Incremental Dynamic Controllability Revisited

Mikael Nilsson and **Jonas Kvarnström** and **Patrick Doherty**
Department of Computer and Information Science
Linköping University, SE-58183 Linköping, Sweden
{mikni,jonkv,patdo}@ida.liu.se

Abstract

Simple Temporal Networks with Uncertainty (STNUs) allow the representation of temporal problems where some durations are determined by nature, as is often the case for actions in planning. As such networks are generated it is essential to verify that they are dynamically controllable – executable regardless of the outcomes of uncontrollable durations – and to convert them to a dispatchable form. The previously published FastIDC algorithm achieves this incrementally and can therefore be used efficiently during plan construction. In this paper we show that FastIDC is not sound when new constraints are added, sometimes labeling networks as dynamically controllable when they are not. We analyze the algorithm, pinpoint the cause, and show how the algorithm can be modified to correctly detect uncontrollable networks.

Introduction and Background

Time and concurrency are increasingly considered essential in automated planning, with temporal representations varying widely in expressivity. For example, Simple Temporal Problems (STPs, Dechter, Meiri, and Pearl 1991) allow us to efficiently determine whether a set of *timepoints* (events) can be assigned real-valued *times* in a way consistent with a set of *constraints* bounding the temporal distances between timepoints. Clearly the start and end of every action in a plan could be represented as such a timepoint, but we can only represent the possible durations of an action as an STP constraint if the execution mechanism can choose durations arbitrarily within the given bounds. Though this is sometimes true, exact durations are often instead chosen by nature.

In STPs with Uncertainty (STPUs, Vidal and Ghallab 1996), a *controlled* timepoint corresponds to the start of an action while a *contingent* timepoint corresponds to its end. The distance between a controlled and a contingent timepoint can be limited by a special *contingent* constraint, representing possible action durations. We then have to find a way to assign times to the controlled timepoints (determine when to start actions) so that for *every* possible outcome for the contingent constraints (which will be decided by nature during execution), there exists *some* solution for the ordinary "STP-like" *requirement* constraints.

If an STNU allows controlled timepoints to be scheduled (actions to be started) incrementally given that we immediately receive information when a contingent timepoint occurs (an action ends), it is *dynamically controllable* (DC) and can be efficiently executed by a dispatcher (Vidal and Fargier 1997). Conversely, ensuring that constraints are satisfied when executing a non-DC plan is impossible: It would require information about future contingent timepoints.

If a plan is not dynamically controllable, adding actions cannot restore controllability. Therefore a planner should verify after each action addition whether the plan remains DC, and if not, backtrack. As testing dynamic controllability takes non-trivial time, one can benefit greatly from using an *incremental* DC verification algorithm rather than redoing the analysis for each new action or constraint. This precludes the use of most known algorithms (Morris, Muscettola, and Vidal 2001; Morris and Muscettola 2005; Morris 2006; Stedl 2004). However, *FastIDC* does support incremental tightening/addition of constraints, which we need, as well as incremental loosening/removal, which is not strictly required for our purposes assuming chronological backtracking (Stedl and Williams 2005; Shah et al. 2007).

In this paper we demonstrate that the tightening/addition algorithm in FastIDC, BackPropagate-Tighten (BPT), cannot always detect plan modifications violating dynamic controllability. We pinpoint and analyze the cause of the problem and show how to solve it, resulting in a sound version of BPT and hence FastIDC.

Definitions

We begin by formally defining STPs, STPUs, and DC.

Definition 1. *A **simple temporal problem (STP**, Dechter, Meiri, and Pearl 1991) consists of a number of real variables $x_1,\ldots,x_n$ and constraints $T_{ij} = [a_{ij},b_{ij}], i \neq j$ limiting the distance $a_{ij} \leq x_j - x_i \leq b_{ij}$ between the variables.*

Definition 2. *A **simple temporal problem with uncertainty (STPU**) (Vidal and Ghallab 1996) consists of a number of real variables $x_1,\ldots,x_n$, divided into two disjoint sets of* controlled timepoints *R and* contingent timepoints *C. An STPU also contains a number of* requirement constraints *$R_{ij} = [a_{ij},b_{ij}]$ limiting the distance $a_{ij} \leq x_j - x_i \leq b_{ij}$, and a number of* contingent constraints *$C_{ij} = [c_{ij},d_{ij}]$ limiting the distance $c_{ij} \leq x_j - x_i \leq d_{ij}$. For the constraints C_{ij} we require that $x_j \in C$, $0 < c_{ij} < d_{ij} < \infty$.*

Algorithm 1: BackPropagate-Tighten (Shah et al. 2007)

```
function BackPropagate-Tighten G, e_1, ..., e_n
Q ← sort e_1, ..., e_n by distance to temporal reference
      (order important for efficiency, irrelevant for correctness)
for each modified edge e_i in ordered Q do
    if IS-POS-LOOP(e_i) then SKIP e_i
    if IS-NEG-LOOP(e_i) then return false
    for each rule (Figure 1) applicable with e_i as focus do
        if edge z_i in G is modified or created then
            if G is squeezed then return false
            if not BackPropagate-Tighten G, z_i then return
            false
        end
    end
end
return true
```

Both STPs and STPUs can be represented in graph form with timepoints as nodes and constraints as edges, in which case they are called Simple Temporal Networks (STNs) and STNs with Uncertainty (STNUs), respectively. As we will work with graphs, we use this terminology below.

Definition 3. *A **dynamic execution strategy** is a strategy for assigning timepoints to controllable events during execution, given that at each timepoint, it is known which contingent events have already occurred. The strategy must ensure that all requirement constraints will be respected regardless of the outcomes for the contingent timepoints.*

*An STNU is said to be **dynamically controllable (DC)** if there exists a dynamic execution strategy for it.*

The BackPropagate-Tighten Algorithm

The basis for the BackPropagate-Tighten aspect of FastIDC (Stedl and Williams 2005; Shah et al. 2007) is an earlier non-incremental DC verification algorithm, sometimes called MMV (Morris, Muscettola, and Vidal 2001). We give a brief overview of these algorithms below and refer the reader to the cited articles for further details and explanations.

MMV first verifies that the graph is *pseudo-controllable*: That it is consistent in the STN sense and that one cannot locally detect that some contingent constraint is *squeezed*, meaning that some of nature's possible outcomes are forbidden. It then iterates over all triples of nodes in the graph using a set of rules to derive new or tighter edges, increasing the information available for the local squeeze check. This can also introduce *wait* constraints labeled $\langle N, z \rangle$, indicating that the execution of a controllable event has to wait until the uncontrollable event N has occurred *or* z timepoints have passed. These steps are repeated until the graph is not pseudo-controllable or no more changes can be derived. In the latter case it is dynamically controllable as well as *dispatchable* (Muscettola, Morris, and Tsamardinos 1998): It has sufficient edges for an execution mechanism to efficiently determine how to execute it using only local information.

Being incremental, FastIDC assumes that there already exists a dispatchable and DC STNU G and that one or more edges $e_1, \ldots, e_n$ in G are added, altered or removed. BackPropagate-Tighten (BPT, Algorithm 1) handles the case of adding or tightening edges. It works on the corresponding distance graph, where edges are kept marked as "requirement", "contingent", or "conditional" in the case of waits, and applies a modified set of derivation rules (Figure 1) only to triples where an altered edge is involved as a *focus edge*. Additionally, an *unconditional unordered reduction* rule (Morris, Muscettola, and Vidal 2001) is applied in certain cases involving conditional edges. As our examples do not use or result in such edges, we omit this rule here.

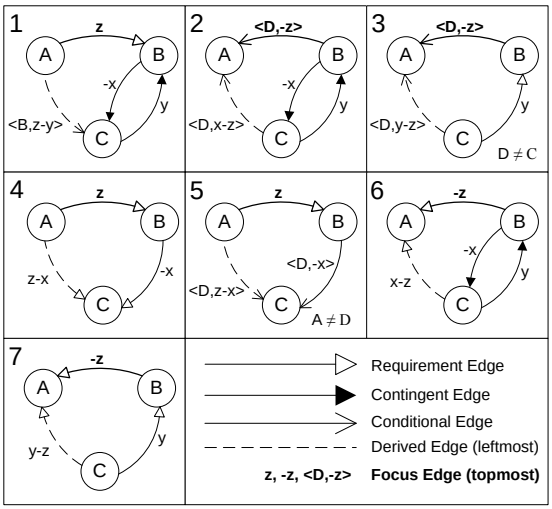

Figure 1: BackPropagate-Tighten Derivation Rules.

Whenever the rules result in additional altered edges, the procedure treats these recursively as focus edges, spreading changes gradually from the location of the original change. If a negative self-loop $N \xrightarrow{\text{neg}} N$ or a local squeeze results, the graph is not DC. Otherwise, it is claimed that the graph is again DC and dispatchable. Examples will be given soon. It has been empirically shown that this is faster than repeatedly checking *all* triples as in MMV.

A Counterexample

Figure 2 shows what happens when all BackPropagate-Tighten rules are applied to an example network where U occurs uncontrollably between 5 and 50 timepoints after A. Initially edges (1)-(3) are not present, and the network is dispatchable and DC. A requirement edge (1) is added by a planner: U must occur at least 15 timepoints after D. Rule 6 is matched with UD as its focus, resulting in a new edge (2):

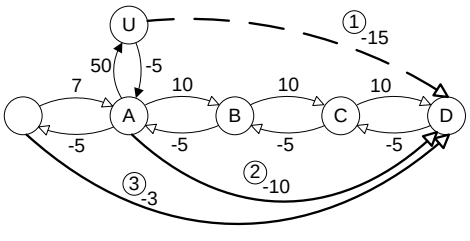

Figure 2: Graph where DC violations are missed.

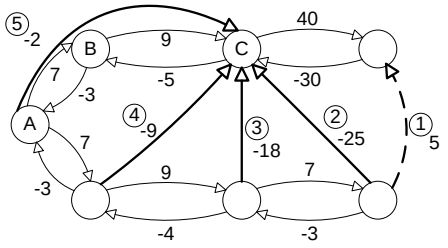

Figure 3: Graph where an STN inconsistency is missed.

Nature *could* decide there will be as few as 5 timepoints between A and U, so there must be at least 10 timepoints between D and A. The only remaining match is for rule 7, resulting in edge (3).

No negative self-loop is generated. Will the algorithm consider G to be squeezed? In MMV this was tested globally using an all pairs shortest path (APSP) algorithm. This would be extremely inefficient in an incremental algorithm, and indeed Stedl and Williams (2005) and Shah et al. (2007) state that APSP algorithms are not used. Then a local check for squeezing must be used, and there are no edges that locally imply that the bounds on the distance between A and U are squeezed. Therefore, BPT will return true – but the graph is not DC: The path UDCBA shows U must be at least 30 after A, while the edge UA shows nature can decide that U will be only 5 after A.

Problem Analysis

To determine *why* BPT can miss the fact that a graph is not DC, we take a closer look at the steps indicated in Figure 2. It is clear that the graph was initially DC and dispatchable. When an edge (1) was added, the distance graph remained globally consistent. However, since this edge involved a contingent timepoint (U), we could derive an additional edge (2) that would not have been entailed in an ordinary STN. This resulted in a *negative cycle* and an inconsistent graph.

BPT should detect such cycles. In fact, Shah et al. (2007) explicitly state that if we have a dispatchable distance graph for an ordinary STN, tighten or add an edge, recursively apply *only rules 4 and 7* and check for negative self-cycles, this "will either expose a direct inconsistency or result in a dispatchable graph".

Below we will call the recursive application of rules 4 and 7 Incremental Dispatchability (ID), as it was originally called in Stedl (2004). To demonstrate more clearly how ID works without reference to STNUs, we turn to the pure STN in Figure 3. Initially edges (1)-(5) do not exist. The dashed constraint (1) is added, leading to a negative cycle. Rules 4 and 7 are used to derive edges (2)-(5), after which no further edges can be derived except for positive self-loops, which we omit as they neither indicate inconsistency nor lead to the generation of additional edges. As ID fails to find a negative self-loop it considers the resulting graph consistent and dispatchable, which it is not. Why, and what can we do about it while retaining efficiency?

Reasons for Failure. To see why ID fails we compare it to MMV, which detects negative cycles by running an APSP algorithm. This is equivalent to repeatedly taking *all* edge pairs $A \xrightarrow{x} B \xrightarrow{y} C$ and deriving/tightening edges $A \xrightarrow{x+y} C$. In ID, rules 4 and 7 only consider edge pairs where $x > 0$ and $y \leq 0$, which is part of the reason for its efficiency.

Lemma STN-DBP (Shah et al. 2007) proves that these derivations are *valid*. The motivation for why they should also be *sufficient* is quite intuitive and based on the fact that as long as a dispatcher ensures that each scheduling decision it makes is consistent with the past, it *will* also be possible to consistently schedule any future timepoint. In other words, any constraint that could possibly be inferred from future timepoints has already been explicitly applied to the current timepoint through new edges derived when the graph was made dispatchable.

As ID requires that the graph was dispatchable before the tightening or addition, it is argued that ID also only has to consider consistency with past timepoints: "To maintain the dispatchability of the STN when a constraint is tightened by a fast replanner, we only need to make the modified constraint consistent with past scheduling decisions, since during execution, the bounds on events are only influenced by preceding events" (Shah et al. 2007). Thus, when a positive constraint AB is tightened, rule 4 only considers how this will affect nodes C forced to be in the past from B, and similarly for rule 7.

This reasoning presumes that there is a well-defined past at each node, towards which the recursion can proceed. This is true when a graph *known* to be dispatchable is executed, but now we are verifying *whether* a change preserves dispatchability. In Figure 3 we violated dispatchability and could deduce both that C must be before A and vice versa: The STN is inconsistent, and so is the concept of "past".

Since ID determines that we must have A before B before C, it does not derive a new edge from AB and BC. The reasoning is again that at execution time, A must occur before B, and *then* the dispatcher will propagate the resulting constraints towards C in the future. This holds for all combinations of negative edges, so the negative cycle is never shortened to a negative self-loop and is therefore missed.

Resolving the Problem. One possible means of resolving this problem would be to fall back on detecting negative cycles using for example incremental APSP calculations or incremental directed path consistency (Chleq 1995) algorithms. But this would lead to the same inefficiency that back-propagation was intended to avoid. For the best possible performance, we would prefer to determine whether we can benefit from the work that is already done when new edges are derived by ID and BPT.

We therefore observe Figure 3 more closely and find that it contains not only a negative cycle but a cycle consisting entirely of negative edges. It turns out that this will always be the case when ID fails to discover an inconsistency.

In the following we assume that any path is simple, i.e. does not contain a repeated node, and that any cycle is simple, i.e. contains only one path from each node to itself.

Lemma 1. *Consider all paths of a given weight n between two nodes N and N' in a distance graph that was constructed incrementally by ID. The shortest of these paths, in terms of the number of edges, must have one of the following forms:*

1. *It contains only positive edges.*
2. *It consists of at least one negative edge followed by zero or more positive edges.*

Proof. If a path with the smallest number of edges does not have this form, it must at some point contain a positive edge before a negative edge: $N \cdots A \xrightarrow{+} B \xrightarrow{-} C \cdots N'$. Either when the edge AB was added/tightened or when the edge BC was added/tightened, ID would have used rule 4 or 7 to derive a new edge $A \rightarrow C$ whose weight was the sum of the weights of $A \rightarrow B$ and $B \rightarrow C$. There would then exist a path between N and N' with the same weight but fewer edges, leading to a contradiction. □

Theorem 1. *Let G be a consistent and dispatchable distance graph constructed using ID. Assume that the edge $A \xrightarrow{f} B$ is added or tightened in G and that the corresponding STN is then inconsistent. Then after applying ID, there will be a cycle in the distance graph consisting of only negative edges.*

Proof. By induction. Suppose that after adding or tightening an edge in G but before applying ID, G is inconsistent. Then G has at least one negative cycle (Dechter, Meiri, and Pearl 1991). Let C_k be a negative cycle in G with the smallest number of edges. Let $k \geq 1$ be the number of edges in C_k.

Basis: If $k = 1$, there is already a cycle of only one negative edge. This cycle will remain after applying ID, because ID never removes edges and never increases edge weights.

Induction assumption: The theorem holds for $k-1$.

Induction step: Does the theorem hold for k, where $k > 1$?

First, if all edges in C_k are negative, then they will remain negative after ID and the theorem holds. Otherwise, at least one edge in the cycle is positive, but there must also be at least one negative edge (else C_k could not be negative).

We know C_k consists of the newly tightened or added edge $A \xrightarrow{f} B$ together with some non-empty path from B to A. If there exist other paths from B to A of the same weight, they cannot have fewer edges – otherwise there would have been a shorter negative cycle than C_k, violating the assumption. Thus, the path from B to A included in C_k has the fewest edges among all paths from B to A of the same weight, and Lemma 1 is applicable.

Suppose that the new value of f is negative. As the entire cycle did not consist of negative edges, there must be at least one positive edge on the path from B to A. This together with the lemma shows that there must be a positive edge $X \rightarrow A$ at the end of that path, for some node X. Since the edge $A \rightarrow B$ was altered, ID will apply rule 7 to derive an edge $X \rightarrow B$, yielding a cycle with the same negative weight but with $k-1$ edges. By the induction assumption, ID will then reduce this cycle to a negative-edge-only cycle.

Suppose instead that the new value of f is positive. As the cycle was negative, the path from B to A must have negative weight, so case 2 of the lemma must hold and the path must begin with a negative edge $B \rightarrow Y$. Since the edge $A \rightarrow B$ was altered, ID will apply rule 4 to derive a new edge $A \rightarrow Y$, again yielding a cycle with the same negative weight but with $k-1$ edges. By the induction assumption, ID will then reduce this cycle to a negative-edge-only cycle. □

As shown in Figure 3, tightening an edge can indeed yield a cycle of *multiple* negative edges, which is not detected by ID. This is still problematic, but we have now verified that it suffices to detect negative-edge-only cycles rather than arbitrary negative cycles also containing positive edges. To detect these we do not need to take edge weights into account.

We therefore incrementally build an *unweighted* Cycle Checking (CC) graph containing the same nodes as the ID distance graph and with a directed edge exactly where the distance graph has a negative edge. Since edge weights can only be decreased and not increased, edges in the CC graph will never be removed. Also, tightening an already negative edge does not change the CC graph.

We then find cycles in the CC graph using an efficient incremental topological ordering algorithm which does not need to take edge weights into account. Since ID generates many negative edges for propagating time bounds during dispatch, an algorithm for dense graphs appears best suited. One such algorithm has a has a runtime of $O(n^2 \log n)$ for incrementally cycle-checking a graph with n nodes and a maximum of $O(n^2)$ edges (Bender et al. 2011). This is dominated by the runtime of BPT, which was empirically shown (Stedl and Williams 2005) to be around $O(n^3)$ in practice.

We can only sketch a correctness proof of the modified BackPropagate-Tighten here. It builds on two facts: (1) In any given situation FastIDC never derives constraints that MMV would not. Therefore it never labels a graph as non-DC by mistake. (2) With the addition of the cycle checking it can be proven that any non-DC graph is detected and correctly labeled. If a non-DC graph was labeled as DC there would be a node that was the first to violate its associated constraints during dispatch. By using the derivation rules and general reduction together with the fact that the graph was free of negative cycles it can be shown that no such first node can exist: Derivation in this case finds an earlier node which was violated.

Conclusions

The FastIDC algorithm presented by Stedl and Williams (2005) and Shah et al. (2007) incrementally verifies dynamic controllability, which is essential when generating plans with the full expressivity of Simple Temporal Networks with Uncertainty. We have shown that the algorithm in certain cases fails to detect networks that are not dynamically controllable, which could potentially lead to planners accepting invalid plans. The problem was localized to the incremental dispatchability and consistency checking part of the BackPropagate-Tighten algorithm. We then analyzed the properties of the problem, resulting in a modification ensuring that inconsistencies will be detected while retaining the incremental properties of the algorithm. The increase in runtime has an upper bound of $O(n^2 \log n)$, which is dominated by the $O(n^3)$ empirical bound of the FastIDC algorithm.

Acknowledgments

This work is partially supported by The Swedish Research Council (VR) Linnaeus Center for Control, Autonomy, and Decision-making in Complex Systems (CADICS), the ELLIIT network organization for Information and Communication Technology, the Swedish Foundation for Strategic Research (CUAS Project) and the EU FP7 project SHERPA, grant agreement 600958.

References

Bender, M.; Fineman, J.; Gilbert, S.; and Tarjan, R. 2011. A new approach to incremental cycle detection and related problems. *arXiv preprint arXiv:1112.0784*.

Chleq, N. 1995. Efficient algorithms for networks of quantitative temporal constraints. In *Proceedings of CONSTRAINTS-95*, 40–45.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61–95.

Morris, P., and Muscettola, N. 2005. Temporal dynamic controllability revisited. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 1193–1198. AAAI Press / The MIT Press.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, 494–499. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Morris, P. 2006. A structural characterization of temporal dynamic controllability. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 4204 of *Lecture Notes in Computer Science*, 375–389. Springer.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR)*.

Shah, J. A.; Stedl, J.; Williams, B. C.; and Robertson, P. 2007. A fast incremental algorithm for maintaining dispatchability of partially controllable plans. In Boddy, M. S.; Fox, M.; and Thibaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS)*, 296–303. AAAI Press.

Stedl, J., and Williams, B. 2005. A fast incremental dynamic controllability algorithm. In *Proceedings of the ICAPS Workshop on Plan Execution: A Reality Check*.

Stedl, J. L. 2004. Managing temporal uncertainty under limited communication: A formal model of tight and loose team coordination. Master's thesis, Massachusetts Institute of Technology.

Vidal, T., and Fargier, H. 1997. Contingent durations in temporal CSPs: From consistency to controllabilities. In *Proceedings of the 4th International Workshop on Temporal Representation and Reasoning (TIME)*, 78. Washington, DC, USA: IEEE Computer Society.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraints networks dedicated to planning. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI)*, 48–52.