Managing Dynamic Multi-Agent Simple Temporal Network

Guillaume Casanova
Guillaume.Casanova@onera.fr

Charles Lesire
Charles.Lesire@onera.fr

Cédric Pralet
Cedric.Pralet@onera.fr

Onera — The French Aerospace Lab
2 Avenue Édouard Belin
31000 Toulouse, France

ABSTRACT

The realization of plans of activities by several agents is usually subject to a set of temporal constraints. The latter are either constraints holding on each agent individually, or constraints allowing the activities of agents to be synchronized. To represent the set of temporal constraints imposed on distributed plans, the framework of Multi-agent Simple Temporal Network (MaSTN) can be used. In this paper, we consider the problem of maintaining the temporal consistency of distributed plans during execution, when temporal constraints may be updated. To this end, we propose new incremental algorithms for managing dynamic MaSTNs. These algorithms help each agent know, as soon as possible, whether the distributed plan to be executed is still temporally consistent. They range from algorithms focusing on keeping as much as possible the privacy between agents to algorithms focusing on improving the response time to updates through more information sharing. We analyze the robustness of these algorithms when communications are intermittent, and we provide experimental results demonstrating the trade-off to be made between performance and privacy.

1. INTRODUCTION

Autonomous robotics applications such as the automatic exploration of large and hazardous areas can have increased performances by the use of multiple robots [10], providing redundancy and parallelism capabilities to the robots team. It is especially interesting to take advantage of heterogeneous robots to assign a large number of capabilities into several entities at a reasonable cost. The cooperation between robots will thus lead to allocate the several tasks of the mission to each robot depending on their capabilities. These tasks will moreover be linked by temporal constraints, enforcing precedence relations or synchronization between tasks.

Simple Temporal Networks (STNs) [5] are a widely used tool for handling temporal constraints. Many planning algorithms used in robotics generate a plan represented as an STN [8, 7, 6, 13]. Multiple tools and algorithms already exist to efficiently propagate constraints on STNs or to maintain STNs during execution while taking into account possible changes in the environment or in the mission specification, such as delays on tasks execution.

Multi-robot systems generally evolves in complex and constrained environments. In such environments, the communication between robots is often chaotic, leading to delayed or even intermittent communications. In order to have the team be robust to both dynamic environments and intermittent communications, it seems mandatory to implement a distributed autonomous system. In such an architecture, it is then impossible to have a unique STN for the whole team being maintained in a centralized way. The Multiagent STN (MaSTN) formalism [1] allows each agent to be only aware of temporal constraints that are involved in its private plan. It has been shown that Incremental Partial Path Consistency (IPPC) can be enforced on MaSTNs, using Distributed Incremental ΔSTP (DIΔSTP) [1] and Distributed Incremental Partial Path Consistency (DIPPC) [1]. However, these algorithms have not been designed to be robust to intermittent communications.

In this paper we propose four algorithmic variants to maintain consistency in a MaSTN: CIP (Centralized Incremental Propagation), a centralized algorithm based on a supervisor agent entirely managing the MaSTN consistency; DIP-G (Distributed Incremental Propagation with Global information sharing), a distributed algorithm based on sharing every information to every agent; DIP-L (Distributed Incremental Propagation with Local information sharing), which focuses on reproducing the mono-agent propagation algorithm across the MaSTN while keeping maximum privacy; and DIP-M (Distributed Incremental Propagation with Macro information sharing), which focuses on increasing performance by lowering privacy requirements. Sec. 2 introduces some STN and MaSTN definition and notations. Sec. 3 then presents the four proposed algorithms. An analysis of the algorithms is proposed in Sec. 4. Sec. 5 finally evaluates and compares the four algorithms according to the number of messages exchanged and required computations to propagate a new disturbance, on randomly generated benchmarks.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

2. BACKGROUND

2.1 Simple Temporal Network (STN)

An STP (*Simple Temporal Problem* [5]) is a pair $S = (V, E)$ composed of:

- a set of time-point variables $V = \{v_1, \ldots, v_n\}$, which are associated with the occurrence of some events in time; a specific variable v_0 called the *reference-point* is usually added to V for representing a reference temporal position;
- a set of constraints E, each constraint $e \in E$ being defined by a set of variables $\{v_i, v_j\} \subseteq V$, with $i < j$, and by two bounds $l_{ij} \in \mathbb{R} \cup \{-\infty\}$ and $u_{ij} \in \mathbb{R} \cup \{+\infty\}$ which respectively specify a minimum and a maximum temporal distance between v_i and v_j (temporal constraint $v_j - v_i \in [l_{ij}, u_{ij}]$); we assume without loss of generality that there are not several constraints in E which hold on the same two variables; unary temporal constraints such as $v_i \in [a, b]$ can be easily expressed as distance constraints with regards to the

reference-point (constraints $v - v_0 \in [a, b]$); moreover, each temporal constraint $v_j - v_i \in [l_{ij}, u_{ij}]$ can be rewritten as two inequality constraints $v_j - v_i \geq l_{ij}$ and $v_i - v_j \geq -u_{ij}$ called *simple temporal constraints*.

From this definition, a *solution* to an STP (V, E) is defined as an assignment of all variables in V such that all temporal constraints in E are satisfied. An STP is said to be *consistent* if it admits a solution, *inconsistent* otherwise. Last, an STP has a natural graphical representation called an STN (*Simple Temporal Network*), which contains one vertex per variable in V and one edge $v_i \rightarrow v_j$ labeled by $[l_{ij}, u_{ij}]$ per temporal constraint $v_j - v_i \in [l_{ij}, u_{ij}]$ in E.

STNs are appealing in practice to deal with temporal aspects because several problems that can be formulated on STNs are solvable in polytime, such as: (1) determining whether an STN is consistent; (2) determining for each time-point variable v its earliest and latest occurrence times l_v and u_v in a solution; (3) determining the minimum and maximum temporal distances between any two variables in a solution. Several algorithms exist for solving such problems, inspired either by shortest path algorithms on graphs or by algorithms developed in the constraint programming community [5, 3, 15, 12].

Algorithms for reasoning about STNs were also extended to a dynamic context [2, 11], where temporal constraints may be updated with two kinds of possible modifications:

- *constraint tightening*, when a simple temporal constraint $w - v \geq d$ is updated to $w - v \geq d'$ with $d' > d$; in this case, incremental reasoning is usually performed by maintaining a queue of temporal constraints to be revised in order to recompute the consistency of the STN or the earliest/latest dates associated with time-point variables; the addition of a constraint to the STN can be seen as the tightening of a virtual edge labeled by $[-\infty, +\infty]$;

- *constraint relaxation*, when a simple temporal constraint $w - v \geq d$ is updated to $w - v \geq d'$ with $d' < d$; in the relaxation case, incremental reasoning is performed by reusing information recorded during previous computations, such as *propagation chains* which record explanations for the current time bounds of time-point variables: if a simple temporal constraint is relaxed and if it explained the lower/upper bound of a variable v, then this bound is reinitialized, as well as all lower/upper bounds of time-point variables which belong to the tree of propagation chains rooted in v [2]; the deletion of a constraint can be seen as the relaxation of an edge to $[-\infty, +\infty]$.

For the rest of the paper, we do not detail these tightening and relaxation algorithms. We only assume that we have two functions called `IncrRelax`(Rlx) and `IncrPropag`(Rvs) respectively. The former takes as an input a set Rlx of edge relaxations represented by triples (v, w, b) (relaxation of bound $b \in \{LB, UB\}$ of edge $v \rightarrow w$); it reinitializes time-bounds of vertices based on propagation chains, and it returns temporal constraints to be revised after the relaxation operation. The latter takes as an input a set Rvs of temporal constraints to be revised, each of which is described by a tuple (v, w, b) (revision of edge $v \rightarrow w$ for bound type b required); it returns a set of pairs (v, b) describing time-point bounds updated by the propagation of temporal constraints. We assume that calls to `IncrPropag` and `IncrRelax` update all features associated with the STN such as its consistency, the lower and upper bounds associated with time-point variables, and propagation chains.

2.2 Multi-agent STN (MaSTN)

STN were extended to a multi-agent context, where time-point variables are not controlled by a single agent but instead are partitioned among a set of agents $\mathcal{A}$. This extension is called MaSTN for *Multiagent Simple Temporal Network* [1]. Formally, an MaSTN is defined by:

- a set of N *local STNs*, one per agent $A \in \mathcal{A}$; the *local* STN associated with agent A, denoted by S_L^A, is defined by V_L^A the set of *local* vertices owned by a, and E_L^A the set of *local* edges connecting two local vertices $v, w \in V_L^A$;

- a set of edges E_X which connect the local STNs; each edge in E_X represents an *external* constraint and connects two local vertices of different agents.

Apart from its local edges in E_L^A, each agent A is aware of the subset of external constraints E_X^A which hold on one of its local vertices ($E_X^A = \{\{v, w\} \in E_X | v \in V_L^A\}$). Apart from its local vertices in V_L^A, each agent is aware of the set V_X^A composed of vertices not owned by A but involved in E_X^A ($V_X^A = \{v \in V | \exists w \in V_L^A, \{v, w\} \in E_X\}$). Therefore, the set of variables known by agent A is $V^A = V_L^A \cup V_X^A$, and the set of edges known by A is $E^A = E_L^A \cup E_X^A$. Additionally, we define for each agent A the set V_F^A of *frontier* vertices of A as the subset of local vertices of A connected to at least one external vertex ($V_F^A = \{v \in V_L^A | \exists \{v, w\} \in E_X\}$). In the following, we denote by *owner*(v) the unique agent which owns variable v, that is the agent A that has v in the set of its local vertices V_L^A.

Fig. 1 gives an example of an MaSTN involving three agents A, B, C. Agent A (resp. B and C) owns variables v_1^A to v_6^A (resp. v_1^B to v_8^B and v_1^C to v_8^C). In its plan, agent A must perform acquisition *acq1*, get energy from agent B, and then perform a maintenance operation. Agent B must perform acquisition *acq2*, charge agent A, and receive data from agent C. A reception instrument must be switched on (variable v_5^B) and off (variable v_8^B) before and after the reception of data from C. Agent C must perform two acquisitions (*acq3* and *acq4*) before transmitting the associated data to agent B. Some of the temporal constraints defined specify bounds on the durations of activities and on transition times between activities. Other temporal constraints are requirements, such as the constraint linking variables v_2^C and v_5^C which may correspond to a requirement to transmit acquisition *acq3* sufficiently fast, or the constraint which links reference-point v_0 with variable v_8^C to limit the duration of the plan of C.

On this example, the sets of external vertices for agents are $V_X^A = \{v_3^B, v_4^B\}$, $V_X^B = \{v_3^A, v_4^A, v_5^C, v_8^C\}$, and $V_X^C = \{v_6^B, v_7^B\}$. The sets of frontier vertices are $V_F^A = \{v_3^A, v_4^A\}$, $V_F^B = \{v_3^B, v_4^B, v_6^B, v_7^B\}$, and $V_F^C = \{v_5^C, v_8^C\}$. External constraints in E_X are depicted with dotted lines. They represent synchronization points between agents.

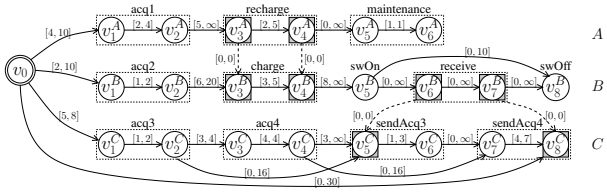


Figure 1: Example of a Multi-agent STN involving 3 agents

3. INCREMENTAL ALGORITHMS FOR DYNAMIC MULTI-AGENT STN

We now consider dynamic MaSTN, in which temporal constraints may be updated following information received at execution time. For instance, the duration of activities may be longer or shorter than expected, time windows available for realizing tasks may change, or agents may replan and change their internal temporal constraints. In such situations, our goal is to incrementally recompute, at the level of each agent A, the current consistency of the MaSTN as well as lower and upper bounds on the variables owned by A. We study four algorithmic variants for managing dynamic MaSTNs. These variants differ in the set of temporal constraints considered by each agent and in the kind of information shared between agents.

3.1 Basic assumptions

In the following, we assume that all external constraints in E_X are static: no change on the existence of these constraints and no change on their labels. The reason for this is that we consider that the modification of a temporal constraint shared by several agents must be handled by a more complex resynchronization process between agents, or by a process which assigns to a single agent the responsibility for updating the constraint.

Also, we make no assumption on the protocol used for communications (broadcast, unicast, multicast...), on how data are actually routed on the communication network formed by the agents, on how reception acknowledgments are handled, on how messages are re-emitted in case of intermittent communications, and on how queues of messages not sent yet are managed. The only assumptions made are that there may be some latency in message transmission, and that when messages are transmitted from agent A to agent B, messages sent first by A are received first by B.

3.2 A first glance at the algorithmic variants

Fig. 2 gives a first glance of the kind of temporal knowledge manipulated by each agent for the four incremental algorithms proposed, given the MaSTN provided in Fig. 1. The four algorithms are called CIP, DIP-G, DIP-M and DIP-L:

- CIP (Fig. 2(a)) stands for "Centralized Incremental Propagation"; in CIP, a supervisor agent maintains all temporal constraints and is responsible for reasoning about these constraints; it receives notifications of changes from other agents and sends back updates regarding the consistency of the MaSTN and the time bounds associated with variables; other agents are only aware of their own local STN;
- DIP-G (Fig. 2(b)) stands for "Distributed Incremental Propagation with Global information sharing"; in DIP-G, each agent maintains the set of all temporal constraints involved in the MaSTN, even the constraints it is not supposed to be aware of; as soon as a change occurs in a local edge of an agent, the latter sends the information to all other agents;
- DIP-L (Fig. 2(c)) stands for "Distributed Incremental Propagation with Local information sharing"; in DIP-L, each agent reasons only about the set of constraints it is supposed to be aware of, and the only data shared between agents are time bounds on external vertices;
- DIP-M (Fig. 2(d)) stands for "Distributed Incremental Propagation with Macro information sharing"; in DIP-M, each agent A reasons about two kinds of constraints: (1) its own local temporal constraints, and (2) a macroscopic view of temporal constraints of other agents; this macroscopic view provides distance constraints between external vertices; each agent is responsible for sending updates on its own macroscopic view, and hence never reveals its internal edges.

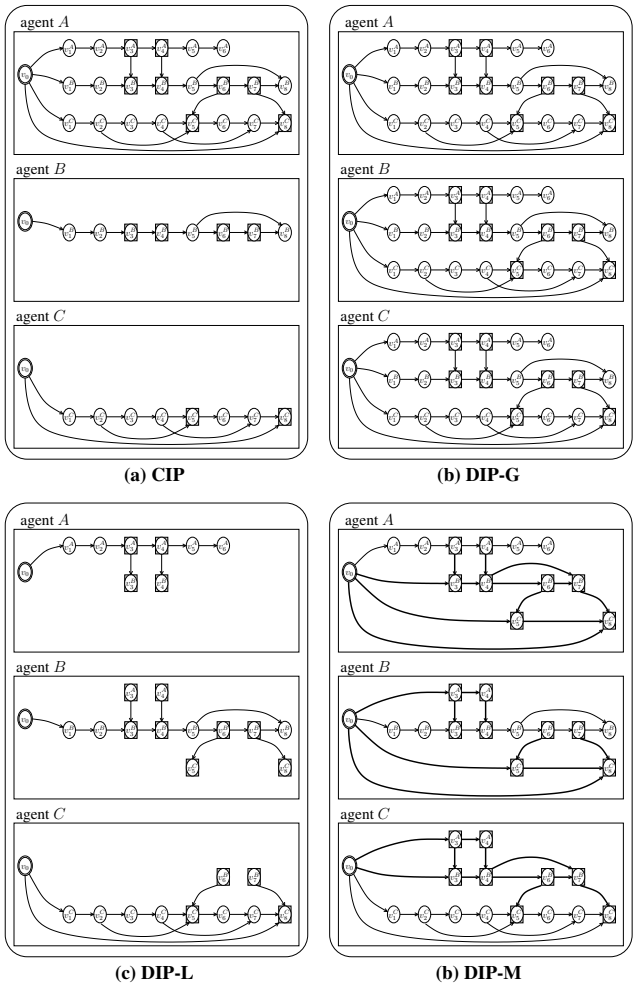


Figure 2: The four incremental algorithms proposed

3.3 Basic Functions and Data Structures

Before providing the details of the algorithms, we introduce some functions and data structures. To send messages, each agent uses a function called `send`$(dstList, content)$ which takes as inputs a list $dstList$ of agents to which the message is addressed (value *all* when the message is broadcasted to all agents), and the $content$ of the message (more details below). This function returns an integer id which corresponds to a unique identifier defining a strict ordering among messages sent by the agent (the id of a message is strictly greater than the id of all messages sent previously). If src denotes the agent sending the message, then each agent in $dstList$ receives message $m = (src, id, content)$.

The content of the message can be of four types:

- *UPDATE*(v, w, b, d): when the agent sending the message indicates that bound $b \in \{LB, UB\}$ of edge $v \rightarrow w$ has been updated to value d;
- *RELAX*(v, b) (used in DIP-L only): when the agent sending the message orders to relax bound b of vertex v;

- *RELAXED*(v, b) (used in DIP-L only): when the agent sending the message indicates that bound b of vertex v has been relaxed as requested;

- *CONSISTENCY*(c, id) (used in CIP and DIP-L): consistency information, where c is a Boolean taking value $true$ when the agent sending the message estimates that the MaSTN is consistent, and value $false$ otherwise; identifier id indicates that for computing this consistency information, the agent sending the message has taken into account all messages received up to identifier id (included).

Each agent also maintains several data structures:

- *MsgRec*: a FIFO list containing messages received and not processed yet;

- *Disturbs*: a FIFO list containing local disturbances not processed yet;

- Rvs: a set of triples (v, w, b) describing temporal constraint revisions to be done on the STN manipulated by the agent;

- Rlx: a set of triples (v, w, b) describing bounds which must be relaxed (to $-\infty$ if $b = LB$ and to $+\infty$ if $b = UB$);

- several elements associated with the STN manipulated by the agent: current consistency, current bounds of variables, current bounds labeling edges, propagation chains...

In addition to the `send` function, each agent uses several basic procedures. Procedures `setBound` and `addUpdate`, shown in Alg. 1, are used respectively to update bounds of temporal constraints and to post updates on the STN manipulated by the agent. Procedure `addUpdate` may update the set Rlx of edges relaxed (line 6) or the set Rvs of edges to be revised (line 7), and it actually sets the required bound for the required edge.

Algorithm 1: Procedures `setBound` and `addUpdate`

```
1 Procedure setBound(v,w,b,d)
2   case b = LB: l_{v,w} ← d
3   case b = UB: u_{v,w} ← d

4 Procedure addUpdate(v,w,b,d)
5   d' ← getBound(v, w, b)
6   if d < d' then Rlx ← Rlx ∪ {(v, w, b)}
7   else if d > d' then Rvs ← Rvs ∪ {(v, w, b)}
8   setBound(v, w, b, d)
```

Each agent also uses three procedures whose definition depends on the algorithmic variant chosen:

- `ProcessMessages`, used by the agent when received messages must be processed;

- `ProcessDisturbances`, used by the agent when a change occurs in it owns temporal constraints (constraints in E_L^A);

- `ProcessUpdates`, used when the agent performs incremental reasoning following some updates.

3.4 Centralized Incremental Propagation (CIP)

As said previously, CIP adopts a centralized approach where a *supervisor* agent is aware of the whole MaSTN. This supervisor may be elected among agents, and elected again if the supervisor fails. It can use well-known mono-agent methods to detect inconsistencies. Details concerning CIP are provided in Alg. 2 for the supervisor agent and in Alg. 3 for the slave agents.

Basically, the supervisor processes messages sent by slave agents by simply adding the set of updates received to the set of updates to be processed (lines 2-5 in Alg. 2). It also maintains, for each slave agent A, a field $lastIdRec(A)$ representing the identifier of the last message from A which has been processed. To process the set of updates induced by local changes or by messages, the supervisor considers the STN $S = (V, E)$ containing all vertices and all temporal constraints of the MaSTN ($V = \cup_{A \in \mathcal{A}} V_A^L$ and $E = (\cup_{A \in \mathcal{A}} E_L^A) \cup E_X$). It uses the `IncrRelax` and `IncrPropag` functions introduced in Section 2 (lines 13 and 14), and then transmits the updates to the appropriate slave agents (lines 16-18). It also sends to the slave agents information regarding the consistency of the MaSTN (lines 20-24). To avoid useless messages, it maintains the content of the last consistency messages sent.

As for slave agents (Alg. 3), each slave agent A receives messages from the supervisor and updates bounds of its own variables in V_L^A accordingly (line 5). Consistency messages are taken into account only if they were processed by taking into account the last message sent (line 7). Also, each time a local update occurs (change in an edge in E_L^A), it is directly sent to the supervisor and the consistency status is temporary set to unknown (lines 10 to 13).

Algorithm 2: CIP - Supervisor's procedures

```
1  Procedure ProcessMessages()
2    while MsgRec ≠ ∅
3      PickNext (src, id, UPDATE(v, w, b, d)) from MsgRec
4      addUpdate(v, w, b, d)
5      lastIdRec(src) ← id
6    ProcessUpdates()

7  Procedure ProcessDisturbances()
8    while Disturbs ≠ ∅
9      PickNext UPDATE(v, w, b, d) from Disturbs
10     addUpdate(v, w, b, d)
11   ProcessUpdates()

12 Procedure ProcessUpdates()
13   Rvs ← Rvs ∪ IncrRelax(Rlx)
14   BoundUpdates ← IncrPropag(Rvs)
15   Rlx ← ∅; Rvs ← ∅
16   foreach (v, b) ∈ BoundUpdates
17     A ← owner(v)
18     lastIdSent(A) ←
       send(A, UPDATE(v0, v, b, getBound(v, b)))
19   c ← getConsistency()
20   foreach A ∈ A
21     s ← CONSISTENCY(c, lastIdRec(A))
22     if lastConsSent(A) ≠ s then
23       send(A, s)
24       lastConsSent(A) ← s
```

3.5 Distributed Incremental Propagation with Global information sharing (DIP-G)

DIP-G is an approach which avoids having a centralized design. In DIP-G, every agent is aware of the whole MaSTN. More precisely, every agent maintains an STN containing all vertices and all edges of the MaSTN. This way, every agent checks independently whether the MaSTN is consistent or not. In this version, agents only have to send the detected disturbances to all other agents. The description of DIP-G provided in Alg. 4 is quite straightforward: each time messages are received, the updates they contain are pro-

Algorithm 3: CIP - Slaves's procedures

```
1  Procedure ProcessMessages()
2      while MsgRec ≠ ∅
3          PickNext m = (src, id, content) from MsgRec
4          if content = UPDATE(v0, v, b, d) then
5              setBound(v, b, d)
6          else if (content = CONSISTENCY(c, idRec))
7              if lastIdSent = idRec then setConsistency(c)

8  Procedure ProcessDisturbances()
9      if Disturbs ≠ ∅ then
10         setConsistency(unknown)
11         while Disturbs ≠ ∅
12             Pick u from MsgRec
13             lastIdSent ← send(supervisor, u)

14 Procedure ProcessUpdates(): empty
```

cessed, and each time there is a change in a local edge, this change is forwarded to all other agents (no privacy). Each agent processes updates as in CIP (see lines 13 to 15). In DIP-G, agents do not exchange any consistency message, they exchange raw information instead, that is raw changes on local edges.

Algorithm 4: DIP-G procedures

```
1  Procedure ProcessMessages()
2      while MsgRec ≠ ∅
3          PickNext m = UPDATE(v, w, b, d) from MsgRec
4          addUpdate(v, w, b, d)
5      ProcessUpdates()

6  Procedure ProcessDisturbances()
7      while Disturbs ≠ ∅
8          PickNext u = UPDATE(v, w, b, d) from Disturbs
9          addUpdate(v, w, b, d)
10         send(all, u)
11     ProcessUpdates()

12 Procedure ProcessUpdates()
13     Rvs ← Rvs ∪ IncrRelax(Rlx)
14     BoundUpdates ← IncrPropag(Rvs)
15     Rlx ← ∅;  Rvs ← ∅
```

3.6 Distributed Incremental Propagation with Local information sharing (DIP-L)

Privacy may be a critical feature in applications in which agents want to share the minimum amount of information. We introduce DIP-L to handle such applications. In DIP-L (Alg. 5), each agent A is only aware of its local STN $S^A = (V_L^A \cup V_X^A, E_L^A \cup E_X^A)$. When an agent detects a disturbance on some of its local edges in E_L^A, it propagates them at the level of its local STN and sends only information related to modified frontier vertices to concerned agents. The latter will perform propagation on their own part of the problem, they may potentially send new updates to the original agent, which itself can then send other messages to other agents concerning other frontier vertices. This way, temporal constraints are propagated inside the MaSTN in a distributed and local way. Agents also share the view that they have concerning the consistency status of their own part of the problem.

The main difficulty for defining DIP-L is related to the fact that performing constraint tightening and constraint relaxation in parallel for STNs can lead to wrong results. On MaSTN, the situation is even worse because it can be shown that performing tightening and relaxation in parallel may lead to infinite cycles of messages. As updates can occur concurrently inside the network of agents, it is therefore required to add mechanisms that prevent agents for propagating updates corresponding to tightenings before all required relaxations are finished. To do this, we maintain, at the level of each agent A, data structures called *relaxation waiting lists*. More precisely, for each vertex w and each bound type b, we maintain a set $RlxWait(w, b)$ to represent the set of frontier nodes v of A contained in the tree of propagation chains rooted in w for bound b, and whose relaxation end must be waited for before tightening bound b of w. In this case, w is called the source of the relaxation of v for bound b, denoted by $w = rlxSrc(v, b)$. Such aspects are not taken into account in algorithm DIPPC [1], which only handles constraint tightening.

In DIP-L, four types of messages can be exchanged between agents: *UPDATE*(v_0, v, b, d) (line 4) when an agent receives an update on the bound of an external vertex, *RELAX*(v, b) (line 6) when an agent receives an order to relax bound b of an external vertex b, *RELAXED*(v, b) (line 8) when an agent receives a confirmation that all vertices lying in the tree of propagation chains rooted in v were relaxed for bound b, and *CONSISTENCY*(c, lid) (line 14) when an agent receives the consistency status c of the local STN of another agent, and when the last message taken into account by this other agent has identifier lid. In particular, when a relaxation waiting list becomes empty (line 11), a relaxation confirmation message is sent to the appropriate agent (line 13).

Procedure `ProcessDisturbances` (lines 19 to 23) simply adds all local disturbances to the set of updates to be processed.

Procedure `ProcessUpdates` involves two parts: one part dedicated to constraint relaxation (lines 25 to 37), and one part dedicated to constraint tightening (lines 38 to 49). The relaxation part sends relaxation requests when frontier vertices are relaxed (lines 31- 32), while the tightening part sends updates to neighboring agents when time bounds of frontier nodes change (lines 41 to 43), as well as consistency information to all agents (lines 45 to 49).

3.7 Distributed Incremental Propagation with Macro-information sharing (DIP-M)

The main issue with the previous method is that as each agent is only aware of its own local STN, a lot of messages may be exchanged between agents before obtaining the right time-bounds for the variables. Typically, an agent sending an update message to a neighbor cannot predict if this message will further impact its own local STN. This increases both the execution time and the message load. To address these issues, we propose DIP-M, an algorithmic variant in which more information is shared between agents while keeping some kind of privacy.

In DIP-M, each agent A builds a global view of its local STN and shares this global view with all other agents. This global view is called the *local macro-STN* of A. Formally, it is an STN $S_M^A = (V_M^A, E_M^A)$ where:

- V_M^A is the set composed of all frontier variables of A plus the reference-point v_0 ($V_M^A = V_F^A \cup \{v_0\}$);
- E_M^A is a set of edges such that any solution of S_M^A can be extended to a solution of local STN S_L^A, and conversely any solution of S_L^A can be projected to a solution of S_M^A.

The local macro-STN associated with A provides a global view, restricted to frontier nodes, of the set of solutions of the local STN of A. This global view allows the other agents to predict how external time-point variables owned by A react to changes.

Algorithm 5: DIP-L procedures

Procedure `ProcessMessages()`
  while *MsgRec* $\neq \emptyset$
    PickNext $m = (src, id, content)$ from *MsgRec*
    if $content =$ *UPDATE*(v_0, v, b, d) **then**
      `addUpdate`(v_0, v, b, d)
    else if $content =$ *RELAX*(v, b) **then**
      $Rlx \leftarrow Rlx \cup \{(v_0, v, b)\}$
    else if $content =$ *RELAXED*(v, b) **then**
      $w \leftarrow rlxSrc(v, b)$
      $RlxWait(w, b) \leftarrow RlxWait(w, b) \setminus \{(v, src)\}$
      if $(RlxWait(w, b) = \emptyset) \wedge (w \in V_X^{this})$ **then**
        $A \leftarrow$ *owner*(w)
        $lastIdSent(A) =$ `send`$(A,$ *RELAXED*$(w, b))$
    else if $content =$ *CONSISTENCY*$(c, idRec)$ **then**
      if $lastIdSent(src) = idRec$ **then**
        $consistency(src) \leftarrow c$
    $lastIdRec(src) \leftarrow id$
  `ProcessUpdates()`

Procedure `ProcessDisturbances()`
  while *Disturbs* $\neq \emptyset$
    PickNext *UPDATE*(v, w, b, d) from *Disturbs*
    `addUpdate`(v, w, b, d)
  `ProcessUpdates()`

Procedure `ProcessUpdates()`
  foreach $(v, b) \in Rlx$
    $RlxFront \leftarrow V_F^{this} \cap$ `getPropagTree`(v, b)
    if $RlxFront \neq \emptyset$ **then**
      foreach $w \in RlxFront$
        $rlxSrc(w, b) \leftarrow v$
      $RlxWait(v, b) \leftarrow \{(w, A) \mid w \in RlxFront \cap V_X^A\}$
      foreach $(w, A) \in RlxWait(v, b)$
        $lastIdSent(A) \leftarrow$ `send`$(A,$ *RELAX*$(w, b))$
    else if $v \in V_X^{this}$ **then**
      $A \leftarrow$ *owner*(v)
      $lastIdSent(A) \leftarrow$ `send`$(A,$ *RELAXED*$(v, b))$
  $Rvs \leftarrow Rvs \cup$ `IncrRelax`(Rlx)
  $Rlx \leftarrow \emptyset$
  if $\forall (v, b) \in V^{this} \times \{LB, UB\},\ RlxWait(v, b) = \emptyset$ **then**
    $BoundUpdates \leftarrow$ `IncrPropag`(Rvs)
    $Rvs \leftarrow \emptyset$
    foreach $(v, b) \in BoundUpdates \mid v \in V_F^{this}$
      foreach $A \in \mathcal{A} \mid v \in V_X^A$
        `send`$(A,$ *UPDATE*$(v_0, v, b,$ `getBound`$(v, b)))$
    $c \leftarrow$ `getConsistency()`
    foreach $A \in \mathcal{A}$
      $s \leftarrow$ *CONSISTENCY*$(c, lastIdRec(A))$
      if $lastConsSent(A) \neq s$ **then**
        $lastIdSent \leftarrow$ `send`(A, s)
        $lastConsSent(A) \leftarrow s$

The *macro-STN* $S_M = (V_M, E_M)$ associated with an MaSTN involving a set of agents $\mathcal{A}$ is then defined as the union of all local macro-STNs of all agents, plus the set of external edges between agents ($V_M = \cup_{A \in \mathcal{A}} V_M^A$ and $E_M = E_X \cup (\cup_{A \in \mathcal{A}} E_M^A)$). Fig. 3(a) gives the structure of the macro-STN associated with the MaSTN described in Fig. 1.

In order to build the local macro-STN S_M^A associated with each agent A, we use a kind of all-pairs shortest paths algorithm which eliminates non-frontier vertices one by one from the local STN S_L^A. Eliminating a non-frontier vertex v consists in computing all intervals of possible distances between vertices in the neighborhood of v, that is between vertices which are linked to v by a temporal constraint. More precisely, for any two vertices u, w in the neighborhood of v, eliminating v means updating the lower and upper bounds of edge $e = \{u, w\}$ by:

$$l_e = \max(l_e, l_{\{u,v\}} + l_{\{v,w\}}) \quad (1)$$

$$u_e = \min(u_e, u_{\{u,v\}} + u_{\{v,w\}}) \quad (2)$$

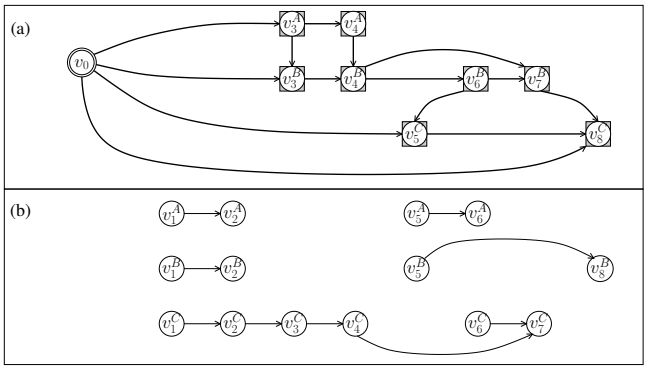


Figure 3: (a) Structure of the macro-STN obtained from the example of Figure 1; (b) set of clusters of the macro-STN: non-frontier time-point variables owned by agents C all belong to a single cluster, whereas for agents A and B they are partitioned into two distinct clusters

Based on this elimination scheme, we proceed as follows to build the local macro-STN S_M^A and to incrementally maintain it during changes in temporal constraints:

- initially, the set of non-frontier variables $V_L^A \setminus V_F^A$ owned by A is partitioned into a set of clusters C^A such that every path in S_L^A between two variables v, w belonging to distinct clusters goes through a frontier node in V_F^A (*i.e.* v and w are separated by frontier vertices); formally, C^A is the set of connected components obtained when removing frontier vertices from S_L^A (see Fig. 3(b)); we then compute, for each cluster $c \in C^A$ and for each frontier vertices v, w connected to c by temporal constraints, the minimum and maximum temporal distances $l_{\{v,w\},c}$ and $u_{\{v,w\},c}$ between v and w owing to constraints involved in cluster c; these distances are provided by the elimination of all vertices belonging to cluster c; the bounds labeling the macro-edge between v and w are then obtained by combining all bounds $l_{\{v,w\},c}/u_{\{v,w\},c}$ provided all clusters c linked to v and w, plus the potential direct edge between v and w;

- in case of changes concerning some local edges, it suffices to apply the elimination procedure again, but restricted to clusters impacted by local changes; these clusters are those which contain variables involved in updated temporal constraints; on this point, it is worth noting that a change in a single local edge has an influence on at most one cluster; the updated labels of macro-edges can then be obtained as previously; such a procedure can be shown to be valid, and more incremental versions could be proposed to avoid eliminating again every variable in every impacted cluster.

Thanks to the macro-STN, each agent can reason over the whole MaSTN in case of changes on its own local STN, without the need to wait messages from others agents. More precisely, in DIP-M, each agent A performs temporal reasoning based on the STN resulting from both its local STN S_L^A and from the part of the macro-STN corresponding to other agents (see Fig. 2(d)). The procedures used

in DIP-M are shown in Alg. 6. The main difference with DIP-G is that instead of transmitting raw updates on internal edges, DIP-M transmits updates on macro-edges (line 5). On this point, DIP-M uses a function called `IncrComputeMacroEdges` to incrementally update its local macro-STN following updates on its local edges (line 3). This function returns the set of updates on macro-edges in E^A_M. In terms of privacy, each agent only reveals distance constraints between its frontier nodes.

Algorithm 6: DIP-M procedures

1 **Procedure** `ProcessMessages()`: *identical to DIP-G*

2 **Procedure** `ProcessDisturbances()`
3 $MacroUp \leftarrow$ `IncrComputeMacroEdges`(*Disturbs*)
4 **foreach** $u \in MacroUp$
5 `send`(all, u)
6 **while** *Disturbs* $\neq \emptyset$
7 PickNext $u =$ *UPDATE*(v, w, b, d) from *Disturbs*
8 `addUpdate`(v, w, b, d)
9 `ProcessUpdates`()

10 **Procedure** `ProcessUpdates()`: *identical to DIP-G*

4. THEORETICAL ANALYSIS

4.1 Complexities

We consider two main metrics to evaluate all four algorithms: a) the *message efficiency* which is the number of sent messages divided by the number of disturbances (relaxation or tightening) and b) the *total computation efficiency* which is the sum of constraint checks performed by all agents divided by the number of constraint checks that would be performed by applying the `IncrPropag` function on the whole MaSTN (which is similar to making the propagation in a classical STN).

CIP is the best regarding the total computation time on the network; computations are indeed centralized in the supervisor agent, the slave agents being not performing any computation. Each disturbance coming from a slave agent will lead to sending one message, from the agent to the supervisor. In return, the supervisor will send at worst one message per vertex to each vertex owner, leading to a communication efficiency of $O(|V|)$ messages.

DIP-G is the best regarding the worst-case message complexity: the agent owning the disturbance will send a fixed number of message to other agents. However the total computation efficiency of DIP-G is always N times more than CIP: DIP-G sends updates no matter how relevant these updates are; in the extreme case where all agents are totally independent DIP-G would still send each and every update.

DIP-L is the best regarding the privacy of each agent information. The worst-case message efficiency and worst-case computation efficiency are $O(|E_X|^N)$. However in realistic cases where messages are roughly received in the same order they were sent we can expect a message efficiency linear in the number of external constraints and a computation efficiency close to CIP.

DIP-M realizes a trade-off between the number of messages, the total computation cost and the privacy of each agent. The worst-case message efficiency is $O(w^2)$, where w is the size of the biggest cluster that can be found in the MaSTN. The worst-case computation efficiency is linear in the number of agents as for DIP-G. Contrarily to DIP-G, DIP-M does not send an edge that would not affect distances between frontier vertices.

These results are summarized in Table 1.

Table 1: Worst-case complexities for the four algorithms

metric	CIP	DIP-G	DIP-L	DIP-M
messages	$\|V\|$	$O(1)$	$O(\|E_X\|^N)$	w^2
computation	1	N	$O(\|E_X\|^N)$	N

4.2 Robustness against intermittent communications

During execution, communications between agents can be intermittent, meaning that at a time t, two agents A and B cannot send any message to each other. The algorithms perform differently in this case.

DIP-G and DIP-M can immediately propagate a disturbance and establish if the MaSTN is still consistent without having to wait other agents messages. CIP can also do as well as soon as the disturbances send by the slave agents are received. This property allow these three algorithms to be as robust as possible against loses of communications.

In CIP, if communication is lost between a slave agent and its supervisor, this slave agent can no longer receive any update and the supervisor cannot take into account any change that would happen to the slave agent. However, the rest of the network can still work as usual. In a similar fashion, if communication is lost between some agents in DIP-G and DIP-M, agents which are still connected can still work by themselves.

However agents in DIP-L lack information on the MaSTN to make complete propagation. If communication is lost between two agents who shared a synchronization constraint, propagation is prematurely stopped along this constraint. The most concerning issue is that in this case, agents cannot even propagate their own disturbances properly and consequently can be unable to answer if the STN is consistent or not.

5. EXPERIMENTS

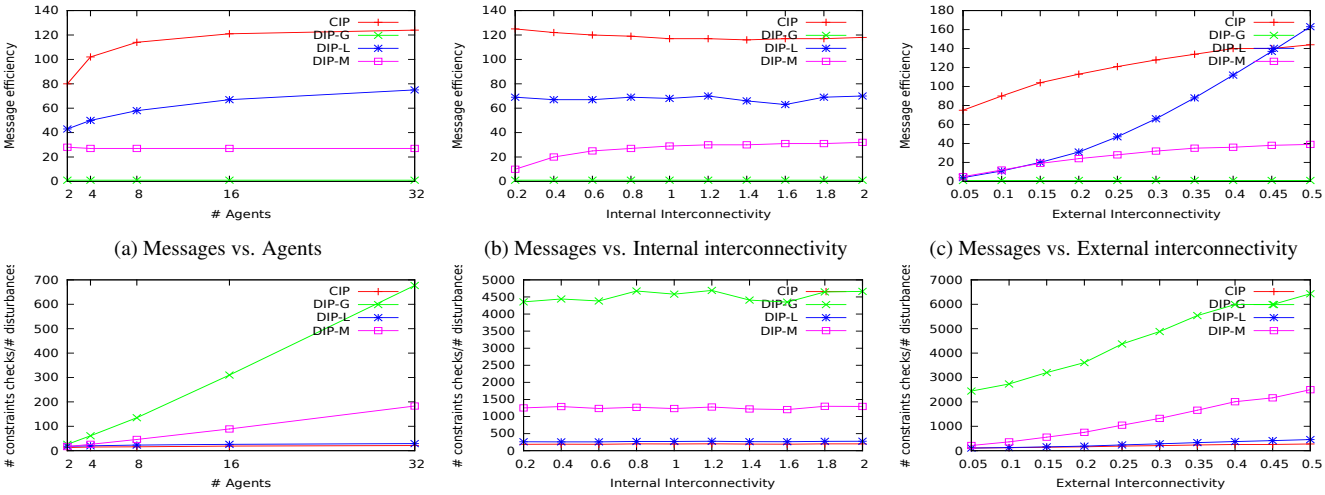
5.1 Experimental setup

We made an evaluation of the the four proposed algorithms on a set of random benchmarks. These benchmarks were generated in two steps: (1) generation of the MaSTN, and (2) generation of a scenario on this MaSTN.

MaSTN generation takes as parameters the number of agents ($\mathcal{N}$), the number of vertices per agent ($\mathcal{V}$), the number of local ($\mathcal{L}$) and external ($\mathcal{E}$) edges per agent. The generation is *structured* in the sense the each local STN S^A_L is made of a main chain, i.e., $S^A_L = (V^A_L, E^A_L)$ with $V^A_L = \{v^A_i\}_{1 \leq i \leq \mathcal{V}}$ and $\forall i, (v^A_i, v^A_{i+1}) \in E^A_L$ (leading to $\mathcal{V} - 1$ edges). The other $\mathcal{L} - (\mathcal{V} - 1)$ local edges are randomly generated between local vertices. Finally, $\mathcal{E}$ external edges are generated between agents. With each edge generation, we generate a value for the upper and lower bounds, while ensuring that the global MaSTN is still consistent.

Then, for each MaSTN instance, a set of scenarios is generated. Each scenario is made of $\mathcal{N} \times \mathcal{V}$ steps (one step per agent timepoint variable). Each step is described by an agent, a local edge of this agent, and the new value of the edge constraint. Most of the time, this new value is taken within the edge bounds (leading to a tightening of the constraint), but we sometimes draw a value outside the bounds (leading to a relaxation). In our experiment, we decided to draw one relaxation for nine tightenings.

For each set of parameters, we generated ten MaSTN instances and then ten scenarios. We have then executed each algorithm (CIP,



(a) Messages vs. Agents (b) Messages vs. Internal interconnectivity (c) Messages vs. External interconnectivity

(d) Computation vs. Agents (e) Computation vs. Internal interconnectivity (f) Computation vs. External interconnectivity

Figure 4: Results for the message metric (top) and computation metric (bottom)

DIP-G, DIP-L, DIP-M) on each scenario by running one process per agent. Each agent generates its own disturbances (based on the scenario description) and propagates the new constraints according to each algorithm. We used arc-consistency algorithms [4, 3] to propagate constraints. We then measured the number of messages exchanged between agents as well as the number of constraints checked by each agent for each propagation. Results are analyzed in the next section.

5.2 Results

Fig. 4 presents the results of experiments and compares the four algorithms. Message efficiency is the number of sent messages per disturbance. The internal interconnectivity is $\frac{\mathcal{L}-(\mathcal{V}-1)}{\mathcal{V}}$, i.e., the number of local edges in addition to the main chain, per vertex (0 means that local STNs are only made of a chain; 2 means that each vertex is connected to 3 other vertices). The external interconnectivity is $\frac{\mathcal{E}}{\mathcal{V}}$, i.e. the ratio of frontier vertices (0 means no frontier vertex, the local STNs being independent; 0.5 means that half of the vertices are frontier vertices).

Fig. 4a and 4d show the message efficiency and the constraint checks when the number of agents varies from 2 to 32. All four algorithms scale well regarding messages, and differences between algorithms are not really significant. The slight increase seen in CIP and DIP-L comes from longer propagation chains that appear in larger MaSTN. Regarding the number of constraint checks, CIP clearly outperforms other algorithms as it performs computations in a centralized way. DIP-L also scales well as no computation is made twice (each agent being responsible of its own constraints). DIP-G and DIP-M are however linearly increasing due to the redundant calculations made on several agents.

Fig 4b and 4e show that the local structure of STNs (local STNs having more or less edges) has almost no influence on the number of messages or constraint checks. The only interesting thing to note is that DIP-M performs particularly well on less internally connected MaSTNs, due to smaller clusters.

DIP-G sends exactly the same number of messages no matter what the external connectivity is, as shown in Fig. 4c, and outperforms significantly other algorithms for high external interconnectivity. The increase of messages sent by CIP is a consequence of increased dependency between agents, meaning the higher the external coupling the higher number of time-point variables will be affected by an edge update. The most interesting point is how DIP-L and DIP-M behave: DIP-L message performance is directly affected by the number of external constraints. DIP-M in comparison scales only partially with external coupling and rapidly stabilizes. At extreme external coupling (not show in this paper), DIP-M curve decreases to match DIP-G performances, due to the fact that with really high coupling the macro-STN tends to the MaSTN. However, Fig. 4f shows that DIP-M computation efficiency become worse on more externally coupled MaSTNs, due to a higher amount of clusters, then requiring more computational work to calculate macro-edges. All three other algorithms have a nearly similar rising rate caused by longer propagation in denser networks.

In conclusion DIP-L is particularly adapted to sparse MaSTNs where agents do not need a lot of information from their neighbors, while DIP-M offers the best overall performances on dense MaSTNs thanks to its macro-information sharing.

6. CONCLUSION AND PERSPECTIVES

In this paper, we proposed four different algorithms to maintain consistency on MaSTN: 1) CIP, a centralized algorithm that has good performance but is sensitive to intermittent communications; 2) DIP-G, a naive but efficient distributed algorithm which offers a constant message complexity and robustness against loses of communication; 3) DIP-L, which reproduces the behavior of mono-agent propagation algorithm over the MaSTN while keeping as much privacy as possible between agents; 4) DIP-M, which aims for a trade-off between DIP-G and DIP-L by reasoning over what agents need to be able to propagate efficiently their constraints. We analyzed these algorithms both on a theoretical point of view and on a set of random benchmarks. We concluded that DIP-L and DIP-M are the best algorithms, the former being more adapted to sparse MaSTNs while the later is more adapted to dense MaSTNs.

In the future, we intend to extend the macro-STN framework to others STN variants such as STNUs [9] and TSTNs [14]. These extensions will provide us more accurate execution models of the plan, as we will be able to integrate uncontrollable time-point variables and time-varying constraints. We are also preparing a real robotic robot experiment with a team of ground and air robots for an exploration mission.

REFERENCES

[1] J. C. Boerkoel, L. Planken, R. Wilcox, and J. A. Shah. Distributed Algorithms for Incrementally Maintaining Multiagent Simple Temporal Networks. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, Rome, Italy, 2013.

[2] R. Cervoni, A. Cesta, and A. Oddi. Managing Dynamic Temporal Constraint Networks. In *Int. Conf. on Artificial Intelligence Planning Systems (AIPS)*, Chicago, IL, USA, 1994.

[3] A. Cesta and A. Oddi. Gaining Efficiency and Flexibility in the Simple Temporal Problem. In *Int. Symposium on Temporal Representation and Reasoning (TIME)*, Key West, FL, USA, 1996.

[4] E. Davis. Constraint Propagation with Interval Labels. *Artificial Intelligence*, 32(3):281–331, 1987.

[5] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[6] M. Di Rocco, F. Pecora, and A. Saffiotti. When Robots Are Late: Configuration Planning for Multiple Robots with Dynamic Goals. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013.

[7] S. Lemai and F. Ingrand. Interleaving Temporal Planning and Execution in Robotics Domains. In *National Conference on Artificial Intelligence (AAAI)*, San Jose, CA, USA, 2004.

[8] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. A Deliberative Architecture for AUV Control. In *Int. Conf. on Robotics and Automation (ICRA)*, Pasadena, CA, USA, 2008.

[9] P. H. Morris, N. Muscettola, and T. Vidal. Dynamic Control Of Plans With Temporal Uncertainty. In *Int. Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, WA, USA, 2001.

[10] L. E. Parker. Distributed intelligence : Overview of the field and its application in multi-robot systems. *Journal of Physical Agents*, 2, 2008.

[11] L. Planken, M. de Weerdt, and N. Yorke-Smith. Incrementally Solving STNs by Enforcing Partial Path Consistency. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, Toronto, Canada, 2010.

[12] L. R. Planken, M. M. de Weerdt, and R. P. van der Krogt. P3C: A New Algorithm for the Simple Temporal Problem. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, Sydney, Australia, 2008.

[13] C. Pralet and C. Lesire. Deployment of mobile wireless sensor networks for crisis management: A constraint-based local search approach. In *Int. Conf. on Principles and Practice of Constraint Programming (CP)*, Lyon, France, 2014.

[14] C. Pralet and G. Verfaillie. Time-Dependent Simple Temporal Networks: Properties and Algorithms. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, Portsmouth, NU, USA, 2014.

[15] L. Xu and B. Y. Choueiry. A New Efficient Algorithm for Solving the Simple Temporal Problem. In *Int. Symposium on Temporal Representation and Reasoning (TIME)*, Cairns, Queensland, Australia, 2003.