A Local Method for Prioritized Fusion of Temporal Information

Mahat Khelfallah and Belaïd Benhamou
Laboratoire des Sciences de l'Information et des Systèmes LSIS - UMR CNRS 6168
Provence University
CMI, Technopôle de Château Gombert
39, rue Joliot-Curie - 13453 Marseille - France.
{mahat,benhamou}@cmi.univ-mrs.fr

Abstract

Information often comes from different sources and merging these sources usually leads to the apparition of inconsistencies. Fusion is the operation which consists in restoring the consistency of the merged information by changing a minimum of the initial information. In this paper, we are interested in linear constraints prioritized fusion in the framework of simple temporal problems (STPs). Priority expresses a preference relation between linear constraints and can represent either confidence or quality degrees of the constraints, or the reliability of their sources. We propose a local fusion method which we experiment on random prioritized STP instances.

1. Introduction

Information often comes from different sources and merging these sources usually leads to apparition of inconsistencies. Fusion consists in restoring the consistency of the merged information by keeping a maximum of the initial information unchanged.

Information fusion is an important area in the artificial intelligence field. Several fusion methods have been proposed in the literature [12, 1, 8]. Most of them was done in the framework of propositional logic or other logic-based formalisms.

There are many fields or applications where the information can be represented by simple linear constraints. For instance scheduling problems [7], some geographic information can be also expressed by linear constraints [9, 13, 5, 6].

In this paper, we are interested in the prioritized fusion of linear constraints in the framework of simple temporal problems (STPs). Each STP is issued from a source of information. We extend the formalism of STP to handle prioritized constraints. Priority expresses a preference relation between constraints. It stems from two situations: *(i)* All the sources of information are equally reliable, thus all the STPs have the same priority. But the constraints of each STP are ordered according to their quality or importance levels. The consideration of all the STPs generates a total preordering on all the constraints. *(ii)* Conversely, there is a total preordering between the sources of information according to their reliability or quality degrees. This defines a total preordering on the STPs. However, the constraints of each STP have the same priority. In both situations, merging all the STPs generates a prioritized STP which is probably inconsistent. The restoration of its consistency has two main steps: the detection of its conflicts, and the elimination of these conflicts.

In [5, 6] revision methods of linear constraints have been proposed in the framework of a real-world geographic application (a flooding application). We extend in this paper what was proposed in [5, 6] by: (1) considering more general linear constraints since the flooding problem was represented by a particular STP. (2) All the STP constraints are subject to correction in this work whereas only constraints involving the origin variable were corrected when revising the flooding problem. (3) handling prioritized constraints.

In [2], a local handling of conflicts in inconsistent belief bases was investigated. However, the local aspect concerned only the resolution of conflicts since an exhaustive identification of the conflicts was performed, before ordering them according to influence relations defined on these conflicts and correcting them. We shall see that the local handling in this paper concerns the identification of conflicts as well as their elimination.

The rest of this paper is organized as follows. In section 2, we recall some background on simple temporal problems STPs. We present, in section 3, the fusion principle and we propose a fusion method in section 4. This method is experimented on random instances of prioritized STPs, and the obtained results are given in section 5, before concluding in section 6.

2. Background

A *Simple Temporal Problem* (STP) S is defined by $S = (\mathcal{X}, C)$ where $\mathcal{X}$ is a finite set of variables $X_0, ..., X_n$, having continuous domains. These variables represent temporal events (time points) and X_0 usually represents the origin of time. C is the set of constraints of the form $X_j - X_i \leq a_{ij}$ defined on these variables, where a_{ij} is a scalar. Each constraint expresses a distance between two temporal events. Constraints of the form $X_j - X_i \geq a_{ij}$ can be also represented since $X_j - X_i \geq a_{ij}$ is equivalent to the constraint $X_i - X_j \leq -a_{ij}$.

A tuple $x = (x_1, ..., x_n)$ of real values is a *solution* of the STP S if the instantiation $\{X_1{=}x_1, ..., X_n{=}x_n\}$ satisfies all its constraints. The STP S is *consistent* if and only if it has a solution.

The STP $S = (\mathcal{X}, C)$ is associated with a directed edge-weighted graph, $G_d = (\mathcal{X}, E_d)$, called its *distance graph* where $\mathcal{X}$, the set of vertices, is the set of variables of the STP S, and E_d is the set of weighted arcs representing the set of constraints C. Each constraint $X_j - X_i \leq a_{ij}$ of C is represented by the arc $i \rightarrow j$[1], which is weighted by a_{ij}. For more details see [4].

3. Prioritized fusion of constraints

Before addressing the prioritized fusion of the constraints of STPs, we extend the definition of an STP by considering priorities. First, we define the notion of a prioritized set of constraints.

Definition 1 *A prioritized set of constraints is a set $\mathcal{C}$ of constraints which is partitioned into r strata $\mathcal{C}^1, \ldots, \mathcal{C}^r$ (i.e., $\mathcal{C} = \mathcal{C}^1 \cup \ldots \cup \mathcal{C}^r$, and $\forall i, j,\ i \neq j$: $\mathcal{C}^i \cap \mathcal{C}^j = \emptyset$), where all the constraints of each strata $\mathcal{C}^i$ have the same priority and have a higher priority than the constraints of $\mathcal{C}^j$ for each j such that $i < j \leq r$.*
A prioritized STP S is defined by $S = (\mathcal{X}, \mathcal{C})$ where $\mathcal{X}$ is a set of variables and $\mathcal{C}$ a set of prioritized constraints.

When different (prioritized) STPs are merged conflicts could appear even if each of the considered (prioritized) STPs is consistent separately. Let $S_1 = (\mathcal{X}_1, \mathcal{C}_1)$,..., $S_p = (\mathcal{X}_p, \mathcal{C}_p)$ be p (prioritized) STPs obtained from different sources. We can distinguish two situations:

- In the first one, we merge p prioritized STPs which have the same priority (reliability degree) such that for each prioritized STP $S_i = (\mathcal{X}_i, \mathcal{C}_i)$, the set of constraints $\mathcal{C}_i$ is stratified into r strata $\mathcal{C}_i^1, \ldots, \mathcal{C}_i^r$. Let S be the prioritized STP obtained from the aggregation of the p prioritized STPs $S_1, \ldots, S_p$ and defined by $S = (\mathcal{X}, \mathcal{C})$ where $\mathcal{X} = \bigcup_{1 \leq i \leq p} \mathcal{X}_i$ and $\mathcal{C}$ is obtained by aggregating the prioritized set of constraints $\mathcal{C}_1, \ldots, \mathcal{C}_p$. $\mathcal{C}$ is partitioned into r strata $\mathcal{C}^1, \ldots, \mathcal{C}^r$ where each set of constraints $\mathcal{C}^i = \bigcup_{1 \leq j \leq p} \mathcal{C}_j^i$.

- In the second situation, there is a total preordering between the p STPs but the constraints of each STP S_i have the same priority. Let S be the prioritized STP obtained from the union of the p STPs S_i with respect to the preordering defined on them. S is defined by $S = (\mathcal{X}, \mathcal{C})$ where $\mathcal{X} = \bigcup_{1 \leq i \leq p} \mathcal{X}_i$ and $\mathcal{C}$ is stratified into r $(r \leq p)$ strata $\mathcal{C}^1, \ldots, \mathcal{C}^r$ where each $\mathcal{C}^i = \bigcup_{S_j \text{ has the priority } i} \mathcal{C}_j$.

In both situations, the aggregation of the p (prioritized) STPs generates the prioritized STP S. With no lost of generality, we suppose in the sequel that S contains at most one constraint between each ordered pair of variables[2]. If the prioritized STP S is consistent, the fusion is done. Otherwise, its consistency has to be restored.

Example 1 *(Inspired from [4]'s example) We have two persons Nana and Sissi which go to work every morning, and we have two scenarii. The first one informs us that: Nana leaves home before 7:05, and arrives at work between 7:20 and 7:30. Sissi leaves home at most 5 mn after Nana does so and arrives at work at least 10 mn before Nana. The second scenario states that: It takes to Nana at most 10 mn to get at work, whereas it takes to Sissi between 10 and 20 mn to get at work, and Nana arrives at work at most 5 mn after Sissi leaves home. Information of each scenario can be represented by an STP. Let S_1 be the STP representing the first scenario, and S_2 be the STP representing the second one. Let X_1, X_2, X_3, X_4 be the variables representing the temporal events "Nana leaves home", "Nana arrives at work", "Sissi leaves home" and "Sissi arrives at work" respectively. Let X_0 be the variable representing the time 7:00 a.m. The information "Nana leaves home before 7:05" can be represented by the constraint $X_1 - X_0 \leq 5$. "Nana arrived at work between 7:20 and 7:30" is represented by $20 \leq X_2 - X_0 \leq 30$, and so on. We obtain the STPs S_1 and S_2 such that: $S_1 = (\mathcal{X}_1, C_1)$ where $\mathcal{X}_1 = \{X_0, X_1, X_2, X_3, X_4\}$ and $C_1 = \{X_1 - X_0 \leq 5\,, X_2 - X_0 \leq 30\,, X_0 - X_2 \leq -20\,, X_3 - X_1 \leq 5\,, X_4 - X_2 \leq -10\}$, and $S_2 = (\mathcal{X}_2, C_2)$ where $\mathcal{X}_2 = \{X_1, X_2, X_3, X_4\}$ and $C_2 = \{X_2 - X_1 \leq 10\,, X_4 - X_3 \leq 20\,, X_3 - X_4 \leq -10\,, X_2 - X_3 \leq 5\}$. The*

[1]For simplicity a vertex X_i of the graph G_d is denoted by its index i.

[2]If there are two constraints $X_i - X_j \leq a_{ij}$ and $X_i - X_j \leq b_{ij}$ in the prioritized STP S such that $a_{ij} < b_{ij}$, then only the constraint $X_i - X_j \leq a_{ij}$ is considered in S.

STPs S_1 and S_2 are consistent separately. We suppose that the first scenario is more reliable than the second one, thus the constraints of S_1 have higher priority than the ones of S_2. The aggregation of S_1 and S_2 gives the prioritized STP $S = (\mathcal{X}, \mathcal{C})$ where $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 = \{X_0, X_1, X_2, X_3, X_4\}$ and $C = C_1 \cup C_2 = \{X_1 - X_0 \leq 5, X_2 - X_0 \leq 30, X_0 - X_2 \leq -20, X_3 - X_1 \leq 5, X_4 - X_2 \leq -10, X_2 - X_1 \leq 10, X_4 - X_3 \leq 20, X_3 - X_4 \leq -10, X_2 - X_3 \leq 5\}$ such that the constraints of $\mathcal{C}_1$ have the priority 1which is the highest and the constraints of $\mathcal{C}_2$ have the priority 2.

The priority notion is not involved in the distance graph. The distance graph definition of a prioritized STP remains the same as the one of classical STPs.

Throughout this paper, we will manipulate the prioritized STP S instead of the (prioritized) STPs $S_1, \ldots, S_p$. Let n , m and r be respectively the number of variables, the number of constraints and the number of priorities of the prioritized STP S. The prioritized STP S is defined by $S = (\mathcal{X}, \mathcal{C})$ such that $\mathcal{C} = \mathcal{C}^1 \cup \ldots \cup \mathcal{C}^r$, where $\mathcal{C}^i$, $1 \leq i \leq r$, is the set of constraints of priority i. We suppose that 1 is the highest priority and r is the lowest. Let G_d be the distance graph associated with the prioritized STP S. Thus, n and m are also the number of vertices and the number of arcs of the distance graph G_d respectively. Restoring the consistency of the prioritized STP S consists in detecting its conflicts, representing them and identifying a subset of constraints whose correction is sufficient to eliminate them. We present in the following each of these steps.

3.1. Detection of Conflicts

The first step of the fusion operation is the detection of conflicts of the prioritized STP S. The method which we propose is based on a variant of the following well known result.

Theorem 1 *([14, 11, 10]) An STP is consistent if and only if its corresponding distance graph does not contain negative circuits.*

We can deduce from Theorem 1 that for restoring the consistency of an STP, we need to remove all the negative circuits of its distance graph. Actually, it is sufficient to remove all the elementary negative circuits in the distance graph to restore the consistency of an STP. This weakened the conditions of Theorem 1. We extend this result to handle prioritized STPs. We obtain the following theorem on which is based our fusion method.

Theorem 2 *A prioritized STP is consistent if and only if its corresponding distance graph does not contain elementary negative circuits.*

Proof 1 *Let S be a prioritized STP, and let S' be an STP obtained from the prioritized S by ignoring the priorities of S. S and S' have the same distance graph, let G_d be this distance graph. The STP S' is consistent if and only if it has a solution x which satisfies all its constraints. It is clear that a solution x of the STP S' is also a solution of the prioritized STP S, since S and S' have the same set of constraints. This implies that S is consistent if and only if S' is consistent. On the other hand, the STP S' is consistent if and only if G_d does not contain negative circuits (Theorem 1). Thus, we can conclude that the prioritized STP S is consistent if and only if its distance graph G_d does not contain negative circuits. If S is consistent then G_d does not contain any negative circuit, and in particular, it does not contain any elementary negative circuit. Conversely, if G_d does not contain any elementary negative circuit, then G_d does not contain any negative circuit. hence, the STP S' is consistent (Theorem 1), and the prioritized STP S is consistent.*

The presence of elementary negative circuits in G_d means that the prioritized STP S contains conflicts. A conflict is defined as follows.

Definition 2 *Let S be a prioritized STP and G_d be its distance graph. A conflict of S is a pair (σ, d) where σ is an elementary negative circuit of the distance graph G_d and d is the distance[3] of the circuit σ.*

Now, we define the *Conflict-Detection* function which detects conflicts of the distance graph. The *Conflict-Detection* function is an extension of the *Bellman-Ford* algorithm which computes the shortest paths of a graph [3]. The main idea is to compute for each pair (i, j) of vertices the shortest path from i to j in the distance graph G_d. In particular, when $i = j$, the function computes the shortest circuit visiting the vertex i. In fact, the *Conflict-Detection* function computes the conflicts, of the distance graph, having the minimum circuit length.

The *Conflict-Detection* function is given in Algorithm 1. It consists in two steps. First, it constructs a matrix $mat^{(0)}$ whose elements are pairs defined by: $mat^{(0)}_{ij} = (p_{ij}, d_{ij})$ where p_{ij} represents a path of length 1 from i to j in G_d, and d_{ij} is the distance of the path p_{ij}. The initialization step terminates by copying the matrix $mat^{(0)}$ in the matrix mat. The second step is a loop which computes the shortest paths between each pair (i, j) of vertices. At each iteration l of the "while" loop, a call to the *Shortest-Path-Extension* function given in Algorithm 2 is made to compute the shortest paths of length l between each pair of vertices. The loop stops either when a conflict is detected or when the length of the computed paths reaches n.

[3]The distance of a path is the sum of its arc weights.

Algorithm 1 Conflict Detection

```
Function Conflict-Detection($G_d$): a set of conflicts
Var $mat^{(0)}$, $mat$: matrices of pairs (path,distance)
    $Conf$: a set of conflicts
Begin
{ Initialization }
  $Conf := \emptyset$
  $l := 2$
  for $i, j := 1$ to $n$ do
    if there is an arc $i \rightarrow j$ in $G_d$ weighted by $a_{ij}$
      then $mat^{(0)}_{ij} := ((i,j), a_{ij})$
      else if $i \neq j$ then $mat^{(0)}_{ij} := (\emptyset, \infty)$
        else $mat^{(0)}_{ij} := (\emptyset, 0)$
  $mat := mat^{(0)}$
{ Path extension and conflict detection }
  while $l \leq n$ and $Conf = \emptyset$ do
  begin
    $mat$ := Shortest-Path-Extension($mat^{(0)}, mat, Conf$)
    $l := l + 1$
  end
  Conflict-Detection := $Conf$
End
```

Algorithm 2 Shortest Path Extension

```
Function Shortest-Path-Extension($mat^{(0)}$, $mat$, Var
$Conf$): the extended matrix of (path,distance)
Var $mat'$ the extended matrix of (path,distance)
Begin
{ Initialization }
  for $i, j := 1$ to $n$, $i \neq j$ do $mat'_{ij} := (\emptyset, \infty)$
  for $i := 1$ to $n$ do $mat'_{ii} := (\emptyset, 0)$
{ Extension of the paths of mat }
  for $i, j := 1$ to $n$ do
    for $k := 1$ to $n$ do
      if ($mat_{ik}.path \neq \emptyset$ and $mat^{(0)}_{kj}.path \neq \emptyset$ and
  $mat_{ik}.distance + mat^{(0)}_{kj}.distance < mat'_{ij}.distance$)
      then begin
        $p := mat_{ik}.path \bullet mat^{(0)}_{kj}.path$
        $d := mat_{ik}.distance + mat^{(0)}_{kj}.distance$
        $mat'_{ij} := (p, d)$
        if $(i = j)$ then $Conf := Conf \cup mat'_{ii}$
      end
  Shortest-Path-Extension := $mat'$
End
```

The *Shortest-Path-Extension* function is based on the following: a shortest path p_{ij} of length l from i to j is composed by a shortest path p_{ik} of length $l-1$ from i to k and an arc from k to j. When the *Shortest-Path-Extension* function is called at the iteration l in the loop of the *Conflict-Detection* function, it takes as arguments: $mat^{(0)}$ the initial matrix of pairs (path,distance) corresponding to the distance graph G_d, mat the matrix of pairs (path,distance) corresponding to the shortest paths of length $l-1$ in G_d. It returns the matrix mat' corresponding to the shortest paths of G_d of length l. In particular, mat'_{ii} will contain a negative circuit of length l including the vertex i, if it exists. Furthermore, the detected negative circuits are elementary and are added to $Conf$. The *Shortest-Path-Extension* function returns in $Conf$ the set of conflicts whose negative circuits are of length l. If $Conf = \emptyset$, then there is no negative circuit of length l in G_d.

Now, We evaluate the complexity of the *Shortest-Path-Extension* function. The initialization phase is performed in $O(n^2)$. The second phase consists in three loops. Each iteration of the internal loop can be done in $O(n)$ since both the path and the distance tests are done in a constant time and path concatenation is done in $O(n)$. Thus, the second phase can be performed in $O(n^4)$. Therefore, the time complexity of the *Shortest-Path-Extension* function is $O(n^4)$ in the worst case.

Now, we evaluate the complexity of the *Conflict-Detection* function. The initialization phase can be performed in $O(n^2)$. In the second phase, the function performs at most $n-1$ iterations, and the complexity of each of them is identical to the complexity of the Shortest-Path-Extension function which is $O(n^4)$. Therefore, the complexity of the *Conflict-Detection* function is $O(n^5)$ in the worst case.

Now, we can prove some properties of the *Shortest-Path-Extension* and *Conflict-Detection* functions.

Proposition 1 *The Shortest-Path-Extension function applied to the matrix mat corresponding to the shortest paths of length l in G_d detects at least one elementary negative circuit if and only if it exists one of length $l+1$ in G_d.*

Proof 2 *Let $mat^{(0)}$ be the matrix of pairs (path,distance) corresponding to the distance graph G_d, and mat be the matrix of pairs (path,distance) representing the shortest paths of G_d of length l. Let mat' =* Shortest-Path-Extension*($mat^{(0)}$,mat,$Conf$) where $Conf$ is the set of detected conflicts. The shortest path p_{ij} from i to j of length $l+1$ of G_d is defined by $p_{ij} = Min_{1 \leq k \leq n}(p_{ik} \bullet (k \rightarrow j))$, where Min is a function which selects the path having the minimal distance, p_{ik} is the shortest path of length l from i to k of G_d, and $(k \rightarrow j)$ is an arc of G_d. This implies that the elements of mat represent really the shortest paths of length $l+1$ of G_d. We can prove now the first statement of the proposition.*

($\Rightarrow$) If the Shortest-Path-Extension function applied to the matrix mat corresponding to the shortest paths of length l in G_d detects at least one elementary negative circuit, then there exists at least one vertex i included in a negative circuit of length $l+1$.
($\Leftarrow$) Suppose that there is an elementary negative circuit σ_i of length $l+1$ including a vertex i of G_d. Let d ($d<0$) be the distance of σ_i. Since $mat'_{ii} = (p_{ii}, d_{ii})$ represents the shortest path p_{ii} of length $l+1$ from i to i and its distance d_{ii}, then the circuit p_{ii} is shorter than the circuit σ_i, and consequently $d_{ii} < d < 0$. This means that the circuit p_{ii} is negative also. Thus, the Shortest-Path-Extension function detects at least one negative circuit, if it exists.

Furthermore, the negative circuits detected by the Shortest-Path-Extension function are elementary. This can be trivially proved since the function is applied the extend the shortest paths of length l only if there are no negative circuits of length l' such that $l' \leq l$. Proposition 2 states that when the *Conflict-Detection* function does not detect any negative elementary circuits in the distance graph G_d the prioritized STP S is consistent.

Proposition 2 *The prioritized STP S is consistent if and only if the Conflict-Detection function, applied to its distance graph G_d, does not detect any conflict.*

Proof 3 *($\Rightarrow$) If the prioritized STP S is consistent then its distance graph G_d does not contain any elementary negative circuit (by Theorem 2). In particular, G_d does not contain any elementary negative circuit of length l, $2 \leq l \leq n$. Thus, the repeated application of the Shortest-Path-Extension function does not detect any negative circuit, and the Conflict-Detection function does not detect any conflict.*
($\Leftarrow$) If the Conflict-Detection function does not detect any conflict, then the Shortest-Path-Extension function has been applied until $l = n$ and for each length, no conflict was detected (i.e., $Conf = \emptyset$). This means that there is no negative circuit of length l, $2 \leq l \leq n$ in G_d (by Proposition 1). Therefore, there is no elementary negative circuit in G_d, hence the prioritized STP S is consistent.

3.2. Representation of conflicts

Each conflict of the prioritized STP S is identified by a tuple (σ, d) where σ is an elementary negative circuit of the distance graph G_d and d is the distance of σ. We recall that each arc $i \rightarrow j$ in G_d, weighted by a_{ij}, represents the constraint $c_{ij} : X_j - X_i \leq a_{ij}$ of the prioritized STP S. We define now the notion of conflicting constraint.

Definition 3 *Let $S = (\mathcal{X}, C)$ be a prioritized STP, G_d its distance graph, and $Conf$ the set of conflicts of S. A constraint $c_{ij} \in C$ is a conflicting constraint if and only if there is a conflict $c = (\sigma, d)$ in $Conf$ such that the arc $i \rightarrow j$ belongs to the elementary negative circuit σ of G_d.*

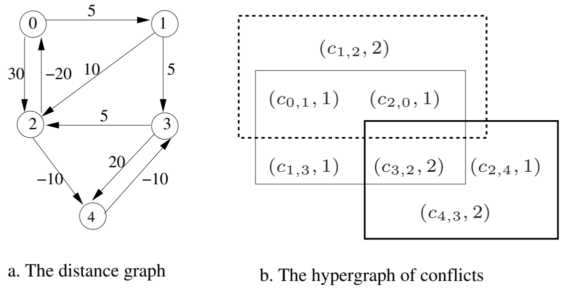

Figure 1. The distance graph and the hypergraph of conflicts of the prioritized STP S defined in Example 1

Let $ConfConst$ be the function which associates to each conflict $c = (\sigma, d)$ the set of its conflicting constraints. That is, $ConfConst(c) = \{c_{ij} \in C : i \rightarrow j$ is an arc of $\sigma\}$.

The set of the detected conflicts $Conf$ is represented by a vertex-weighted hypergraph $H_c = (V, E_c)$ where V is the set of vertices corresponding to the set of all conflicting constraints defined by: $V = \bigcup_{c \in Conf} ConfConst(c)$, and E_c is the set of hyperedges defined as follows: each hyperedge e represents a conflict c of $Conf$ such that e represents the conflicting constraints of the conflict c, i.e., $e = ConfConst(c)$. Thus $E_c = \{ConfConst(c) : c \in Conf\}$. Each vertex c_{ij} of H_c is weighted by k, where k is the priority of the constraint c_{ij}, i.e., $c_{ij} \in \mathcal{C}^k$. H_c is called the *hypergraph of conflicts* of the prioritized STP S. Let $V^1, \ldots, V^r$ be a partition of the set of vertices V such that: $v \in V^i$ if the weight of the vertex v in the hypergraph H_c equals i.

Example 2 *The distance graph of the prioritized STP S defined in Example 1 is represented in Figure 1.a. The elementary negative circuit {(0,1),(1,2),(2,0)} shows a conflict between the constraints $c_{0,1}$, $c_{1,2}$ and $c_{2,0}$. This adds the hyperedge $\{(c_{0,1}, 1), (c_{1,2}, 2), (c_{2,0}, 1)\}$ in the hypergraph of conflicts. By considering all the elementary negative circuits of the distance graph of Figure 1.a, we obtain the hypergraph of conflicts of Figure 1.b. Each vertex of the hypergraph is a constraint which is weighted by its priority.*

3.3. Identification of a subset of constraints to correct

Removing all the detected conflicts needs correcting some constraints involved in them. At least one conflicting constraint of each detected conflict has to be corrected. In other words, the intersection of the set of the corrected constraints and the set of conflicting constraints of each conflict is not empty. Therefore, the subset of corrected constraints is a *transversal* of the hypergraph of conflicts H_c of the prioritized STP S. Minimizing change when restoring the consistency of a prioritized STP S amounts to minimizing the number of corrected constraints of $\mathcal{C}^1$ (having the highest priority), then the number of corrected constraints of $\mathcal{C}^2$, and so on. We recall the definition of a transversal before defining the notion of *prioritized transversal*.

Definition 4 *Let $H = (V, E)$ be a hypergraph. T is a* transversal *of the hypergraph H if $T \subseteq V$ and for each hyperedge e of E, $T \cap e \neq \emptyset$.*

Now, we define a preference relation on the transversals of a vertex-weighted hypergraph.

Definition 5 *Let $H = (V, E)$ be a vertex-weighted hypergraph where $V = V^1 \cup \ldots \cup V^r$ such that the vertex $v \in V^i$ if v is weighted by i in H. Let T_1 and T_2 be two transversals of the hypergraph H. T_1 is preferred to T_2 if (i) $\exists i$, $1 \leq i \leq r$ such that $|T_1 \cap V^i| < |T_2 \cap V^i|$, and (ii) $\forall j$, $1 \leq j < i$, $|T_1 \cap V^j| = |T_2 \cap V^j|$.*

Definition 6 *Let $H = (V, E)$ be a vertex-weighted hypergraph. T_p is a prioritized transversal of the hypergraph H if (i) T_p is a transversal of H, and (ii) for each transversal T of H, T is not preferred to T_p.*

Example 3 *The hypergraph of conflicts shown in Figure 1.b has many transversals. For instance T=$\{(c_{2,0}, 1), (c_{4,3}, 1)\}$. It has one prioritized transversal T_p=$\{(c_{1,2}, 2), (c_{3,2}, 2)\}$.*

Algorithm 3 Prioritized Transversal procedure

```
Procedure Prioritized-Transversal($H_c = (V, E_c)$,$T$, $W$,
          Var $T_p$)
Begin
 if $E_c = \emptyset$ then $T_p := T$
 else if $V \neq W$ then
 begin
   select a vertex $v$ from $V - W$
   Prioritized-Transversal ($H_c, T, W \cup \{v\}, T_p$)
   $V := V - \{v\}$
   $E_c := E_c - \{e : e \in E_c$ and $e \ni v\}$
   if ($T_p = \emptyset$ or $T \cup \{v\}$ is preferred to $T_p$)
      then Prioritized-Transversal ($H_c, T \cup \{v\}, W, T_p$)
 end
End
```

We define the *Prioritized-Transversal* procedure (Algorithm 3) to compute a prioritized transversal of the hypergraph of conflicts H_c. The search space of this procedure is a binary tree. At each node of this tree, a vertex is either considered in the current partial transversal T, or not considered then it is put in the set W. This procedure does not enumerate systematically all the possible transversals of the hypergraph of conflicts to find a prioritized one. Indeed, it prunes the branches of the search tree leading to partial transversals T which are not preferred to the current prioritized transversal T_p. The *Prioritized-Transversal* procedure is called with the parameters $H_c = (V, E_C)$ which is the hypergraph of conflicts, T the current partial transversal, W the set of non considered vertices and T_p the current prioritized transversal of the hypergraph H_c which the procedure returns. Initially, $T = W = T_p = \emptyset$.

To evaluate roughly the complexity of the *Prioritized-Transversal* procedure, let n_c and m_c be respectively the number of vertices and the number of hyperedges of the hypergraph of conflicts H_c. The search space is bounded by 2^{n_c} which is the number of all the possible transversals. The selection of a vertex $v \in V - W$ can be performed in $O(n_c)$, and the removal of the hyperedges incident to v is performed in $O(m_c)$. Testing if the current partial transversal $T \cup \{v\}$ is preferred to the current prioritized transversal T_p can be done in $O(r)$. Since $r < n_c$, then the complexity of the *Prioritized-Transversal* algorithm is $O((m_c + n_c)2^{n_c})$ in the worst case.

Example 4 *The application of the Prioritized-Transversal procedure to the hypergraph of conflicts shown in Figure 1.b returns the prioritized transversal $T_p = \{(c_{1,2}, 2), (c_{3,2}, 2)\}$.*

3.4. Correction of the conflicting constraints

Now, we shall see how to perform the corrections. Let $c = (\sigma, d)$ be a conflict of the prioritized STP S. The elimination of the conflict c needs the elimination of its associated elementary negative circuit σ. This implies the correction of at least one of the constraints involved in σ, i.e., at least one of the constraints of $ConfConst(c)$ is corrected. The following proposition shows how this correction is done.

Proposition 3 *Let S be a prioritized STP and $c = (\sigma, d)$ be a conflict of S. Let $c_{ij} : X_j - X_i \leq a_{ij}$ be a conflicting constraint of c ($c_{ij} \in ConfConst(c)$). Replacing the constraint $c_{ij} : X_j - X_i \leq a_{ij}$ by the constraint $X_j - X_i \leq a_{ij} - d$ eliminates the conflict c.*

Proof 4 *Let $c = (\sigma, d)$ be a conflict, and let $c_{ij} : X_j - X_i \leq a_{ij}$ be a conflicting constraint of c, $c_{ij} \in ConfConst(c)$.*

The circuit σ has a negative distance d, and replacing the constraint $X_j - X_i \leq a_{ij}$ by the constraint $X_j - X_i \leq a_{ij} - d$, will make this distance equal to zero. Hence the negative circuit σ is eliminated and the conflict c is corrected.

Example 5 *In Figure 1, the elementary negative circuit {(0,1),(1,2),(2,0)} whose distance is -5 identifies the conflict ({(0,1),(1,2),(2,0)},-5) between the constraints $X_1 - X_0 \leq 5$ (representing the fact that Nana leaves home before 7:05), $X_2 - X_1 \leq 10$ (representing the fact that it takes less than 10 mn to Nana to get at work) and $X_0 - X_2 \leq -20$ (representing the fact that Nana arrives at work after 7:20). This conflict can be removed by replacing the constraint $X_2 - X_1 \leq 10$ by the constraint $X_2 - X_1 \leq 15$ (it takes to Nana less than 15 mn to get work instead of 10 mn).*

The following proposition states that a conflict is eliminated if and only if one of its conflicting constraints is corrected.

Proposition 4 *Let S be an inconsistent prioritized STP and $c = (\sigma, d)$ be a conflict of S. The conflict c is eliminated if and only if at least one of its conflicting constraints $c_{ij} \in ConfConst(c)$ is corrected with respect to Proposition 3.*

Proof 5 *Let S be an inconsistent prioritized STP and $c = (\sigma, d)$ be a conflict of S. ($\Rightarrow$) Suppose that the conflict $c = (\sigma, d)$ is eliminated. This implies that the negative circuit σ is eliminated, i.e., its distance is changed from negative to positive. This is done by changing at least the weight of one of its arcs. In other words, by correcting, at least, one constraint c_{ij} of $ConfConst(c)$. ($\Leftarrow$) The correction of a constraint $c_{ij} \in ConfConst(c)$ makes the conflict circuit σ positive. Hence, the conflict $c = (\sigma, d)$ is eliminated.*

When correcting a constraint no new conflicts are generated and the following theorem states that the correction of the constraints corresponding to a transversal of the hypergraph of conflicts representing the detected conflicts, eliminates these conflicts.

Theorem 3 *Let S be an inconsistent prioritized STP and let $Conf$ be a set of detected conflicts of S. Let H_c be the hypergraph of conflicts representing the set $Conf$. The conflicts of $Conf$ are removed from the prioritized STP S if and only if the constraints corresponding to a transversal of the hypergraph of conflicts H_c are corrected.*

Proof 6 *Let S be an inconsistent STP and let $Conf$ be the set of detected conflicts of S. Let H_c be the hypergraph of conflicts representing $Conf$.*
($\Rightarrow$) Suppose that all the conflicts of $Conf$ are removed. Let C_r be the subset of constraints corrected when removing the conflicts of $Conf$. Each conflict c of $Conf$ is removed by the correction of, at least, one of its conflicting constraints, that is a constraint of $ConfConst(c)$ (Proposition 4). This means that $C_r \cap ConfConst(c) \neq \emptyset$ for each conflict $c \in Conf$. However, $ConfConst(c) = e$ where e is the hyperedge representing the conflict c in H_c. Then, $C_r \cap e \neq \emptyset$ for each hyperedge e of the hypergraph of conflicts H_c. Thus C_r is a transversal of the hypergraph of conflicts H_c.
($\Leftarrow$) Suppose that C_r is a transversal of the hypergraph of conflicts H_c and all its corresponding constraints are corrected. This means that for each hyperedge e of H_c, $e \cap C_r \neq \emptyset$. However, each hyperedge e of H_c represents the set of conflicting constraints of a conflict c of $Conf$, i.e., $e = ConfConst(c)$. This implies that for each conflict c of $Conf$, at least one conflicting constraint of c is corrected (because $C_r \cap ConfConst(c) \neq \emptyset$). Hence, each conflict c is eliminated (by Proposition 4). We conclude that the correction of all the constraints corresponding to C_r eliminates all the conflicts of $Conf$.

4. Local prioritized fusion algorithm

Since the number of elementary negative circuits of the distance graph of a prioritized STP is potentially high, the exhaustive detection of conflicts can be impossible. A local handling of the problem seems to be a good alternative to this problem. That is, if the conflicts are detected and corrected bundle by bundle, the complexity of the fusion operation decreases. On other hand, if a detected conflict c, of the prioritized STP S, involves for instance a constraint c_{ij}, and if this constraint participates in another not yet detected conflict c', then the correction of the constraint c_{ij} could eliminate the conflict c'.

The *Local Prioritized Fusion* function, sketched in Algorithm 4, consists in detecting a bundle of conflicts, then in eliminating them by correcting the conflicting constraints corresponding to a prioritized transversal of the hypergraph of conflicts. It repeats these operations until the restoration of the consistency.

Theorem 4 *The Local Prioritized Fusion algorithm, applied to the prioritized STP S, terminates and restores the consistency of S.*

Proof 7 *Let S be a prioritized STP. The number of conflicts of S is finite and the correction of constraints cannot generate new ones. At each iteration of the Local-Prioritized-Fusion algorithm, the Conflict-Detection function detects a subset of conflicts. The elimination of these conflicts is guaranteed by the correction of the constraints of a prioritized transversal T_p of the hypergraph of conflicts H_c representing the detected conflicts (see Theorem 3). This implies that at each iteration, the total number of conflicts*

Algorithm 4 Local Prioritized Fusion

```
Function Local-Prioritized-Fusion($S$ prioritized STP): a
    consistent prioritized STP
Begin
  Construct $G_d$ the distance graph of $S$
  $Conf$ := Detection-Conflict($G_d$)
  repeat
    Construct $H_c$ the hypergraph of conflicts corresponding
        to $Conf$
    $T_p := \emptyset$
    Prioritized-Transversal($H_c, \emptyset, \emptyset, T_p$)
    Correct the constraints corresponding to the prioritized
        transversal $T_p$
    $Conf$ := Detection-Conflict($G_d$)
  until $Conf = \emptyset$
  Local-Prioritized-Fusion := $S$
End
```

decreases, until the elimination of all conflicts. The algorithm stops when no conflicts are detected by the Conflict-Detection function. This implies that there is no conflict in the prioritized STP S (Proposition 2). Therefore, the Local-Prioritized Fusion function terminates and restores the consistency of the prioritized STP S.

To evaluate the complexity of the *Local-Prioritized-Fusion* algorithm, we proceed step by step. Let m_c be the number of conflicts of the prioritized STP S. This number is bounded by the number of possible elementary circuits of the distance graph G_d which is itself bounded by $\sum_{k=2}^{n} A_n^k$ where $A_n^k = \frac{n!}{(n-k)!}$. The complexity of the *Conflict-Detection* function is $O(n^5)$, and the construction of the distance graph is in $O(nm)$. Now, we evaluate the complexity of each iteration. Since the number of detected conflicts at each iteration is at most n and each conflict can involve at most n conflicting constraints, the construction of the hypergraph of conflicts H_c corresponding to $Conf$ is in $O(n^2)$. The prioritized transversal T_p of the hypergraph of conflicts H_c is computed in $O((m'_c + n'_c)2^{n'_c})$ where m'_c is the number of handled conflicts and n'_c is the number of conflicting constraints. The number m'_c is bounded by n because at most n conflicts are handled in each iteration and n'_c is bounded by n^2. Thus, the Prioritized transversal search is performed in $O(n^2 2^{n^2})$ in the worst case. The correction of the constraints of T_p is performed in $O(n)$ since at most n constraints can be corrected. Therefore, each iteration is performed in $O(n^2 2^{n^2})$ in the worst case. The *Local Prioritized Fusion* algorithm performs at most $\frac{m_c}{n}$ iterations. Therefore, its complexity is $O(m_c n 2^{n^2})$ in the worst case.

5. Experimental results

The local prioritized fusion algorithm presented in this paper is implemented in C and tested on randomly generated prioritized STPs. The program runs on a P4 with 2.2 MHz and 512 MB of RAM.

Generation of random prioritized STPs is based on three parameters: the number of variables n, the number of priorities (strata) r, and the constraint density d which is the ratio of the number of constraints to the number of possible constraints, $d = \frac{number\ of\ constraints}{n(n-1)}$. The tightness of the constraints is represented by the interval [-50,50] where the constraint weights are generated. A sample of 50 problems is generated for each tuple (n, r, d) and the measures are taken in average. The experimental results obtained by the application of the *Local Prioritized Fusion* algorithm on random prioritized STP instances are shown in Table 1. A dash (-) means that the algorithm did not answer after running 2 hours.

We can see in Table 1 that when the density grows, the number of conflicts grows. This increases the number of corrected constraints, the number of iterations and the CPU time. The fusion algorithm becomes faster as the number of considered priorities increases even though the number of detected conflicts, the number of corrected constraints and the number of iterations increase. This can be explained by the speed up of the prioritized transversal search when there are many strata. As expected, the number of corrected constraints when only one stratum is considered is the smallest. This is due to minimal change policy. In the same spirit, when the number of considered priorities increases, the number of corrected constraints having the highest priority decreases. We can see that the method fuses in reasonable time problems having 30 variables with a small density ($d = 0.2$), but reaches its limit on higher densities.

6. Conclusion

In this paper, we investigated the prioritized fusion of Simple Temporal Problems STPs. That is, giving a set of prioritized STPs to fuse, we considered the prioritized STP S resulting from their union. If S is consistent, then the fusion is done. Otherwise, the consistency of S has to be restored. We presented a principle of a prioritized fusion on which is based the *local prioritized* fusion method which we proposed. Two justifications motivate the local strategy. The first one is the high complexity of the exhaustive detection of the conflicts. The second justification is related to the conflict nature. If a bundle of conflicts is detected and corrected, this could eliminate not yet detected conflicts. This speeds up the fusion operation.

In future, we can improve the local prioritized fusion algorithm efficiency by considering a ”good” transversal

		$\mathbf{n} = 10$			$\mathbf{n} = 20$			$\mathbf{n} = 30$		
		Density			**Density**			**Density**		
# priorities		**0.2**	**0.5**	**0.8**	**0.2**	**0.5**	**0.8**	**0.2**	**0.5**	**0.8**
r = 1	**# conflicts**	1	23	43	42	160	232	142	-	-
	# corrected const	1	14	25	23	92	144	68	-	-
	# corrected const. highest prior.	1	14	25	23	92	144	68	-	-
	# corrected const. lowest prior.	1	14	25	23	92	144	68	-	-
	# iterations	2	8	11	11	21	26	19	-	-
	CPU time (s)	0	0	0	0	33	120	416	-	-
r = 5	**# conflicts**	1	25	47	44	197	292	187	-	-
	# corrected const	1	15	32	29	141	219	123	-	-
	# corrected const. highest prior.	0	0	0	0	2	9	0	-	-
	# corrected const. lowest prior.	1	6	11	13	42	57	47	-	-
	# iterations	2	8	12	12	27	35	24	-	-
	CPU time (s)	0	0	0	0	1	7	135	-	-
r = 10	**# conflicts**	1	22	48	47	204	325	192	-	-
	# corrected const	1	13	13	28	150	251	130	-	-
	# corrected const. highest prior.	0	0	0	0	0	1	0	-	-
	# corrected const. lowest prior.	1	4	5	7	23	34	22	-	-
	# iterations	2	8	13	12	28	38	24	-	-
	CPU time (s)	0	0	0	0	3	7	127	-	-

Table 1. Experimental results obtained by the application of the Local-Prioritized-Fusion algorithm on random prioritized STP instances

rather than a prioritized one. We also hope to extend this work to handle disjunctive temporal problems.

References

[1] S. Benferhat, D. Dubois, and H. Prade. Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study part 1: The flat case. *Studia Logica*, 58:17–45, 1997.

[2] S. Benferhat and L. Garcia. Handling locally stratified inconsistent knowledge bases. *Studia Logica*, 70:77–104, 2002.

[3] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[4] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[5] M. Khelfallah and B. Benhamou. Geographic information revision based on constraints. In *Proc. of the 14th European Conference on Artificial Intelligence, ECAI'04*, pages 828–832, 2004.

[6] M. Khelfallah and B. Benhamou. Two revision methods based on constraints: Application to a flooding problem. In *Proc. of the 7th Int. Conf. of Artificial Intelligence and Symbolic Computation AISC'04*, volume 3249 of *LNAI*, pages 265–270, 2004.

[7] R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *Omega*, 29:249–272, 2001.

[8] S. Konieczny, J. Lang, and P. Marquis. Distance based merging: A general framework and some complexity results. In *Proc. of the 8th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'02)*, pages 97–108, 2002.

[9] G. Kuper, G. L. Libkin, and J. Paradaens, editors. *Constraint Databases*. Springer-Verlag, 2000.

[10] C. Leiserson and J. Saxe. A mixed-integer linear programming problem which is efficiently solvable. In *Proc. of the 21st annual Allerton conference on Communications, Control, and Computing*, pages 204–213, 1983.

[11] Y. Lia and C. Wong. An algorithm to compact a vlsi symbolic layout with mixed constraints. In *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, volume 2, pages 62–69, 1983.

[12] J. Lin and A. Mendelzon. *Dynamic Worlds: From the Frame Problem to Knowledge Management*, volume 12 of *Applied Logic Series*, chapter Knowledge Base Merging by Majority. Kluwer, 1999.

[13] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases with Application to GIS*. Morgan Kaufmann, 2002.

[14] R. Shostak. Deciding linear inequalities by computing loop residues,. *Journal of ACM*, 28(4):769–779, 1981.