Gestion des Réseaux Temporels Simples Multi-agents dynamiques

G. Casanova
Guillaume.Casanova@onera.fr

C. Lesire
Charles.Lesire@onera.fr

C. Pralet
Cedric.Pralet@onera.fr

Onera – The French Aerolab
2 Avenue Édouard Belin
31000 Toulouse, France

Résumé

La réalisation de plans d'activités par plusieurs agents est généralement soumise à un ensemble de contraintes temporelles, impliquant notamment des contraintes de synchronisation entre agents. L'ensemble des contraintes temporelles d'un plan distribué peut être représenté en utilisant une structure Multi-agent Simple Temporal Network *(MaSTN). Dans ce papier, nous considérons le problème du maintien de la cohérence temporelle des plans distribués durant l'exécution, où les contraintes temporelles peuvent être modifiées. Pour cela, nous proposons de nouveaux algorithmes incrémentaux pour gérer les MaSTN dynamiques. Nous analysons les performances de ces algorithmes lorsque les communications sont intermittentes.*

Mots-clés : *Planification Multi-Agents, Coordination, Exécution*

Abstract

The realization of plans of activities by several agents is usually subject to a set of temporal constraints, including synchronization constraints between agents. 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. We propose new incremental algorithms for managing dynamic MaSTNs, and we analyze the performance of these algorithms when communications are intermittent.

Keywords: *Multi-Agent Planning, Coordination, Execution*

1 Introduction

Les applications en robotique autonome telles que l'exploration automatique de larges zones dangereuses peuvent avoir des performances améliorées par l'utilisation de plusieurs robots [7], en exploitant au maximum les capacités hétérogènes des robots pour distribuer au mieux les différentes tâches de la mission. Un autre point est que les systèmes multirobots évoluent généralement dans des environnements complexes et restreints. Dans de tels environnements, la communication entre les robots est généralement chaotique, menant à des communications retardées ou même intermittentes. Dans le but d'avoir une équipe robuste à la fois aux environnements dynamiques et aux communications intermittentes, il semble indispensable d'implémenter un système autonome distribué.

Dans ce contexte, il est nécessaire d'assurer malgré les communications intermittentes une cohérence temporelle entre les plans exécutés par les robots. Pour gérer ces aspects temporels, les *Simple Temporal Network* (STN) sont souvent utilisés en robotique pour représenter des plans [5, 10], et des techniques sont disponibles pour maintenir ces STN durant l'exécution lorsque surviennent des changements tels que des retards dans la réalisation des tâches. Dans le système que nous considérons, il est cependant impossible d'avoir un STN unique pour toute l'équipe maintenu de façon centralisée. A la place, il est possible de se tourner vers le formalisme des *Multi-agent STN* (MaSTN) [1], dans lequel chaque agent connaît seulement les contraintes temporelles présentes dans son propre plan. Plusieurs algorithmes existent pour gérer les MaSTN (DIPPC et DIΔSTP [1]), cependant ils ne sont pas directement adaptés à notre besoin car ils n'ont pas été prévus pour être robustes aux communications intermittentes.

Dans ce papier, nous proposons quatre algorithmes pour maintenir la cohérence dans un MaSTN : CIP, un algorithme centralisé basé sur un agent superviseur gérant entièrement la cohérence du MaSTN ; DIP-G, un algorithme distribué basé sur le partage de toutes les informations entre agents ; DIP-L, qui reproduit la

propagation mono-agent sur le MaSTN tout en maintenant le maximum de confidentialité entre les agents ; et DIP-M, qui cherche à améliorer les performances en réduisant les impératifs de confidentialité. La section 2 fait quelques rappels sur les STN et les MaSTN. La section 3 présente les quatre algorithmes proposés. Ces algorithmes sont analysés dans la section 4. Enfin, la section 5 compare sur des scénarios générés aléatoirement les quatre algorithmes selon le nombre de messages échangés et les calculs nécessaires pour propager des perturbations.

2 Contexte

2.1 Simple Temporal Network (STN)

Un STP (*Simple Temporal Problem* [4]) est une paire $S = (V, E)$ composée d'un ensemble de variables temporelles $V = \{v_1, \ldots, v_n\}$ et d'un ensemble de contraintes temporelles E. Chaque variable $v \in V$ est associée à l'occurrence d'un évènement dans le temps ; une variable spécifique v_0 est habituellement ajoutée à V afin de représenter une position temporelle de référence. Chaque contrainte $e \in E$ prend la forme $v_j - v_i \in [l_{ij}, u_{ij}]$ avec $l_{ij} \in \mathbb{R} \cup \{-\infty\}$ et $u_{ij} \in \mathbb{R} \cup \{+\infty\}$ deux bornes spécifiant respectivement des distances temporelles minimales et maximales entre v_i et v_j. On suppose sans perte de généralité qu'il n'existe pas plusieurs contraintes dans E portant sur les mêmes variables temporelles $\{v_i, v_j\}$. Les contraintes temporelles unaires telles que $v_i \in [a, b]$ peuvent être aisément exprimé par des contraintes de distance par rapport au point de référence (contraintes $v - v_0 \in [a, b]$).

À partir de cette définition, une *solution* à un STP (V, E) est une instanciation de toutes les variables dans V telle que toutes les contraintes temporelles dans E sont satisfaites. Un STP est dit *cohérent* s'il admet une solution, *incohérent* sinon. Pour finir, un STP a une représentation graphique naturelle appelée STN (*Simple Temporal Network*), qui représente chaque variable temporelle dans V par un sommet et chaque contrainte temporelle $v_j - v_i \in [l_{ij}, u_{ij}]$ dans E par un arc $v_i \rightarrow v_j$ étiqueté par $[l_{ij}, u_{ij}]$. Les STN sont intéressants en pratique car de nombreux problèmes qui peuvent être formulés sur les STN sont solubles en temps polynomial, comme la détermination pour chaque variable temporelle v de ses dates d'occurrence au plus tôt et au plus tard dans une solution (voir [4, 3, 11, 9] pour des algorithmes).

Les algorithmes pour raisonner sur les STN ont également été étendus à des contextes dynamiques [2, 8], où les contraintes temporelles peuvent être mises à jour par deux types de modifications : (1) des *contractions*, quand une contrainte temporelle $w - v \geq d$ est mise à jour par $w - v \geq d'$ avec $d' > d$, et (2) des *relâchements*, quand une contrainte temporelle $w - v \geq d$ est mise à jour par $w - v \geq d'$ avec $d' < d$. Dans le premier cas (contraction), on effectue un raisonnement incrémental en maintenant une file de contraintes temporelles à réviser pour recalculer la cohérence du STN ou les dates au plus tôt / au plus tard associées aux variables temporelles. Dans le second cas (relâchement), on effectue un raisonnement incrémental en utilisant des informations enregistrées durant les calculs précédents, telles que les *chaînes de propagation* qui décrivent les causes des bornes actuelles des différentes variables temporelles : si une contrainte temporelle est relâchée et était la cause d'une borne d'une variable temporelle, alors cette borne est réinitialisée ainsi que toutes les bornes qui en découlent [2].

Dans ce qui suit, nous ne détaillons pas ces algorithmes de contraction et de relâchement. Nous supposons seulement que nous avons deux fonctions notées respectivement `IncrRelax`(Rlx) et `IncrPropag`(Rvs). La première prend comme paramètre un ensemble Rlx de relâchements d'arcs représentés par des triplets (v, w, b) (relâchement de la borne $b \in \{LB, UB\}$ de l'arc $v \rightarrow w$) ; elle réinitialise les bornes temporelles des sommets grâce aux chaînes de propagation, et elle renvoie les contraintes temporelles qui doivent être révisées après l'opération de relâchement. La seconde prend comme paramètre un ensemble Rvs de contraintes temporelles à réviser, chacune étant décrite par un triplet (v, w, b) ; elle renvoie un ensemble de paires (v, b) décrivant les bornes temporelles mises à jour par la révision des contraintes temporelles. Nous supposons que les appels à `IncrPropag` et `IncrRelax` actualisent tous les paramètres associés aux STN (cohérence, dates au plus tôt / au plus tard et chaînes de propagation).

2.2 Multi-agent STN (MaSTN)

Les STN ont été étendus au contexte multi-agent, dans lequel les variables temporelles ne sont pas contrôlées par un seul agent mais sont partagées par un ensemble d'agents $\mathcal{A}$. Cette extension est appelée MaSTN (*Multiagent Simple Temporal Network* [1]). Un MaSTN est défini formellement par (1) un ensemble de N STN

locaux, un par agent $A \in \mathcal{A}$, et (2) un ensemble d'arcs E_X connectant ces STN locaux. Le STN local associé à l'agent A, noté S_L^A, est défini par V_L^A l'ensemble de sommets *locaux* possédés par A, et E_L^A l'ensemble d'arcs *locaux* reliant deux sommets locaux $v, w \in V_L^A$. Chaque arc dans E_X représente une contrainte *externe* et relie deux sommets locaux d'agents différents.

En plus de ses arcs locaux, chaque agent A connaît le sous-ensemble des contraintes externes E_X^A qui s'appliquent sur un de ses sommets locaux ($E_X^A = \{\{v, w\} \in E_X | v \in V_L^A\}$). En plus de ses sommets locaux, chaque agent A connaît l'ensemble V_X^A composé des sommets non possédés par A mais impliqués dans E_X^A ($V_X^A = \{v \in V | \exists w \in V_L^A, \{v, w\} \in E_X\}$). Ainsi, l'ensemble des variables temporelles connues par l'agent A est $V^A = V_L^A \cup V_X^A$, et l'ensemble des arcs connus par A est $E^A = E_L^A \cup E_X^A$. De plus, nous définissons pour chaque agent A l'ensemble V_F^A des sommets *frontières* comme étant le sous-ensemble des sommets locaux de A connectés à au moins un sommet externe ($V_F^A = \{v \in V_L^A | \exists \{v, w\} \in E_X\}$). Par la suite, nous notons *owner*(v) l'unique agent possédant la variable temporelle v, c'est-à-dire l'agent A tel que v est compris dans V_L^A.

La figure 1 donne un exemple de MaSTN impliquant trois agents A, B et C. L'agent A (resp. B, C) possède les variables temporelles v_1^A à v_6^A (resp. v_1^B à v_8^B et v_1^C à v_8^C). Dans son plan, l'agent A doit effectuer l'acquisition *acq1*, se recharger grâce à l'agent B puis effectuer une opération de maintenance. L'agent B doit effectuer l'acquisition *acq2*, recharger l'agent A et recevoir des données de l'agent C. Un instrument de réception doit être activé (variable v_5^B) puis désactivé (variable v_8^B) avant et après la réception des données venant de C. L'agent C doit effectuer deux acquisitions (*acq3* et *acq4*) avant de transmettre à l'agent B les données collectées. Certaines contraintes temporelles définissent des bornes sur la durée des activités et sur les temps de transition entre activités. D'autres sont des exigences, comme la contrainte reliant v_2^C et v_5^C qui impose de transmettre l'acquisition *acq3* suffisamment rapidement, ou la contrainte reliant le point de référence v_0 avec la variable temporelle v_8^C qui impose une limite sur la durée du plan de C. Les contraintes externes dans E^X sont représentées par des lignes en pointillés. Elles correspondent aux points de synchronisation entre les agents.

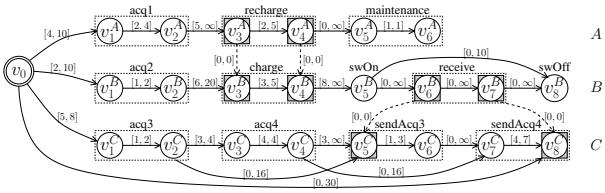


FIGURE 1 – Exemple de MaSTN impliquant trois agents.

3 Algorithmes incrémentaux pour les MaSTN dynamiques

Nous considérons maintenant les MaSTN dynamiques, dans lesquels les contraintes temporelles peuvent être mises à jour selon les informations reçues pendant l'exécution. Par exemple, la durée des activités peut être plus longue ou plus courte que prévue, les fenêtres temporelles disponibles pour la réalisation des activités peuvent changer, ou les agents peuvent replanifier et modifier leurs contraintes temporelles internes. Dans ce cas, notre but est de recalculer incrémentalement, au niveau de chaque agent A, la cohérence du MaSTN ainsi que les dates au plus tôt et au plus tard pour chaque variable possédée par A. Nous étudions quatre variantes algorithmiques pour gérer les MaSTN dynamiques. Ces variantes diffèrent par l'ensemble des contraintes temporelles prises en compte par chaque agent et par le type d'informations partagées entre les agents.

3.1 Hypothèses fondamentales

Par la suite, nous supposons que les contraintes externes dans E_X sont statiques : aucun changement sur leur existence ou leurs étiquettes. La raison à cela est que nous considérons que la modification d'une contrainte temporelle partagée par plusieurs agents doit être gérée par un processus plus complexe de re-synchronisation entre agents, ou par un processus assignant à un agent unique la responsabilité de mise à jour de la contrainte.

D'autre part, nous ne faisons pas d'hypothèse sur le protocole de communication utilisé (broadcast, unicast, multicast...), ou sur comment les données sont émises, routées et acquittées sur le réseau formé par les agents, notamment en cas de communications intermittentes. Les seules hypothèses faites sont qu'il peut exister des délais dans les transmissions et que les messages sont réceptionnés dans le même ordre que leur ordre d'envoi.

3.2 Un premier aperçu des algorithmes

La figure 2 donne un premier aperçu sur le type de connaissances manipulées par chaque agent pour les quatre algorithmes incrémentaux proposés, en reprenant le MaSTN de la figure 1. Les quatre algorithmes sont appelés CIP, DIP-G, DIP-M et DIP-L.

Dans CIP (Centralized Incremental Propagation, fig. 2(a)), un agent superviseur maintient toutes les contraintes temporelles et est responsable des calculs sur ces contraintes. Il reçoit les notifications de modifications des autres agents, et il renvoie les mises à jour concernant la cohérence du MaSTN et les bornes temporelles associées aux variables. Les autres agents ne connaissent que leur propre STN local.

Dans DIP-G (Distributed Incremental Propagation with Global information sharing, fig. 2(b)), chaque agent maintient l'ensemble de toutes les contraintes temporelles présentes dans le MaSTN, même celles qu'il n'est pas supposé connaître. Dès qu'un changement intervient sur un arc local d'un agent, cet agent envoie l'information correspondante à tous les autres agents.

Dans DIP-L (Distributed Incremental Propagation with Local information sharing, fig. 2(c)), chaque agent ne raisonne que sur les contraintes temporelles qu'il est supposé connaître, et les seules données échangées entre les agents sont les bornes temporelles sur les sommets externes.

Dans DIP-M (Distributed Incremental Propagation with Macro information sharing, fig. 2(d)), chaque agent raisonne sur ses contraintes temporelles locales et sur une vue macroscopique des contraintes temporelles des autres agents. Cette vue macroscopique donne des distances temporelles entre sommets externes. Chaque agent est responsable de l'envoi des mises à jour de sa propre vue macroscopique, et ce faisant ne révèle jamais ses contraintes internes.

3.3 Fonctions de base et structures de données utilisées

Avant de détailler les algorithmes, nous introduisons certaines fonctions et structures de données. Pour envoyer des messages, chaque agent utilise une fonction appelée $\texttt{send}(dstList, contenu)$ qui prend comme paramètre une liste $dstList$ d'agents à qui les messages sont destinés (valeur *all* quand le message est diffusé à tous les agents), et le $contenu$ du message. Cette fonction retourne un entier id

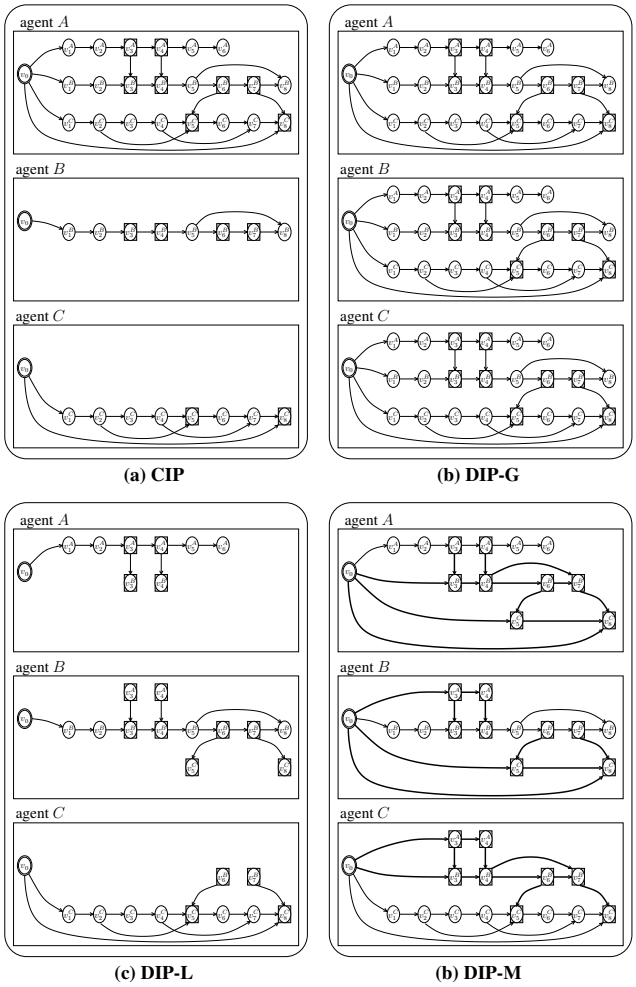


FIGURE 2 – Les quatre algorithmes proposés

qui correspond à un identificateur unique définissant un ordre strict sur les messages envoyés par les agents (l'id d'un message est strictement supérieur aux id de tous les messages envoyés précédemment). Soit src l'agent envoyant le message, alors chaque agent dans $dstList$ reçoit le message $m = (src, id, contenu)$. Le contenu des messages peut être de quatre types :

- *UPDATE*(v, w, b, d) : quand l'agent envoyant le message signale que la borne $b \in \{LB, UB\}$ de l'arc $v \rightarrow w$ a été mise à jour à la valeur d ;
- *RELAX*(v, b) (utilisé dans DIP-L) : quand l'agent émetteur ordonne de relâcher la borne b du sommet v ;
- *RELAXED*(v, b) (utilisé dans DIP-L) : quand l'agent émetteur signale que la borne b du sommet v a bien été relâchée ;
- *CONSISTENCY*(c, id) (utilisé dans CIP et DIP-L) : information de cohérence, où c prend la valeur $vrai$ quand l'agent envoyant le message estime que le MaSTN

est cohérent, et la valeur $faux$ sinon ; l'identificateur id indique que pour calculer cette cohérence, l'agent envoyant le message a pris en compte tous les messages reçus jusqu'à id inclus.

Chaque agent maintient plusieurs structures de données :

- *MsgRec* : une liste FIFO contenant les messages reçus non encore traités ;
- *Disturbs* : une liste FIFO contenant les perturbations locales non encore traitées ;
- Rvs : un ensemble de triplets (v, w, b) décrivant les contraintes temporelles à réviser sur le STN manipulé par l'agent ;
- Rlx : un ensemble de triplets (v, w, b) décrivant des bornes à relâcher ;
- de nombreux objets associés avec le STN manipulé par l'agent : la cohérence courante, les bornes actuelles des variables temporelles, les bornes actuelles des arcs, les chaînes de propagation...

En plus de la fonction `send`, chaque agent utilise plusieurs procédures élémentaires. Les procédures `setBound` et `addUpdate`, données à l'algorithme 1, sont utilisées respectivement pour mettre à jour les bornes des contraintes temporelles et pour poster les mises à jour sur le STN manipulé par l'agent. La procédure `addUpdate` peut mettre à jour l'ensemble Rlx des arcs relâchés (ligne 6) ou l'ensemble Rvs des arcs à réviser (ligne 7). Elle fixe aussi les bornes de l'arc concerné (ligne 8).

Algorithme 1 :

```
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)
```

Chaque agent utilise également trois procédures dont la définition dépend de la variante algorithmique choisie :

- `ProcessMessages`, utilisée par l'agent pour traiter les messages reçus ;
- `ProcessDisturbances`, utilisée par l'agent quand une perturbation survient sur ses propres contraintes temporelles (contraintes dans E_L^A) ;
- `ProcessUpdates`, utilisée quand l'agent effectue un raisonnement incrémental suite à des mises à jour ;

3.4 Centralized Incremental Propagation (CIP)

CIP adopte une approche centralisée dans laquelle un agent *superviseur* connaît l'ensemble du MaSTN. Il peut utiliser des méthodes mono-agent déjà étudiées pour détecter les incohérences. Les détails concernant CIP sont fournis dans l'algorithme 2 pour l'agent superviseur et dans l'algorithme 3 pour les agents esclaves.

Le superviseur traite les messages envoyés par les agents esclaves en ajoutant tout simplement l'ensemble des mises à jour reçues à l'ensemble des mises à jour à traiter (lignes 2-5 dans l'algorithme 2). Il maintient également, pour chaque agent esclave A, un champ $lastIdRec(A)$ représentant l'identificateur du dernier message envoyé par A qui a été traité. Pour traiter l'ensemble des mises à jour induites par les changements locaux ou par les messages, le superviseur considère le STN $S = (V, E)$ contenant tous les sommets et tous les arcs du MaSTN ($V = \cup_{A \in \mathcal{A}} V_A^L$ et $E = (\cup_{A \in \mathcal{A}} E_L^A) \cup E_X$). Il utilise les fonctions `IncrRelax` et `IncrPropag` introduites en à la section 2 (lignes 13 et 14), puis il transmet les mises à jour aux agents esclaves concernés (lignes 16-18). Le superviseur envoie également aux agents esclaves l'information de cohérence du MaSTN (lignes 20-24).

Concernant les agents esclaves (algorithme 3), chaque agent A reçoit du superviseur les messages et les mises à jour des bornes de ses propres variables dans V_L^A (ligne 5). Les messages de cohérence sont pris en compte seulement s'ils ont été traités connaissant le dernier message envoyé (ligne 7). De plus, chaque perturbation locale (changement dans E_L^A) est directement envoyée au superviseur et le statut de cohérence est temporairement mis à inconnu (lignes 10 à 13).

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

DIG-G est une approche évitant une structure centralisée. Dans DIP-G, chaque agent connaît l'ensemble du MaSTN. Ainsi, chaque agent peut indépendamment vérifier la cohérence du MaSTN. Dans cette version, les agents n'ont qu'à envoyer les perturbations détectées à tous les autres agents. La description de DIP-G est donnée à l'algorithme 4 : à chaque réception de message, les mises à jour qu'il contient sont trai-

Algorithme 2 : CIP - procédures du superviseur

```
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
```

Algorithme 3 : CIP - procédures des esclaves

```
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
```

tées, et à chaque modification d'un arc local, ce changement est transmis à tous les autres agents (pas de confidentialité). Chaque agent traite les mises à jour de manière analogue au superviseur dans CIP (voir lignes 13 à 15). Dans DIP-G, les agents n'échangent pas de message de cohérence, ils échangent uniquement les informations brutes sur les arcs locaux.

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

Dans DIP-L (algorithme 5), chaque agent A ne connaît que son propre STN local $S^A = (V_L^A \cup V_X^A, E_L^A \cup E_X^A)$. Quand un agent détecte

Algorithme 4 : Procédures DIP-G

```
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 ← ∅
```

une perturbation sur certains de ses arcs locaux dans E_L^A, il la propage au niveau de son STN local puis envoie uniquement les informations relatives aux sommets frontières modifiés aux agents concernés. Ces derniers vont ensuite propager les changements sur leur propre partie du problème, et éventuellement envoyer de nouvelles mises à jour à l'agent original ou aux autres agents. Ainsi, les contraintes temporelles sont propagées dans le MaSTN de manière distribuée et locale. Les agents partagent également leur vision de la cohérence de leur propre partie du problème.

La principale difficulté pour définir DIP-L vient du fait qu'effectuer des contractions et des relâchements de contraintes en parallèle sur les STN peut mener à des résultats faux. Sur les MaSTN, le problème est amplifié car on peut montrer qu'effectuer des contractions et des relâchements en parallèle peut mener à un cycle infini d'envois de messages. Lorsque les mises à jour surviennent simultanément à l'intérieur du réseau d'agents, il est nécessaire d'ajouter des mécanismes empêchant les agents de propager les mises à jour correspondant à des contractions avant que tous les relâchements soient terminés. Pour cela, nous maintenons, au niveau de chaque agent A, une structure de donnée appelée *liste de relâchements en attente*. Pour chaque sommet w et pour chaque type de borne b, nous maintenons un ensemble $RlxWait(w, b)$ pour représenter l'ensemble de sommets frontières v de A contenus dans l'arbre de chaînes de propagation de racine w pour la borne b, et dont le relâchement doit être terminé avant qu'une contraction puisse être faite sur la borne b de w. Dans ce cas, w est appelé la source du relâchement de v pour la borne b, noté par $w =$

$rlxSrc(v, b)$. Ces aspects ne sont pas pris en compte dans l'algorithme DIPPC [1], qui gère uniquement les contractions de contraintes.

Dans DIP-L, quatre types de messages sont échangés entre les agents : *UPDATE*(v_0, v, b, d) (ligne 4) quand un agent reçoit une mise à jour sur la borne d'un sommet externe, *RELAX*(v, b) (ligne 6) quand un agent reçoit une requête de relâchement sur la borne b d'un sommet externe v, *RELAXED*(v, b) (ligne 8) quand un agent reçoit la confirmation que tous les sommets appartenant à l'arbre de chaînes de propagation de racine v ont été relâchés pour la borne b, et *CONSISTENCY*(c, lid) (ligne 14) quand un agent reçoit l'information de cohérence c du STN local d'un autre agent, et quand le dernier message pris en compte par l'autre agent a pour identificateur lid. En particulier, quand une liste de relâchements en attente devient vide (ligne 11), un message de confirmation de relâchement est envoyé à l'agent concerné (ligne 13).

La procédure `ProcessDisturbances` (lignes 19 à 23) ajoute toutes les perturbations locales à l'ensemble des mises à jour à traiter.

La procédure `ProcessUpdates` se décompose en deux parties : une partie est dédiée aux relâchements de contraintes (lignes 25 à 37), et une autre est dédiée aux contractions de contraintes (lignes 38 à 49). La partie relâchement envoie les requêtes de relâchement quand les sommets frontières sont relâchés (lignes 31-32), alors que la partie contraction envoie les mises à jour aux agents voisins quand les bornes temporelles des sommets frontières changent (lignes 41 à 43), ainsi que les informations de cohérence (lignes 45 à 49).

Algorithme 5 : Procédures DIP-L

```
1  Procedure ProcessMessages()
2    while MsgRec ≠ ∅
3      PickNext m = (src, id, content) from MsgRec
4      if content = UPDATE(v0, v, b, d) then
5        addUpdate(v0, v, b, d)
6      else if content = RELAX(v, b) then
7        Rlx ← Rlx ∪ {(v0, v, b)}
8      else if content = RELAXED(v, b) then
9        w ← rlxSrc(v, b)
10       RlxWait(w, b) ← RlxWait(w, b) \ {(v, src)}
11       if (RlxWait(w, b) = ∅) ∧ (w ∈ V_X^this) then
12         A ← owner(w)
13         lastIdSent(A) = send(A, RELAXED(w, b))
14     else if content = CONSISTENCY(c, idRec) then
15       if lastIdSent(src) = idRec then
16         consistency(src) ← c
17     lastIdRec(src) ← id
18   ProcessUpdates()

19 Procedure ProcessDisturbances()
20   while Disturbs ≠ ∅
21     PickNext UPDATE(v, w, b, d) from Disturbs
22     addUpdate(v, w, b, d)
23   ProcessUpdates()

24 Procedure ProcessUpdates()
25   foreach (v, b) ∈ Rlx
26     RlxFront ← V_F^this ∩ getPropagTree(v, b)
27     if RlxFront ≠ ∅ then
28       foreach w ∈ RlxFront
29         rlxSrc(w, b) ← v
30       RlxWait(v, b) ← {(w, A) | w ∈ RlxFront ∩ V_X^A}
31       foreach (w, A) ∈ RlxWait(v, b)
32         lastIdSent(A) ← send(A, RELAX(w, b))
33     else if v ∈ V_X^this then
34       A ← owner(v)
35       lastIdSent(A) ← send(A, RELAXED(v, b))
36   Rvs ← Rvs ∪ IncrRelax(Rlx)
37   Rlx ← ∅
38   if ∀(v, b) ∈ V^A × {LB, UB}, RlxWait(v, b) = ∅ then
39     BoundUpdates ← IncrPropag(Rvs)
40     Rvs ← ∅
41     foreach (v, b) ∈ BoundUpdates | v ∈ V_F^this
42       foreach A ∈ A | v ∈ V_X^A
43         send(A, UPDATE(v0, v, b, getBound(v, b)))
44     c ← getConsistency()
45     foreach A ∈ A
46       s ← CONSISTENCY(c, lastIdRec(A))
47       if lastConsSent(A) ≠ s then
48         lastIdSent ← send(A, s)
49         lastConsSent(A) ← s
```

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

La principale limitation de la méthode précédente est que comme chaque agent connaît uniquement son STN local, de nombreux messages peuvent être échangés entre les agents avant d'obtenir les bornes des variables temporelles.

Dans DIP-M, des informations supplémentaires sont partagées entre les agents tout en respectant une certaine confidentialité : chaque agent A construit une vue globale de son propre STN local et partage cette vue avec tous les autres agents. Cette vue globale est appelée le *macro-STN local* de A. De manière formelle, c'est un STN $S_M^A = (V_M^A, E_M^A)$ où V_M^A est l'ensemble composé de tous les sommets frontières de A plus le point de référence v_0 ($V_M^A = V_F^A \cup \{v_0\}$), et où E_M^A est un ensemble d'arcs tel que toute solution de S_M^A peut être étendue à une solution du STN local S_L^A, et réciproquement toute solution de S_L^A peut être projetée en une solution de S_M^A.

Le macro-STN local associé à A fournit une vue globale, restreinte aux sommets frontières,

de l'ensemble de solutions du STN local de A. Cette vue globale permet aux autres agents de prédire comme les variables temporelles externes possédées par A réagissent aux perturbations.

Le *macro-STN* $S_M = (V_M, E_M)$ associé à un MaSTN impliquant un ensemble d'agents $\mathcal{A}$ est ensuite défini comme l'union de tous les macro-STN locaux de tous les agents, plus l'ensemble des arcs externes ($V_M = \cup_{A\in\mathcal{A}} V_M^A$ et $E_M = E_X \cup (\cup_{A\in\mathcal{A}} E_M^A)$). La figure 3(a) donne la structure du macro-STN associé au MaSTN de la figure 1.

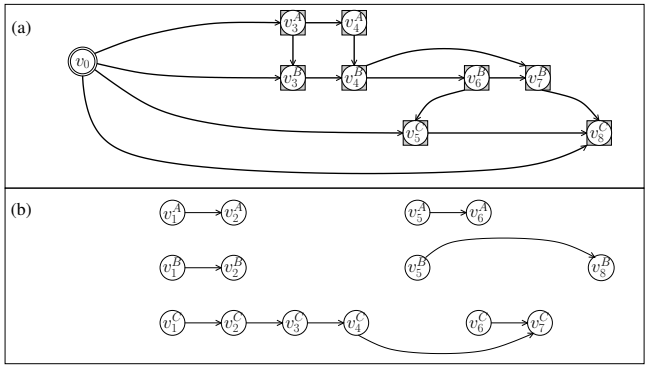


FIGURE 3 – (a) Structure du macro-STN obtenu sur l'exemple de la figure 1 ; (b) ensemble des composantes du macro-STN

Afin de construire le macro-STN local S_M^A associé à chaque agent A, nous utilisons un algorithme semblable à l'algorithme *all-pairs shortest paths* qui élimine un par un les sommets non-frontières du STN local S_L^A. Éliminer un sommet non-frontière consiste à calculer tous les intervalles possibles de distance entre les sommets dans le voisinage de v, c'est-à-dire entre les sommets liés directement à v par une contrainte temporelle. Plus précisément, pour tous sommets u, w dans le voisinage de v, éliminer v signifie mettre à jour les bornes au plus tôt et au plus tard de $e = \{u, w\}$ par :

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

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

À partir de ce processus d'élimination, nous pouvons construire le macro-STN local S_M^A et le maintenir incrémentalement durant les changements de contraintes temporelles. Initialement, l'ensemble des variables non-frontières $V_L^A \backslash V_F^A$ possédées par A sont partitionnées en un ensemble de composantes C^A tel que tout chemin dans S_L^A entre deux variables v, w appartenant à des composantes distinctes passe par un sommet frontière dans V_F^A ; formellement, C^A est l'ensemble de l'ensemble de composantes connexes obtenu en supprimant les sommets frontières de S_L^A (voir la figure 3(b)) ; nous calculons ensuite, pour chaque composante $c \in C^A$ et pour chaque paire de sommets frontières v, w connectés à c par une contrainte temporelle, la distance temporelle minimale et maximale $l_{\{v,w\},c}$ et $u_{\{v,w\},c}$ entre v et w selon les contraintes impliquées dans c ; ces distances sont calculées par élimination de tous les sommets appartenant à la composante c ; les bornes étiquetant les macro-arcs entre v et w sont alors obtenues en combinant toutes les bornes $l_{\{v,w\},c}/u_{\{v,w\},c}$ de toutes les composantes c reliées à v et w, plus éventuellement l'arc direct entre v et w.

Dans le cas de changements concernant certains arcs locaux, il suffit d'appliquer à nouveau la procédure d'élimination, mais restreinte aux composantes touchées par les changements locaux ; ces composantes sont celles contenant des variables temporelles impliquées dans la mise à jour de contraintes temporelles.

Grâce au macro-STN, chaque agent peut raisonner sur l'ensemble du MaSTN dans le cas de changements sur son STN local, sans avoir à attendre des messages de la part des autres agents (voir la figure 2(d)). Les procédures utilisées dans DIP-M sont données à l'algorithme 6. La principale différence avec DIP-G est qu'au lieu de transmettre des mises à jour brutes sur les arcs internes, DIP-M transmet des mises à jour portant uniquement sur les macro-arcs (ligne 5). Sur ce point, DIP-M utilise une fonction appelée `IncrComputeMacroEdges` qui met à jour de manière incrémentale son propre macro-STN local suite aux perturbations survenant sur ses arcs locaux (ligne 3). Cette fonction renvoie l'ensemble des modifications sur les macro-arcs dans E_M^A. En terme de confidentialité, chaque agent ne révèle que les contraintes de distance entre ses sommets externes.

4 Analyse Théorique

Nous considérons deux métriques principales pour évaluer les quatre algorithmes : a) l'*efficacité des messages*, correspondant au nombre de messages envoyés divisé par le nombre de perturbations (relâchements ou contractions) et b) l'*efficacité des calculs*, correspondant au nombre total de vérifications de contraintes effectuées par tous les agents divisé

Algorithme 6 : Procédures DIP-M

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

2  Procedure ProcessDisturbances()
3  |  MacroUp ← IncrComputeMacroEdges(Disturbs)
4  |  foreach u ∈ MacroUp
5  |  |  send(all, u)
6  |  while Disturbs ≠ ∅
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
```

par le nombre de vérifications de contraintes qui seraient effectuées en appliquant la fonction `IncrPropag` sur l'ensemble du MaSTN (ce qui équivaudrait à effectuer une propagation mono-agent sur un STN classique).

Les complexités sont résumés (sans démonstrations) dans la Table 1. w correspond à la taille de la plus grande composante dans le MaSTN.

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

TABLE 1 – Complexité des pires cas pour les quatre algorithmes

DIP-L est sensible aux communications intermittentes car les agents ne possèdent pas les informations nécessaires sur le MaSTN pour effectuer une propagation complète. Les trois autres algorithmes sont moins sensibles grâce à la redondance d'information.

5 Expérimentations

5.1 Méthodologie

Nous évaluons les quatre algorithmes proposés sur un ensemble de problèmes aléatoires. Ces problèmes ont étés générés en deux étapes : génération du MaSTN, puis génération du scénario sur ce MaSTN.

La génération du MaSTN prend comme paramètres le nombre d'agents (N), le nombre de sommets par agent ($\mathcal{V}$), le nombre d'arcs locaux ($\mathcal{L}$) et externes ($\mathcal{E}$) par agent. La génération est *structurée* dans le sens où chaque STN local S_L^A est composé d'une chaîne principale, à laquelle sont ajoutés des arcs locaux entre des sommets aléatoires. Finalement, $\mathcal{E}$ arcs externes sont générés entre les agents. Pour chaque arc, nous générons une valeur pour les bornes tout en assurant que le MaSTN global reste cohérent.

Ensuite, pour chaque instance de MaSTN, un ensemble de scénarios de $N \times \mathcal{V}$ pas est créé. Chaque pas est décrit par la nouvelle valeur d'un arc appartenant à un agent. La plupart du temps, cette nouvelle valeur est prise entre les bornes de l'arc (contraction de contrainte), mais occasionnellement nous tirons une valeur hors de ces bornes (relâchement de contrainte). Dans nos expériences, nous tirons en moyenne un relâchement pour neuf contractions.

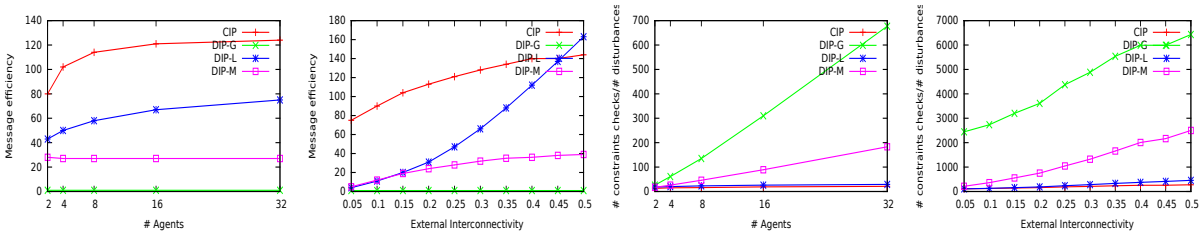
Pour chaque ensemble de paramètres, nous générons dix MaSTN et pour chacun d'eux un scénario correspondant. Nous exécutons ensuite chaque algorithme (CIP, DIP-G, DIP-L, DIP-M) sur chaque scénario en utilisant un processus par agent. Chaque agent génère ses propres perturbations (suivant le scénario) et propage les nouvelles contraintes selon l'algorithme utilisé. Nous mesurons ensuite le nombre de messages échangés entre les agents et le nombre de contraintes vérifiées par chaque agent pour chaque propagation.

5.2 Résultats

La figure 4 présente les résultats permettant de comparer les quatre algorithmes. L'efficacité messages est le nombre moyen de messages envoyés par perturbation. L'interconnectivité externe est $\frac{\mathcal{E}}{\mathcal{V}}$, c.-à-d. le ratio du nombre de sommets frontières par le nombre de sommets (0 signifie que les STN locaux sont indépendants ; 0.5 signifie que la moitié des sommets sont partagés).

Les figures 4a et 4c montrent l'efficacité en termes de messages et en termes de vérifications de contraintes. Au niveau des messages, les quatre algorithmes conservent de bonnes performances lors du passage à l'échelle, et les différences sont peu significatives. La légère augmentation pour CIP et DIP-L vient de l'apparition de plus longues chaînes de propagation dans les MaSTN. Concernant le nombre de vérifications de contraintes, CIP surpasse nettement les autres algorithmes. DIP-L est également très performant grâce à la faible redondance des calculs (chaque agent est responsable de ses propres contraintes). Pour DIP-G et DIP-M, une augmentation linéaire peut être observée à cause de la redondance des calculs effectués sur plusieurs agents.

DIP-G envoie exactement le même nombre de



(a) Messages/Agents (b) Messages/Inter. Externe (c) Calculs/Agents (d) Calculs/Inter. Externe

FIGURE 4 – Résultats pour l'efficacité messages (a, b) et l'efficacité calculs (c, d)

messages quelle que soit l'interconnectivité externe, comme montré à la figure 4b, et il surpasse significativement les autres algorithmes pour les interconnectivités externes les plus hautes. L'augmentation du nombre de messages envoyés par CIP est une conséquence de l'augmentation des dépendances entre les agents, car plus le couplage externe est élevé, plus le nombre de variables temporelles affectées par une mise à jour d'arc est grand. DIP-L et DIP-M présentent des comportements plus intéressants : les performances de DIP-L sont directement affectées par le nombre de contraintes externes. DIP-M n'est cependant que partiellement affecté par le couplage externe et se stabilise rapidement. Cependant, la figure 4d montre que l'efficacité des calculs de DIP-M empire pour les MaSTN plus denses, à cause de l'effort calculatoire requis pour calculer les macro-arcs de composantes plus larges. Les trois autres algorithmes ont un taux d'augmentation similaire, causé par des propagations plus longues dans les réseaux plus denses.

En conclusion, DIP-L est particulièrement adapté aux MaSTN creux où les agents n'utilisent que peu d'informations concernant leurs voisins, tandis que DIP-M offre dans l'ensemble les meilleures performances grâce au partage de macro-informations.

6 Conclusion et Perspectives

Dans ce papier, nous avons proposé quatre algorithmes pour maintenir la cohérence des MaSTN : CIP, DIP-G, DIP-L et DIP-M. Ces algorithmes ont été analysés à la fois d'un point de vue théorique et sur un ensemble de tests générés aléatoirement. Nous concluons que DIP-L et DIP-M sont les algorithmes les plus adaptés pour les cas d'utilisation réels, le premier étant plus adapté aux MaSTN creux tandis que le second est plus adapté aux MaSTN denses.

Dans le futur, nous comptons étendre le cadre macro-STN aux autres variantes de STN telles que les STNU [6]. Ces extensions apporterons des modèles plus précis d'exécution de plans, tout en intégrant les concepts de variables temporelles incontrôlables. Nous préparons également une expérimentation réelle avec une équipe de robots terrestres et aériens pour une mission d'exploration.

Références

[1] J. C. Boerkoel, L. Planken, R. Wilcox, and J. A. Shah. Distributed Algorithms for Incrementally Maintaining Multiagent Simple Temporal Networks. In *ICAPS*, Rome, Italy, 2013.

[2] R. Cervoni, A. Cesta, and A. Oddi. Managing Dynamic Temporal Constraint Networks. In *AIPS*, Chicago, IL, USA, 1994.

[3] A. Cesta and A. Oddi. Gaining Efficiency and Flexibility in the Simple Temporal Problem. In *TIME*, Key West, FL, USA, 1996.

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

[5] M. Di Rocco, F. Pecora, and A. Saffiotti. When Robots Are Late : Configuration Planning for Multiple Robots with Dynamic Goals. In *IROS*, Tokyo, Japan, 2013.

[6] P. H. Morris, N. Muscettola, and T. Vidal. Dynamic Control Of Plans With Temporal Uncertainty. In *IJCAI*, Seattle, WA, USA, 2001.

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

[8] L. Planken, M. de Weerdt, and N. Yorke-Smith. Incrementally Solving STNs by Enforcing Partial Path Consistency. In *ICAPS*, Toronto, Canada, 2010.

[9] L. R. Planken, M. M. de Weerdt, and R. P. van der Krogt. P3C : A New Algorithm for the Simple Temporal Problem. In *ICAPS*, Sydney, Australia, 2008.

[10] C. Pralet and C. Lesire. Deployment of mobile wireless sensor networks for crisis management : A constraint-based local search approach. In *CP*, Lyon, France, 2014.

[11] L. Xu and B. Y. Choueiry. A New Efficient Algorithm for Solving the Simple Temporal Problem. In *TIME*, Cairns, Queensland, Australia, 2003.