Contribution au Raisonnement Temporel : Étude des Techniques de Satisfaction de Contraintes Numériques et Symboliques

Malek Mouhoub

CRIN-CNRS & INRIA Lorraine, Bat LORIA,BP 239,
54506 Vandœuvre-lès-Nancy, France
E-mail : mouhoub@loria.fr

Résumé. Durant ces dernières années les tentatives de prise en compte du temps sous ses deux formes, symbolique et numérique, ont été nombreuses et variées. Dans ce cadre d'intérêt, nous avons défini un modèle qui généralise l'algèbre d'intervalles de Allen afin d'intégrer des informations métriques sous forme de fenêtres temporelles. Cette représentation se base sur un domaine discret du temps. Afin de raisonner sur ces informations temporelles, l'approche algorithmique de propagation de contraintes sur des domaines discrets a été choisie. Nous proposons dans cet article un algorithme général de résolution de problèmes de contraintes temporelles. La résolution de ces problèmes étant NP-difficile, des algorithmes de consistance d'arcs et de chemins sont utilisés pour réduire l'espace de recherche de solutions et augmenter par conséquent les performances de l'algorithme général. Pour faire face aux problèmes de complexité, nous avons étudié les contraintes temporelles afin d'extraire des propriétés que nous avons utilisé dans les algorithmes de consistance d'arcs et de chemins afin d'améliorer leur efficacité. Dans le but de déterminer jusqu'a quel degré les consistances d'arcs et de chemins sont utiles pour résoudre des problèmes de satisfaction de contraintes temporelles, nous avons effectué une étude comparative des différentes stratégies de recherche de solutions afin de déterminer la meilleure d'entre elles. Une évaluation pratique des performances de notre algorithme utilisant la stratégie optimale pour résoudre des problèmes de satisfaction de contraintes temporelles ainsi que des problèmes d'ordonnancement de tâches, présentée dans cet article, donne des résultats très intéressants.

1 Introduction

Le temps intervient de plus en plus dans la plupart des activités industrielles et administratives. Les tentatives de prise en compte du temps sous ses deux formes, symbolique et numérique, ont alors été nombreuses et variées. Elles concernent aussi bien la représentation et la gestion des informations temporelles que le raisonnement temporel. Dans ce cadre d'intérêt, nous avons défini un modèle qui généralise l'algèbre d'intervalles de Allen afin d'intégrer des informations métriques sous forme de fenêtres temporelles. Cette représentation se

base sur un domaine discret du temps.
Afin de raisonner sur ces informations temporelles, l'approche algorithmique de propagation de contraintes temporelles sur des domaines discrets a été choisie (les fenêtres temporelles définissent des ensembles finis d'intervalles convexes). Pour ce faire, nous ramenons tout problème de contraintes temporelles représenté par notre modèle en un problème d'étiquetage pouvant être résolu efficacement par des techniques de propagation de contraintes. Nous nous intéressons de ce fait aux problèmes de satisfaction de contraintes temporelles qualitatives et numériques. Pour résoudre ces problèmes, nous proposons un algorithme général de recherche de solutions avec retour arrière. La résolution de ces problèmes étant NP-difficile, des algorithmes de consistance d'arcs et de chemins sont utilisés pour réduire l'espace de recherche de solutions et augmenter par conséquent les performances de l'algorithme général. Pour faire face aux problèmes de complexité, nous avons étudié les contraintes temporelles afin d'extraire des propriétés que nous avons utilisé dans les algorithmes de consistance d'arcs et de chemins afin d'améliorer leur efficacité. L'algorithme que nous proposons est divisé en deux phases :

– une phase de pre-traitement dans laquelle nous appliquons des algorithmes de consistance locale permettant de réduire la taille de l'espace de recherche de solutions, et

– une phase de recherche de solutions dans laquelle nous appliquons une stratégie de recherche avec retour arrière. Des algorithmes de consistance locale sont également utilisés dans cette phase afin d'augmenter les performances de l'algorithme général que nous proposons.

Afin de répondre à la question suivante : jusqu'a quel degré la consistance d'arcs et de chemins sont utiles pour résoudre efficacement les problèmes de satisfaction de contraintes temporelles ? nous avons effectué une étude comparative des différentes stratégies de recherche de solutions afin de déterminer la meilleure d'entre elles à résoudre ces problèmes de satisfaction de contraintes. Ces stratégies diffèrent, en effet, dans leur degré d'utilisation des algorithmes de consistance locale.
La suite de cet article est divisée de la manière suivante. Dans la section suivante, nous présentons notre modèle de représentation des informations temporelles qualitatives et numériques. L'algorithme de résolution, des problèmes de satisfaction de contraintes temporelles représentés par notre modèle, utilisant des algorithmes de consistance locale est présenté en section trois. Les améliorations apportées à ces derniers algorithmes sont aussi présentés dans cette section. Une étude comparative des différentes stratégies de recherche de solutions est présentée en section quatre. Dans la section cinq, nous présentons les performances de notre algorithme général à résoudre des problèmes de contraintes temporelles ainsi que des problèmes d'ordonnancement.

2 Représentation de connaissances

2.1 Introduction

Depuis plusieurs années, les tentatives de prise en compte du temps sous ses deux formes formes numériques et symboliques ont été nombreuses et variées notamment en intelligence artificielle. Bien que les approches numériques soient efficaces dans certains domaines d'application, elles restent limitées lorsqu'il s'agit d'exprimer des informations qualitatives. D'un autre coté, les approches traitant l'aspect qualitatif du temps[Allen 83][Vilain 86] trouvent des difficultés à raisonner sur des informations métriques. D'autres travaux ont alors été effectués dans le but de capturer et raisonner sur les deux types d'informations. Meiri[Meiri 91] propose une structure généralisant l'algèbre d'instants, l'algèbre d'intervalles et les réseaux métriques. Les deux types de contraintes sont supportées par un seul réseau. Contrairement au modèle de Meiri, Kautz et Ladkin[Kautz 91] traitent les informations numériques et symboliques séparément. Les contraintes métriques sont représentées sous forme d'un système d'équations linéaires alors que les relations qualitatives entre points temporels sont décrites par un système de calcul basé sur les relations de base de Allen. Deux réseaux permettent de supporter les deux types de contraintes. La propagation se fera alors sur les deux réseaux. La gestion des contraintes numériques et symboliques est assurée par des procédures de translation entre les deux types d'informations. En gardant l'idée d'utiliser deux réseaux différents pour les deux types de contraintes, nous proposons un modèle qui diffère de celui de Kautz et Ladkin au niveau de la définition des contraintes métriques. Bien que notre modèle soit moins expressif, il permet, comme nous allons le montrer dans la section suivante, une homogénéité entre la représentation des deux types de contraintes (puisque on ne manipule que des intervalles) ce qui facilite le traitement de consistance des deux types de contraintes et offre donc une grande puissance de calcul aux algorithmes de consistance utilisés.

2.2 Notions de base : Définitions

Droite temporelle : Le modèle que nous proposons est basé sur une représentation discrète du temps. Nous définissons une référence temporelle **Tr** (cf figure 1) comme l'ensemble discret et adjacent d'unités de base $\mathbf{U_i}$. Chaque unité $\mathbf{U_i}$ représente le grain le plus fin accessible sur cette référence temporelle.

Intervalle : Un intervalle **I** est un ensemble d'unités adjacentes $\mathbf{U_k}$. Il est représenté par le couple d'unités (U_i, U_j), où U_i et U_j sont respectivement sa date de début et sa date de fin.

Fenêtre temporelle : Dans notre modèle, les contraintes numériques sont exprimées sous forme de fenêtres temporelles. Une fenêtre temporelle est l'ensemble des occurrences possibles d'un événement donné (cf figure 1). On parlera alors de Domaines d'Occurrences Possibles d'un événement ou tout simplement de DOP. Dans notre cas, un DOP est un ensemble fini d'intervalles avec une durée constante. Il est défini par : les dates de début au plus tot et de fin au plus tard de chaque intervalle appartenant au DOP, le pas définissant la distance minimale entre les dates de début de chaque paire d'intervalles adjacents, et la durée de chaque intervalle du DOP.

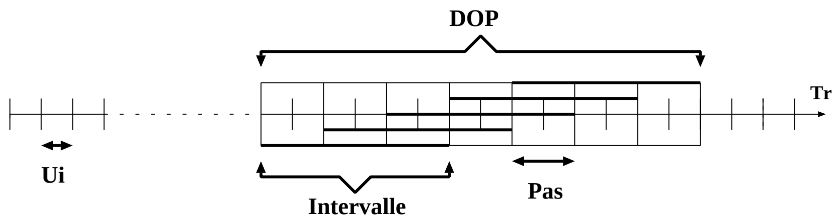


FIG. 1 –. *Représentation d'une contrainte numérique par un SOPO*

2.3 Le langage de représentation

Les événements : Les objets temporels que nous manipulons dans notre modèle sont appelés événements. Notre modèle étant basé sur la logique réifiée et plus exactement l'algèbre d'intervalles, nous définissons un événement par le couple (ϕ, I) où ϕ est une assertion logique atemporelle et I l'intervalle durant lequel ϕ est vrai. Par la suite on confondra entre la qualification temporelle de l'événement et l'événement lui même.

Les contraintes qualitatives : Relations symboliques entre événements Notre modèle étant basé sur la logique réifiée et plus précisément sur l'algèbre d'intervalles, nous utilisons, pour situer qualitativement un événement temporel par rapport à un autre, une disjonction des relations de base définies par Allen[Allen 83].
Soit e_1 et e_2 deux événements tel que $e_1 = (p, I_1)$ et $e_2 = (p, I_2)$, la contrainte qualitative entre ces deux événements est alors définie par l'expression $I_1 \;\; r_1 \vee r_2 \vee \ldots \vee r_n \;\; I2$ où chacun des r_i est l'une des 13 relations de base de Allen : "precedes", "during", "overlaps", "meets", "starts", "finishes" notées respectivement P, D, O, M, S et F; leurs inverses $P^{\smile}$, $D^{\smile}$, $O^{\smile}$, $M^{\smile}$, $S^{\smile}$, et $F^{\smile}$; et la relation égalité E.

Les contraintes quantitatives : Chaque événement est contraint numériquement à l'intérieur d'un DOP. Nous représentons un DOP par le quadruple $[datedebut,\ datefin,\ pas,\ duree]$ où $datedebut$ est la date de début au plus tôt du DOP, $datefin$ la date de fin au plus tard du DOP, pas le pas de discretisation du DOP (nombre d'unités de base) et $duree$ la durée de l'intervalle appartenant au DOP (durée de l'événement).

Soit e_i un événement contraint numériquement par le DOP suivant : $DOP(i) = [inf_i,\ sup_i,\ p_i,\ d_i]$, nous définissons l'ensemble E_i des occurrences possibles de e_i, à partir de $DOP(i)$, comme suit :
$E_i = \{occj \mid debut(occ_j) = inf_i + k * p_i, fin(occ_j) = debut(occ_j) + d_i * p_i,$
$fin(occ_j) <= sup_i, k \in [0, \frac{sup_i - inf_i}{p_i} - d_i] \cap \aleph\}$ où
$debut$ et fin sont des fonctions appliquées à des intervalles numériques et renvoyant respectivement le début et la fin d'un intervalle donné.

2.4 Réseau de contraintes temporelles numériques et symboliques

Afin de faciliter le processus de traitement des contraintes numériques et symboliques décrites précédemment, nous modélisons tout problème de contraintes par un graphe temporel dont les nœuds représentent les événements et les arcs les relations symboliques binaires entre événements. À chaque nœud du graphe est attaché l'ensemble des occurrences possibles de l'événement correspondant, le DOP. Cet ensemble constitue la contrainte métrique de l'événement auquel il est attaché.

Exemple : Soit 3 tâches T_1, T_2 et T_3 devant être réalisées par une certaine machine mono processeur M (M ne peut effectuer qu'une seule tâche à la fois) et une quatrième tâche T_4 devant être exécutée avant les tâches T_1 et T_2 et supposons que les informations numériques sur ces tâches soient les suivantes :

- Tâche 1 : durée = 3h, date de début = 10h, date de fin = 15h.
- Tâche 2 : durée = 3h, date de début = 20h, date de fin = 24h.
- Tâche 3 : durée = 4h, date de début = 7h, date de fin = 12h.
- Tâche 4 : durée = 1h, date de début = 9h, date de fin = 11h.

Ce problème d'ordonnancement peut être représenté par le graphe temporel de la figure 2 . Ce graphe est rendu complet par l'ajout d'arcs étiquetés avec la relation universelle I (disjonction des 13 relations de base de Allen[Allen 83]) entre chaque paire de nœuds n'ayant pas de contraintes symboliques entre eux (dans le cas où l'information symbolique concernant deux événements donnés n'est pas disponible, cas des événements T_3 et T_4).

3 Résolution de problèmes de satisfaction de contraintes temporelles

3.1 Problème de satisfaction de contraintes

Soit N un ensemble de variables $\{X_1 \ldots X_n\}$ tel que chaque variable x_i est définie sur un domaine discret $\{a_{i1} \ldots a_{in}\}$ et R un ensemble de contraintes (nous nous limitons dans ce qui suit aux relations binaires [1]) sur un sous ensemble de ces variables. Un problème de satisfaction de contraintes (appelé CSP

1. Notons qu'il est possible de convertir un problème CSP ayant des contraintes n-aires vers un CSP équivalent ayant des contraintes binaires

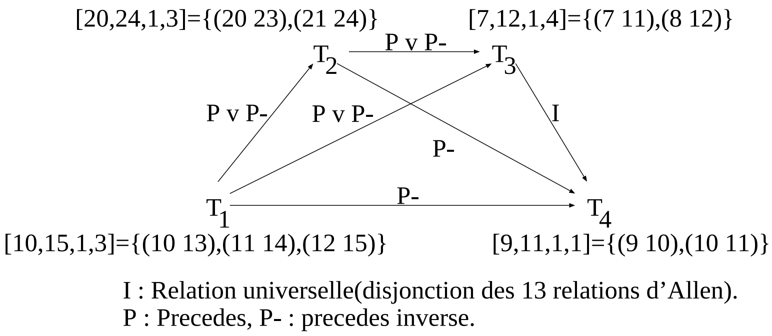


FIG. **2** –. *Graphe temporel représentant un problème d'ordonnancement de tâches*

pour Constraint Satisfaction Problem) consiste à déterminer tous les ensembles de valeurs $\{a_{1j1} \ldots a_{1jn}\}$ pour $\{X_1 \ldots X_n\}$ satisfaisant toutes les relations de R. Le réseau $G = (N, R)$ caractérisant un problème de satisfaction de contraintes est représenté généralement par un graphe dans lequel les nœuds représentent les variables et les arcs les relations binaires entre variables.
L'algorithme classique utilisé pour résoudre un tel problème est un algorithme de recherche de solutions avec retour arrière. Les problèmes de satisfaction de contraintes étant des problèmes NP-difficile, un tel algorithme est très coûteux surtout lorsque l'espace de recherche est très grand. Afin d'améliorer le recherche de solutions, des algorithmes de consistance locale sont utilisés comme filtre dans le but de réduire l'espace de recherche de solutions en éliminant de chaque domaine de variable, tous les éléments qui ne respectent pas les contraintes locales. Ces algorithmes (appelés aussi algorithmes de k-consistance) suppriment toutes les inconsistances qui ne vérifient pas les contraintes entre tous les sous-ensembles de k variables appartenant à N. On parle alors de consistance d'arcs lorsque $k = 2$ et de chemins lorsque $k = 3$. Le problème de k-consistance est un problème polynômial en temps $O(n^k)$. Un algorithme de consistance d'arcs ou de chemins ne permet pas de résoudre un problème de satisfaction de contraintes mais le simplifie. Une étape de recherche de solutions est alors nécessaire pour résoudre le problème.

3.2 Algorithme de résolution de problèmes de satisfaction de contraintes temporelles

Soit un problème donné de satisfaction de contraintes temporelles numériques et symboliques. La résolution de ce problème consiste à vérifier sa consistance et déterminer une solution numérique. Cette solution est constituée d'un ensemble d'intervalles de temps vérifiant toutes les contraintes symboliques et numériques. Pour ce faire, nous ramenons le problème de contraintes en un problème d'étiquetage supporté par un graphe temporel. L'algorithme classique permettant de résoudre un tel problème est un algorithme de recherche avec retour arrière. Afin d'améliorer l'efficacité de cet algorithme nous utilisons les algorithmes de consistance d'arcs AC-3 et AC-7 sur les contraintes métriques et l'algorithme de consistance de chemins PC-2 sur les relations symboliques.

La première tâche de cet algorithme est la vérification de la consistance d'un problème de contraintes temporelles donné. Les problèmes ne vérifiant pas la consistance d'arcs ou de chemins (et qui ne sont donc pas consistants) sont détectés en phase de pre-traitement de l'algorithme général. D'un autre coté, les problèmes vérifiant la consistance d'arcs et de chemins mais qui ne sont pas consistants sont détectés lors de l'étape de recherche de solutions grâce à l'utilisation des algorithmes de consistance d'arcs durant le processus de recherche. Dans le cas de problèmes consistants l'algorithme détermine une, plusieurs ou toutes les solutions numériques. Dans ce qui suit, nous présentons les deux étapes de l'algorithme général.

1- Phase de pré-traitement : Dans cette phase nous appliquons d'abord l'algorithme de consistance de chemins PC-2 sur les relations symboliques du graphe de contraintes. En plus de détecter certains problèmes inconsistants (problèmes ne vérifiant pas la consistance de chemins) l'algorithme de consistance de chemins PC-2 permet de réduire le nombre de primitives de Allen par relation qualitative disjonctive ce qui minimise le nombre de tests effectué par la suite par les algorithmes de consistance d'arcs sur les relations numériques. Dans la deuxième partie de la phase de pre-traitement nous appliquons l'algorithme de consistance d'arcs AC-7. La fonction d'élagage utilisée par cet algorithme est basée sur la traduction en équations numériques des relations qualitatives entre événements. Cette étape permet de détecter certaines inconsistances (inconsistances d'arcs) et de réduire le nombre d'étiquettes par domaine de valeurs (occurrences possibles) d'événements ce qui permet de restreindre l'espace de recherche de solutions.

2- Phase de recherche de solutions : Comme nous l'avons précisé auparavant, nous optons pour un algorithme de recherche avec retours arrière. La méthode de recherche que nous proposons utilise l'algorithmes de consistance d'arcs AC-3 durant le processus de recherche de solutions. Elle se déroule comme suit :

> *Choisir un nœud donné du graphe et instancier la variable correspondante à une valeur possible de son domaine (toutes les autres valeurs seront alors éliminées de l'étiquette du nœud). Appliquer un algorithme de consistance d'arcs sur le graphe. Si le nouveau graphe vérifie la consistance d'arcs, choisir un autre nœud et instancier sa variable à une valeur donnée du domaine de cette dernière, répéter le processus jusqu'a ce que toutes les variables soient instanciées, on obtiendra alors une solution, la consistance du graphe est alors vérifiée. Si à une étape donnée de la recherche le graphe ne vérifie plus la consistance d'arcs, un retour arrière est effectué, la dernière variable sélectionnée est alors affectée à*

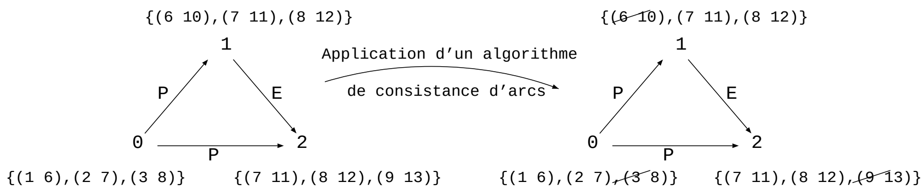
une autre valeur de son domaine. Si on n'arrive pas à instancier toutes les variables (i.e la consistance d'arcs ne peut être obtenue avec l'instanciation de toutes les variables), une solution numérique ne peut être trouvée, le graphe n'est donc pas consistent.

Afin d'améliorer la recherche, nous utilisons les techniques suivantes :

- L'ordre dans lequel les nœuds sont explorés étant très important [Sabin 94] [Haralick 80] [Nudel 83], nous ordonnons les différents nœuds dans l'ordre croissant du cardinal de leur domaine de valeurs. L'idée étant de choisir en premier les nœuds les plus restrictifs.
- L'appel à l'algorithme de consistance d'arcs n'est effectué que si le domaine de la variable choisie contient plus d'une valeur.

3.3 Consistance d'arcs dans un réseau de contraintes temporelles

Consistance d'arcs : Un réseau de contraintes temporelles représenté par notre modèle est arc-consistant si tous ses arcs sont arc consistants. Un arc (i, j) est arc consistant si et seulement si pour chaque valeur occ_i appartenant à D_i, il existe une valeur occ_j appartenant à D_j tel que la relation qualitative disjonctive R_{ij} de la paire de variables (EV_i, EV_j) est satisfaite. On dira alors que occ_i appartenant à D_i est supporté par occ_j appartenant à D_j. Le réseau de contraintes peut être converti en un réseau équivalent vérifiant la consistance d'arcs en appliquant un algorithme de consistance d'arcs (voir figure ci-dessous).



Algorithmes de consistance d'arcs : Les algorithmes de consistance d'arcs ont été largement étudié. Dans un ordre chronologique, nous pouvons citer l'algorithme de filtrage de Waltz[Waltz 72], AC-1, AC-2 et AC-3[Mackworth 77], AC-4[Mohr 86], AC-5 [Deville 91][van Hentenryck 92], AC-6[Bessire 94] et enfin AC7[Bessire 95]. Une étude comparative des algorithmes de consistance d'arcs que nous avons effectuée [Mouhoub 96] nous a permis d'opter pour l'algorithme AC-7 pour l'étape de filtrage de notre algorithme général et pour l'algorithme AC-3 pour l'étape de recherche de solutions. Dans ce qui suit, nous présentons ces deux algorithmes.

a) Algorithme AC-3 : L'algorithme AC-3, ci-après, appelé aussi "*algorithme de révision d'arcs*", se base sur la fonction $REVISE$ permettant de rendre consistant chaque arc (X_i, X_j) du graphe en éliminant toutes les valeurs du domaine de la variable X_i qui ne vérifient pas la contrainte entre X_i et X_j pour toute valeur de la variable X_j (i.e toute valeur de X_i qui n'est supportée par aucune valeur de la variable X_j). AC-3 utilise une structure de données contenant tous les arcs à réviser. Avant le déroulement de l'algorithme AC-3, cette liste est initialisée à tous les arcs du graphe et à chaque fois que le domaine de valeurs d'un nœud donné du graphe a été modifié, tous les arcs ayant pour origine ce dernier nœud sont ajoutés dans la liste.

Début
`Soit le graphe` $G = (U, E)$
  (`$U$: ensemble d'arcs, $E$: ensemble de noeuds`)
  $Q \leftarrow \{(i,j) \mid (i,j) \in U\}$
    `(liste contenant initialement tous les arcs de` G`)`
  Tant-Que $Q \neq Nil$ **Faire**
    $Q \leftarrow Q - \{(i,j)\}$
    Si $REVISE(i,j)$ **Alors**
      $Q \leftarrow Q \sqcup \{(k,i) \mid (k,i) \in U \wedge k \neq j\}$
    Fin-Si
  Fin-Tant-Que
Fin

Soit d le cardinal du plus grand domaine de variables, e le nombre d'arcs du graphe, la complexité de AC-3 est majorée par $O(ed^3)$[Mackworth 85].

b) Algorithme AC-7 : L'algorithme AC-7 (cf [Bessire 96] pour le code de l'algorithme AC-7) utilise un meta-niveau de connaissances pour inférer le support des valeurs des différentes variables afin de réduire le nombre de tests de consistance nécessaires aux autres algorithmes pour établir le support des différentes variables. En utilisant une propriété générale des contraintes, à savoir la "bidirectionalité" [2], AC-7 élimine, en effet, certains tests de consistance effectués par les autres algorithmes (AC-3, AC-4 et AC-6).

Aux deux algorithmes AC-3 et AC-7 nous avons apporté les améliorations suivantes :

1. Chaque contrainte numérique est représentée par une fenêtre temporelle définie par l'ensemble d'occurrences possibles de l'événement qu'elle contraint. Cet ensemble d'occurrences est croissant par rapport aux dates de début de chaque événement. Nous pouvons donc utiliser la propriété suivante pour minimiser le nombre de tests effectué par un algorithme de consistance d'arcs :
   Soit deux événements e_i et e_j ayant respectivement pour ensemble de valeurs D_i et D_j et soit R_{ij} la relation qualitative entre e_i et e_j :
   $$[P^{\smile} \in R_{ij} \wedge occ_{ik}\ P^{\smile}\ occ_{jl}] \Rightarrow \forall m > k \quad occ_{im}\ P^{\smile}\ occ_{jl} \qquad (1)$$
   où occ_{ik}, $occ_{im} \in D_i$ et $occ_{jl} \in D_j$.

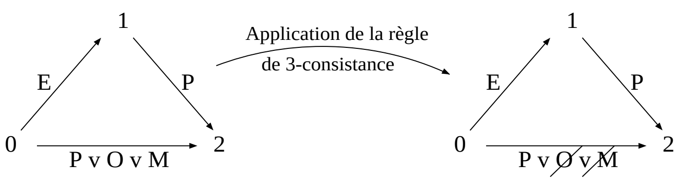
2. La notion de bidirectionalité stipule qu'une valeur a d'une variable i est supportée par une valeur b d'une variable j si et seulement si b de j est supportée par a de i.

Lorsque l'algorithme de consistance d'arcs est appliqué sur l'arc (i, j), cette propriété permet, si une certaine occurrence $occ_{ik} \in D_i$ est supportée par $occ_{jl} \in D_j$, étant donné la relation $P^{\smile}$, d'éviter de tester toutes les occurrences $occ_{im} \in D_i$ (avec $m > k$). La propriété (1) s'applique bien sur l'algorithme AC-3, elle est par contre difficile à mettre en œuvre pour l'algorithme AC-7. De plus, elle n'améliore pas l'efficacité de ce dernier algorithme.

2. Toutes les listes utilisées dans les deux algorithmes de consistance d'arcs sont maintenues en ordre croissant de leurs éléments ce qui permet de réduire le temps mis pour chercher si un élément appartient ou non à une liste donnée avant de l'ajouter à cette dernière.
3. L'algorithme AC-3 est appliqué d'une manière incrémentale, selon une stratégie donnée, dans l'étape de recherche de solutions de notre algorithme général. La différence entre les différentes stratégies, que nous allons présenter et comparer en sous section 4.3, est leur degré d'utilisation de l'algorithme AC-3.

3.4 Consistance de chemins dans un réseau de contraintes temporelles

Un réseau de contraintes temporelles représenté par notre modèle vérifie la consistance de chemins (pour les contraintes symboliques) si tous ses chemins sont consistants. Lorsque les chemins sont de longueur 2, la consistance de chemins est appelée 3-consistance. Le réseau vérifie la 3 consistance si pour chaque triplet (i, j, k) $\; r \in R_{ik} \Rightarrow r \in R_{ij} \circ R_{jk}$ où r est une relation de base de Allen. Les relations qualitatives R_{ij}, R_{ik} et R_{jk} sont des disjonctions de relations de Allen et $R_{ij} \circ R_{jk}$ est calculée en utilisant la table de composition des relations de base définie par Allen. Montanari[Montanari 74] montre que la 3-consistance implique la consistance de chemins.
Un réseau de contraintes temporelles peut être converti en un réseau équivalent vérifiant la consistance de chemins en appliquant un algorithme de consistance de chemins sur les relations qualitatives du réseau (voir figure ci-dessous). Une étude comparative des deux algorithmes de consistance de chemins PC-1 et PC-2 nous a permis d'opter pour l'algorithme PC-2 pour assurer la consistance de chemins dans un graphe de contraintes temporelles représenté par notre modèle [Mouhoub 96]. L'algorithme PC-2[Mackworth 77] représente, dans son principe, le pendant de l'algorithme AC-3 proposé par le même auteur. Dans PC-2, une structure de données est utilisée pour stocker, à chaque fois qu'une relation est modifiée, tous les arcs appartenant aux triplets de nœuds dans lesquels cette relation est considérée.



Nous utilisons la composition de matrices relationnelles pour appliquer l'algorithme PC-2. Nous représentons donc notre réseau de contraintes symboliques par une matrice de relations qualitatives (de taille n x n où n est le nombre de variables du réseau) et tel que chaque entrée M_{ij} correspond à la relation qualitative R_{ij} entre la paire de nœuds (i, j). L'intersection et la composition définies sur les relations qualitatives disjonctives sont alors définies comme le produit et la composition des matrices relationnelles :

$$(M \cdot M')_{ij} = M_{ij} \cdot (M')_{ij}$$
$$(M \circ M')_{ij} = \prod_{k \leq n} M_{ik} \circ (M')_{kj}.$$

La règle de transitivité sur laquelle se base l'algorithme PC-2 pour assurer la consistance de chemins est de ce fait traduite comme suit :

$$m_{ij} = m_{ij} + \sum_{k=0}^{n} m_{ik} * m_{kj} \text{ où}$$

$+$: Opérateur de conjonction de 2 relations disjonctives et
$*$: Opérateur de composition de 2 relations disjonctives (basé sur la table de transitivité de Allen[Allen 83]).

Pour améliorer l'efficacité de l'algorithme PC-2, nous utilisons la propriété suivante : $\forall i, j \quad M_{ji} = inverse(M_{ij})$ où l'opération *inverse* d'une relation disjonctive représente l'inverse de chaque élément de cette dernière relation.

PC-2 utilise la règle de 3-consistance pour réduire les relations qualitatives en éliminant certaines primitives de Allen inconsistantes. L'application de la règle de 3-consistance est basé sur le calcul de la composition des relations qualitatives. Ce calcul étant très coûteux, nous avons étudié les relations qualitatives dans le but de déterminer des cas où un tel calcul peut être évité. Ces cas sont les suivants. Soit R_1 et R_2 deux relations qualitatives.

- $R_1 \circ E = E \circ R_1 = R_1$
- $R_1 \circ I = I \circ R_1 = I$
- $[P \in R_1 \vee P^{\smile} \in R_2] \Rightarrow R_1 \circ R_2 = I$
- $[P^{\smile} \in R_1 \vee P \in R_2] \Rightarrow R_1 \circ R_2 = I$
- $[D \in R_1 \vee D^{\smile} \in R_2] \Rightarrow R_1 \circ R_2 = I$

4 Étude comparative des techniques de propagation de contraintes temporelles

Nous présentons dans cette section une étude comparative des différentes techniques que nous avons utilisées dans notre algorithme général. Cette étude a pour but de déterminer, parmi ces techniques, celles qui sont les plus efficaces à résoudre des problèmes de contraintes temporelles représentés par le modèle que nous avons proposé. Tous les tests que nous allons présenter sont effectués sur des problèmes de contraintes temporelles numériques et symboliques générés aléatoirement de la façon que nous allons décrire dans ce qui suit. Ces tests sont réalisés sur une station SUN SPARC 20 ayant 64 Mo de mémoire vive. Les problèmes de contraintes temporelles générés sont caractérisés par la dureté de leur contraintes définie comme suit :

> *Soit une contrainte donnée entre deux variables. La dureté de cette contrainte est la fraction du nombre de couples, du produit cartésien du domaine de ces deux variables, qui ne respectent pas cette contrainte[Sabin 94].*

En utilisant la définition précédente, nous pouvons calculer la dureté des contraintes en utilisant la formule suivante :

$$Durete(C_{ij}) = \frac{|\{(v_i, v_j) \mid \neg(v_i\ C_{ij}\ v_j)\}|}{|D_i\ \times\ D_j|}$$

La dureté d'un problème de contraintes est proportionnelle au nombre d'appels de l'algorithme de consistance d'arcs durant l'étape effective de recherche de solutions. En effet, lorsque la valeur de la dureté est grande, le problème correspondant est sur-contraint. L'espace de recherche se réduit alors à la solution numérique après seulement quelques appels de l'algorithme de consistance d'arcs. D'un autre coté, lorsque la valeur de la dureté est petite, la solution est obtenue après plusieurs appels de l'algorithme de consistance d'arcs.

4.1 Générateur de problèmes de contraintes temporelles

Afin d'effectuer les tests de comparaison que nous allons voir par la suite, nous avons réalisé un générateur de contraintes symboliques et numériques permettant de générer d'une manière aléatoire des exemples de problèmes de contraintes temporelles constitués chacun d'un ensemble de contraintes symboliques et numériques. Ces problèmes générés sont consistants (contiennent au moins une solution numérique) et sont caractérisés par la valeur de leur dureté, déterminée par les deux paramètres suivants :

1- l'Horizon : Date avant laquelle tous les événements doivent être réalisés, et
2- Nb-Allen : Nombre maximal de primitives de Allen par relation symbolique.

Dans ce qui suit, nous présentons les trois étapes de notre algorithme de génération.

Génération aléatoire de la solution numérique : Soit N le nombre de variables du problème de contraintes à générer, l'algorithme génère aléatoirement N couples d'entiers compris entre 0 et l'horizon. Chaque couple correspond à un intervalle du temps, les éléments des couples générés doivent donc vérifier la contrainte suivante : $\{\ (x, y) \mid x < y\})$. Ces N couples d'intervalles générés aléatoirement constituent la solution numérique à partir de laquelle nous construisons un ensemble de contraintes numériques et symboliques.

Génération des contraintes symboliques : À partir de la solution numérique, nous déterminons la solution symbolique correspondante constituée d'un ensemble de relations de base de Allen vérifiant la solution numérique. À chaque atome (primitive de Allen) de la solution symbolique, nous ajoutons un nombre de relations atomiques choisies aléatoirement et appartenant à l'intervalle [0,Nb-Allen - 1] afin d'obtenir une relation symbolique constituée d'une disjonction de primitives de Allen. Nous obtenons alors un ensemble de contraintes symboliques contenant au moins une solution symbolique.

Génération de contraintes numériques : Pour chaque intervalle de la solution numérique, nous générons un autre intervalle qui le contient, en tirant aléatoirement un couple d'entiers correspondant aux bornes de cet intervalle, et qui correspond à sa fenêtre temporelle. Cette dernière aura pour durée, la durée de l'intervalle qu'elle contient.
Les problèmes générés sont caractérisés par la valeur de leur dureté, déterminée par les deux paramètres Horizon et Nb-Allen. En effet la dureté d'un problème de contraintes temporelles est inversement proportionnelle à ces deux paramètres. Ceci s'explique de la manière suivante :
La valeur de la dureté d'une relation symbolique entre deux événements est inversement proportionnelle au nombre d'atomes de la relation symbolique. La dureté d'un problème de contraintes est donc inversement proportionnelle à **Nb-Allen**. D'un autre coté, comme nous l'avons précisé, les atomes de la relation symbolique sont déterminés à partir des couples de valeurs (intervalles) tirés aléatoirement dans l'intervalle $[0, Horizon]$. De ce fait, plus la valeur de l'horizon est grande et moins il y'a de chance aux atomes M, $M^{\smile}$, S, $S^{\smile}$, F, $F^{\smile} et E$ d'appartenir à la solution numérique. Ces atomes étant plus contraignants que les autres primitives de Allen, nous déduisons alors que la valeur de la dureté de contraintes est inversement proportionnelle à la valeur de l'horizon. Nous pouvons donc générer des problèmes avec différentes valeurs de dureté en faisant varier les paramètres horizon et Nb-Allen.

Les tests que nous présentons dans cette section sont effectués comme suit : Chaque test, correspondant à une valeur donnée de la dureté, est effectué sur 1000 exemples de problèmes de contraintes générées aléatoirement de la façon que nous venons de décrire. En effet, le paramètre essentiel étant la dureté des problèmes, nous faisons à chaque fois varier les paramètres Nb-Allen et Horizon pour avoir différentes valeurs de la dureté des contraintes. Les problèmes générés possèdent 100 variables chacun. Le domaine de chaque variable contient 20 valeurs en moyenne.

4.2 Utilité de la consistance de chemins pour la recherche de solutions numériques

Rappelons que l'algorithme de consistance de chemins PC-2 est appliqué dans notre algorithme général pour réduire les relations symboliques utilisées par la suite par les algorithmes de consistance d'arcs pour élaguer les différentes étiquettes inconsistantes. Cependant, un algorithme de consistance de chemins est très coûteux en temps d'exécution. Nous avons donc effectué des tests pour vérifier si ce temps est compensé par le temps global de la recherche et déterminer par conséquent si la consistance de chemins est utile pour la recherche de solutions numériques. Nous comparons donc les deux méthodes suivantes :

PC-2+AC-3+AC-3 : PC-2 est utilisé pour la propagation de contraintes symboliques, AC-3 assure la propagation numérique dans les deux phases de l'algorithme général.

AC-3+AC-3 : Seule la consistance d'arcs, assurée par AC-3, est appliquée dans cette méthode.

La figure 3a présente les temps CPU moyens mis par chacune des deux méthodes pour obtenir la première solution numérique de problèmes générés aléatoirement.

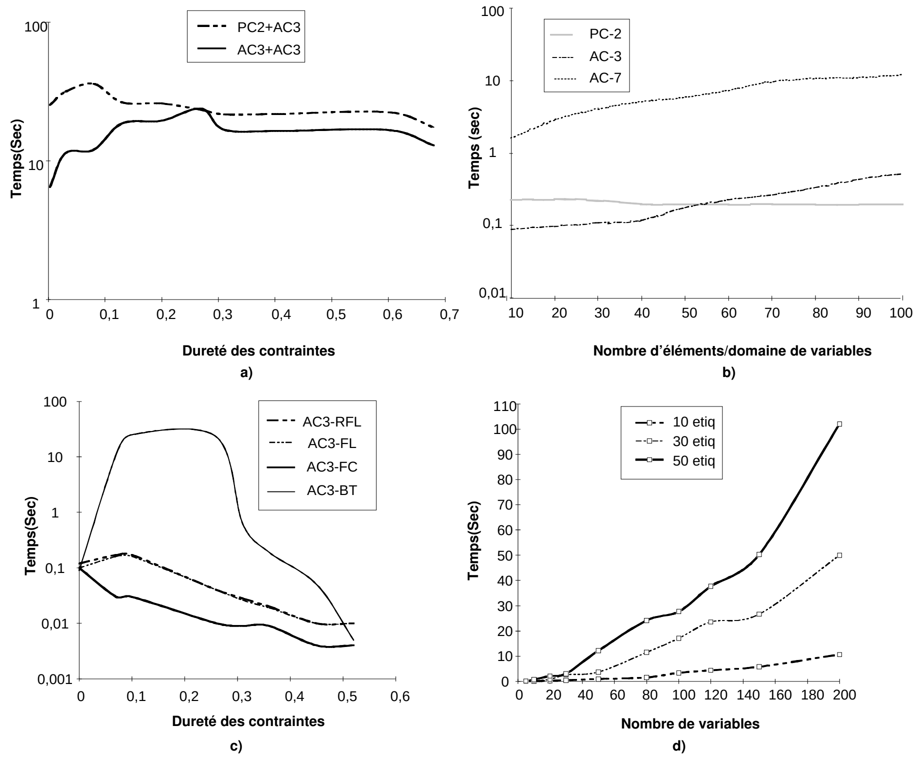


FIG. **3** –.

D'après les résultats, nous constatons que l'application de la consistance de chemins utilisant l'algorithme PC-2 n'améliore pas le temps global de la recherche, bien au contraire, le temps perdu par l'algorithme de consistance de chemins n'est pas compensé par la suite. En effet, nous remarquons que la deuxième méthode qui n'utilise pas de consistance de chemins est la plus performante en temps. Ceci s'explique par le fait que, comme nous venons de le signaler, un algorithme de consistance de chemins est très coûteux en temps. Ce temps est généralement perdu lors du calcul de la composition de relations qualitatives nécessaire pour vérifier la 3-consistance sur chaque triplet d'arcs. Ce calcul nécessite généralement l'accès à la table de transitivité de Allen ce qui est très coûteux en temps. Bien que nous avons minimisé le nombre d'accès en déterminant des cas où ce dernier peut être évité, un grand nombre d'accès reste nécessaire pour assurer la consistance de chemins. De plus, un graphe de contraintes symboliques est rendu complet durant l'application de l'algorithme de consistance de chemins. En effet, pour appliquer la règle de 3-consistance sur les arcs du graphe, nous devons ajouter des contraintes (des arcs) entre variables si ces dernières n'existent pas. Dans le cas des relations temporelles, nous ajoutons la relation universelle I, correspondant à la disjonction des 13 primitives de Allen, entre chaque paire de variables qui ne sont pas reliées qualitativement par une contrainte symbolique. Cela engendre les deux problèmes suivants :

1. Alors que nous utilisons l'algorithme de consistance de chemins dans le but de réduire le nombre de primitives de Allen par relation qualitative, ce dernier algorithme augmente, par l'ajout de nouvelles relations, la taille du problème de contraintes. Le nombre de tests effectué par la suite par les algorithmes de consistance d'arcs va, par conséquent, augmenter aussi.
2. Nous sommes amené, en utilisant la consistance de chemins, à modifier la structure même des problèmes de contraintes.

Toutefois, bien que la consistance de chemins est assez coûteuse pour déterminer les différentes solutions numériques d'un problème de contraintes temporelles donné, elle est indispensable dans le cas de problèmes de contraintes dans lesquels l'information numérique n'est pas disponible. De plus, la consistance de chemins est plus efficace que la consistance d'arcs à détecter l'inconsistance des problèmes de contraintes. en effet, nous avons effectué des tests sur des problèmes générés aléatoirement mais ne contenant pas de solution numérique au départ (la première étape de notre algorithme de génération qui consiste à générer aléatoirement une solution numérique est supprimée). Tous ces problèmes générés sont inconsistants et possèdent 200 variables chacun.

La figure 3b présente les résultats des tests de comparaison des temps mis par chacun des trois algorithmes AC-3, AC-7 et PC-2 à déterminer l'inconsistance du problème de contraintes. Nous remarquons d'après la figure que PC-2 est le plus performant des trois algorithmes pour des problèmes ayant plus de 50 valeurs par domaine de variables.

4.3 Comparaison des stratégies de recherche de solutions

Notre objectif est d'offrir à l'utilisateur du système que nous concevons une rapidité d'exécution ainsi qu'un temps de réponse acceptable permettant de résoudre des problèmes de contraintes temporelles. Pour ce faire, nous devons déterminer, parmi les différentes stratégies de recherche de solutions[Haralick 80], la plus rapide en temps d'exécution. La différence entre ces stratégies que nous allons présenter et comparer est le degré d'utilisation de la consistance d'arcs lors de la recherche de solutions numériques. Bien que la stratégie que nous ayons adoptée jusqu'a présent, appelée "*Real Full Look ahead*", utilise l'algorithme de consistance d'arcs, à chaque assignation de variables, sur tous les nœuds du graphe et réduit donc considérablement l'espace de recherche, cette stratégie consomme plus de temps que les autres stratégies. Ce temps est-il compensé par le temps global de recherche de solutions? les tests que nous présentons par la suite permettront de répondre à cette question.

1- Real Full Look ahead (RFL) À chaque instanciation de variables, AC-3 est appliqué sur tous les nœuds du graphe. C'est la technique que nous avons utilisée jusqu'a présent.

2- Full Look ahead (FL) Dans cette technique, la consistance d'arcs est appliquée entre le nœud courant et les nœuds futurs (nœuds non instanciés) d'une part, et entre les nœuds futurs entre eux d'autre part. Pour appliquer cette technique, nous utilisons l'algorithme AC-3 de la technique RFL (Real Full Look ahead) avec les changements suivants :

1. La liste Q est initialisée non pas à tous les arcs ayant pour extrémité le nœud courant mais uniquement ceux, parmi ces derniers, qui ont pour origine un nœud futur. À la ligne 3 de l'algorithme AC-3 nous aurons :

$$Q \leftarrow \{(k,i) \mid (k,i) \in U \;\wedge\; k \texttt{ nœud futur}\}$$

2. De même, lorsqu'il y'a un changement dans un nœud donné, on ne met pas tous les arcs ayant pour extrémité le nœud qui vient d'être modifié dans la liste Q mais uniquement ceux ayant pour origine un nœud futur. À la ligne 8 de l'algorithme AC-3 nous aurons :

$$Q \leftarrow Q \sqcup \{(j,k) \mid (j,k) \in U \wedge j \texttt{ nœud futur}\}$$

3- Forward Checking (FC) Cette technique est une variante de la technique FL. La consistance d'arcs est appliquée ici seulement entre le nœud courant et les nœuds futurs. Pour appliquer cette technique, nous utilisons l'algorithme AC-3 de la technique FL (Full Look ahead) en supprimant la ligne 8 (ce qui évite la propagation du test de consistance d'arcs entre les nœuds futurs).

4- Backtraking (BT) Dans cette technique, la consistance d'arcs est appliquée entre le nœud courant et les nœuds passés (nœuds instanciés). Nous utilisons dans ce cas l'algorithme AC-3 de la technique FC (Forward Checking) en modifiant la ligne 2 de la manière suivante :

$$Q \leftarrow \{(i,k) \mid (i,k) \in U \wedge \;k \texttt{ nœud passé}\}$$

Nous avons testé les 4 stratégies sur des problèmes de contraintes contenant chacun 30 variables. Chaque domaine de variables contient 20 valeurs. Chaque test, correspondant à une valeur donnée de la dureté, est effectué sur 1000 problèmes générés aléatoirement. La figure 3c présente les résultats de ces tests. La stratégie du backtracking est celle qui utilise le moins de tests de consistance d'arcs à chaque fois qu'une variable donnée est instanciée par une valeur de son domaine. En effet, AC-3 est appliqué, dans ce cas, seulement entre les nœuds passés et le nœud courant. Tous ces nœuds étant instanciés, les tests de consistance d'arcs s'effectuent sur des variables dont le domaine contient une seule valeur. Bien que cette stratégie soit la plus rapide dans certains cas où la dureté des contraintes tend vers 0, c'est la moins bonne dans le cas général. En effet, étant donné qu'il y a très peu de valeurs inconsistantes détectées, l'algorithme de recherche de solutions passe beaucoup de temps à effectuer des retours arrière avant de trouver la première solution au problème.
Les stratégies "*Real Full Look ahead*" et "*Full Look ahead*" sont, parmi les stratégies que nous avons comparées, celles qui enlèvent le plus d'inconsistances à chaque assignation de variables. Ceci permet de réduire considérablement l'espace de recherche de solutions et minimise par conséquent le nombre de retours arrière effectués pour déterminer la première solution. Cependant, le temps perdu pour enlever ces inconsistances n'est pas compensé par le temps mis pour déterminer la première solution numérique. En effet, nous constatons, d'après les résultats, que la stratégie "*Forward Checking*" qui enlève moins d'inconsistances que les stratégies RFL et FL présente un temps global de recherche meilleur que ces deux dernières stratégies. Nous rejoignons donc Nudel[Nudel 88] et Haralick[Haralick 80] pour affirmer que la stratégie "*Forward Checking*" est la meilleure en temps d'exécution.

5 Évaluation de notre outil de raisonnement

Afin de déterminer les performances, en temps, de notre outil à résoudre des problèmes de contraintes temporelles symboliques et numériques ainsi que les problèmes d'ordonnancement, nous avons effectués des tests expérimentaux d'évaluation sur des problèmes de contraintes temporelles générés aléatoirement.

5.1 Recherche de solutions numériques

Nous avons effectué des tests permettant de déterminer les temps moyens mis pour obtenir la première solution numérique et vérifier par conséquent la consistance de problèmes de contraintes temporelles numériques et symboliques générés aléatoirement de la façon que nous avons décrite précédemment. Chaque test, correspondant à un nombre de variables donné du problème, est effectué sur 100 exemples de problèmes consistants générés aléatoirement.
La figure 3d présente les temps CPU moyens mis pour déterminer la première solution numérique pour des problèmes ayant entre 5 et 200 variables. Chaque

courbe correspond à un nombre moyen donné d'étiquettes par domaine de variables. La dureté des problèmes générés est de 0.10 en moyenne, cela correspond aux problèmes les plus difficiles à traiter.
Les courbes de la figure 3d montrent que le temps d'exécution est directement proportionnel au nombre d'étiquettes à propager sur chaque nœud. La première phase de l'algorithme général consomme beaucoup plus de temps que la phase de recherche de solutions. En effet, étant donné que nous utilisons une version incrémentale de l'algorithme AC-3 lors de la recherche de solutions numériques, il effectue beaucoup moins de tests de consistance que AC-7 appliqué dans la phase de filtrage.

5.2 Résolution de problèmes d'ordonnancement de tâches non-préemptives

Le problème d'ordonnancement de tâches est un cas particulier de problèmes de satisfaction de contraintes temporelles dans lequel nous cherchons à respecter des contraintes temporelles associées à des tâches partageant une ou plusieurs ressources. La résolution d'un problème d'ordonnancement consiste à ordonner les tâches afin de satisfaire toutes les contraintes de ce dernier (contraintes temporelles liées à chaque tâche, disponibilité des ressources non partageables etc ...) de manière à respecter un critère donné (coût, rapidité etc ...). Ce genre de problèmes est un cas particulier de problèmes de satisfaction de contraintes symboliques et numériques pouvant être représentées par le modèle que nous proposons et résolu par notre outil de raisonnement.
Les problèmes que nous traitons dans cette section concernent l'ordonnancement de tâches non préemptives, indépendantes et partageant une seule ressource. La résolution d'un problème d'ordonnancement consiste à trouver une solution numérique à ce dernier respectant toutes ses contraintes et minimisant la date de fin d'exécution de la dernière tâche. Afin de déterminer les performances, en temps, de notre outil de raisonnement à résoudre de tels problèmes, nous avons effectué des tests expérimentaux sur des problèmes d'ordonnancement de tâches générés aléatoirement de la manière suivante :

Soit $\mathtt{Horizon}$ *la date de fin de disponibilité de la ressource (date avant laquelle toutes les tâches doivent être exécutées) et* $\mathtt{Duree} - \mathtt{max}$ *la durée maximale d'exécution de chaque tâche du problème à générer. Pour chaque tâche nous générons aléatoirement :*

- *sa durée* $\mathtt{d}$ *parmi* $[1 \ldots \mathtt{Duree} - \mathtt{max}]$,
- *la taille de sa fenêtre* $\mathtt{duree} - \mathtt{f}$ *parmi* $[\mathtt{d} + 1 \ldots \mathtt{Horizon}]$,
- *sa date de début au plus tôt* *parmi* $[0 \ldots \mathtt{Horizon} - \mathtt{duree} - \mathtt{f}]$.

Chaque test, caractérisé par une valeur de l'horizon et un nombre de tâches données, est effectué sur 1000 problèmes générés aléatoirement de la façon que nous venons de montrer. La durée maximale des tâches est fixée à 10 unités de temps.

Nous avons montré dans la sous section 3.3 une propriété permettant de minimiser le nombre de tests effectué par l'algorithme AC-3 pour assurer la

consistance d'arcs[3]. Cette propriété est applicable à chaque fois que la primitive de Allen $P^{\smile}$ appartient à une relation qualitative entre deux variables. Dans le cas particulier des problèmes d'ordonnancement, la primitive $P^{\smile}$ est présente dans toutes les relations qualitatives. Elle permet, par conséquent, de réduire considérablement le nombre de tests effectués par l'algorithme AC-3.
Bien que AC-7 soit le meilleur algorithme pour assurer la consistance d'arcs dans le cas général de problèmes de contraintes temporelles, il n'en est pas de même pour les problèmes d'ordonnancement. En effet, avec la propriété 1 que nous avons appliquée sur AC-3, ce dernier algorithme devient plus performant, en temps CPU, que AC-7.

Nous présentons dans la figure 4 les résultats des tests de comparaison des deux algorithmes correspondant aux temps CPU moyens mis pour assurer la consistance d'arcs sur des problèmes d'ordonnancement de tâches non-préemptives. Nous remarquons, d'après la figure 4, que AC-3 auquel nous avons appliqué la propriété 1 est plus performant que AC-7 pour assurer la consistance d'arcs. Dans le cas des problèmes d'ordonnancement nous utiliserons donc AC-3 dans les deux phases de l'algorithme général.

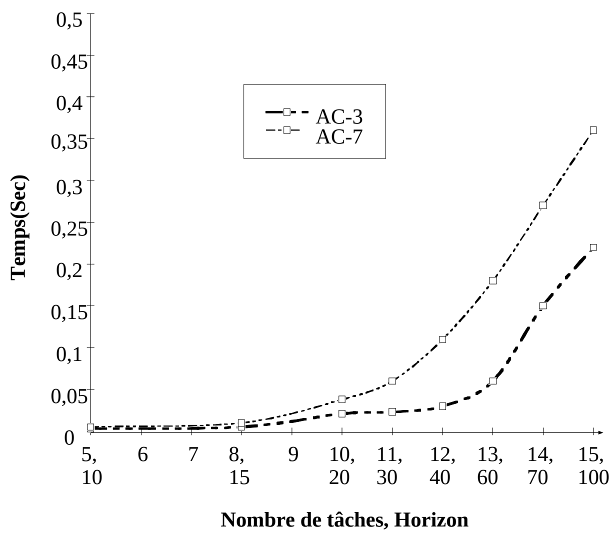


FIG. 4 –. *Performances en temps CPU pour assurer la consistance d'arcs.*

3. Notons, comme nous l'avons précisé en 7.3, que cette propriété s'applique difficilement sur les algorithmes AC-4 et AC-7 et n'améliore pas les performances de ces deux algorithmes

6 Conclusion

Nous avons présenté dans cet article un modèle de représentation des informations temporelles qualitatives et numériques et un outil de raisonnement associé à ce modèle. Le modèle de représentation est une extension de l'algèbre de Allen permettant de supporter les informations métriques. Ce modèle permet de traduire tout problème de satisfaction de contraintes temporelles qualitatives et numériques en un problème d'étiquetage représenté par un graphe temporel pouvant être résolu efficacement par l'outil de raisonnement que nous proposons. Cet outil est basé sur un algorithme général de recherche de solutions avec retour arrière. Ce dernier algorithme utilise des algorithmes de consistance locale selon une stratégie donnée de recherche de solutions dans le but de réduire l'espace de recherche et augmenter l'efficacité de l'algorithme général. Afin de déterminer jusqu'a quel degré la consistance d'arcs et de chemins sont utiles pour résoudre un problème de satisfaction de contraintes temporelles, nous avons effectué une étude comparative des différentes stratégies de recherche de solutions. Cette étude nous a permis de faire les conclusions suivantes :

- Bien que la consistance de chemins permet de réduire la taille des relations symboliques ce qui minimise le nombre de tests effectués par la suite lors de la recherche de solutions, elle est très coûteuse en temps d'exécution lorsqu'il s'agit de déterminer une ou plusieurs solutions numériques. Elle est cependant utile pour déterminer l'inconsistance des problèmes de contraintes temporelles.
- Les stratégies "*Real Full Look ahead*" et "*Full Look ahead*" sont, parmi les stratégies que nous avons étudiées, celles qui enlèvent le plus d'inconsistances à chaque assignation de variables. Ceci permet de réduire considérablement l'espace de recherche de solutions et minimise par conséquent le nombre de retours arrière effectués pour déterminer la première solution. Cependant, le temps perdu pour enlever ces inconsistances n'est pas compensé par le temps mis pour déterminer la première solution numérique. D'un autre coté, la stratégie du Backtracking qui enlève le moins d'inconsistance à chaque assignation est la plus rapide dans des cas particuliers étant donné qu'elle passe moins de temps que les autres stratégies à vérifier la consistance d'arcs mais pose des problèmes dans le cas général. L'algorithme de recherche effectue généralement beaucoup de tests de retour arrière avant de déterminer la première solution. Ceci est du au fait qu'il y'a peu d'inconsistances enlevées. Nous constatons enfin que la stratégie "*Forward Checking*" qui enlève moins d'inconsistances que les stratégies RFL et FL présente, par rapport aux autres stratégies, le meilleur temps d'exécution.
  Nous rejoignons donc Nudel[Nudel 88] et Haralick[Haralick 80] pour affirmer que cette stratégie est la meilleure en temps d'exécution.

Références

[Allen 83] J.F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11):832–843, 1983.

[Bessire 94] C. Bessire. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.

[Bessire 95] C. Bessire, E. Freuder et J.C. Regin. Using inference to reduce arc consistency computation. *IJCAI'95*, pages 592–598, Montral, Canada, 1995.

[Bessire 96] C. Bessire, A. Isli et G. Ligozat. Global consistancy in interval algebra networks : tractable subclasses. *ECAI'96*, Budapest, Hongrie, 1996.

[Deville 91] Y. Deville et P van Hentenryck. An efficient arc consistency algorithm for a class of CSP problems. *In Proc. IJCAI'91*, pages 325–330, Sidney, Australie, 1991.

[Haralick 80] R.M. Haralick et G.L. Elliott. Increasing tree search efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.

[Kautz 91] H.A. Kautz et P.B. Ladkin. Integrating metric and qualitative temporal reasoning. *AAAI-91*, pages 241–246, Anaheim, CA, 1991.

[Mackworth 77] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mackworth 85] A. K. Mackworth et E. Freuder. The complexity of some polynomial network-consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[Meiri 91] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *AAAI*, pages 260–267, 1991.

[Mohr 86] R. Mohr et T. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[Montanari 74] U. Montanari. Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[Mouhoub 96] M. Mouhoub, F. Charpillet et J.P. Haton. Comparison of Constraint Propagation Techniques for Interval Based Temporal Reasoning. *Proc of the AAAI Workshop on Spatial and Temporal Reasoning*, pages 127–134, Portland, 1996.

[Nudel 83] B. Nudel. Consistent-Labeling Problems and Their Algorithms : Expected Complexities and Theory-Based Heuristics. *Artificial Intelligence.*, 21:135–178, 1983.

[Nudel 88] B. Nudel. Tree search and arc consistency in constraint-satisfaction algorithms. *In Search in Artificial Intelligence.*, pages 287–342, 1988.

[Sabin 94] D. Sabin et E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. *Proc. 11th European Conference on Artificial Intelligence*, pages 125–129, Amsterdam, Hollande, 1994.

[van Hentenryck 92] P van Hentenryck, Y. Deville et C. M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[Vilain 86] M. Vilain et H. Kautz. Constraint propagation algorithms for temporal reasoning. *AAAI-86*, pages 377–382, Philadelphia, PA, 1986.

[Waltz 72] D. Waltz. Generating semantic description from drawing of scenes with shadows. *A.I. Memo n271,MIT AI Laboratory*, 1972.