node features per layer. The ”-R” tag refers to the presence of residual layers. Experiments are done on a DTNU benchmark containing 400 instances with 20 to 25 controllable timepoints and up to 3 uncontrollable timepoints per DTNU. Timeout is set to 30 seconds per DTNU instance.