Département de Formation Doctorale en Informatique **École Doctorale IAE + M**
UFR STMIA

Contribution à l'étude des techniques de propagation de contraintes symboliques et numériques pour le raisonnement temporel

THÈSE

présentée et soutenue publiquement le 9 décembre 1996

pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy I

(Spécialité Informatique)

par

Malek Mouhoub

Composition du jury

Président :	Hélène Kirchner, Directeur de recherche au CNRS, CRIN & INRIA Lorraine
Directeurs :	Jean Paul Haton, Professeur à l'U.H.P Nancy 1 François Charpillet, Chargé de recherche à l'INRIA, CRIN & INRIA Lorraine
Rapporteurs :	François Rousselot, Maître de conférences à l'U.S.H de Strasbourg Sylviane Schwer, Professeur à l'Université de Paris Villeutaneuse

Centre de Recherche en Informatique de Nancy
INRIA–Lorraine

Remerciements

Je tiens à remercier sincèrement :

– Mr Jean Paul Haton, professeur à l'Université H.P. Nancy I, de m'avoir accueilli dans l'équipe RFIA et d'avoir suivi mon travail de thèse, j'espère qu'il trouvera dans ce modeste rapport, l'expression de ma reconnaissance et de mes remerciements.

– Mr François Charpillet, Chargé de recherche à l'INRIA, de m'avoir encadré durant cette thèse, pour ses conseils, ses critiques et sa gentillesse.

– Monsieur François Rousselot, Directeur d'ERIC, Madame Sylviane Schwer, Professeur à l'Université de Paris Villeutaneuse et Madame Hélène Kirchner, Directeur de recherche au CNRS, pour avoir accepté d'être rapporteurs de ma thèse, et m'avoir fourni à cette occasion de nombreuses remarques et suggestions pour son amélioration.

– Madame Martine Kuhlmann, secrétaire de l'équipe RFIA pour son aide et sa disponibilité.

– Tous les membres du laboratoire CRIN & INRIA Lorraine qui m'ont aidé à la réalisation de mon travail et à la rédaction de mon rapport.

À Samia,
mes parents,
mes frères et sœurs,
et tous mes amis

Table des matières

Introduction générale

1 Présentation du sujet : Motivations et problématique

Le temps est un aspect essentiel en informatique et spécialement en intelligence artificielle dès lors que l'on s'intéresse au traitement des applications dans un univers évolutif. Dans une application capable de gérer un processus industriel, nous avons besoin de représenter aussi bien des connaissances temporelles liées à l'évolution du processus que le modèle contrôlant ce dernier. Dans des problèmes d'ordonnancement de tâches où le rôle du temps est significatif, nous cherchons à respecter des contraintes temporelles associées à des tâches partageant une ou plusieurs ressources.

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.
Plusieurs schémas de représentation des informations temporelles ont été proposés. Nous en distinguons deux aspects fondamentaux, l'aspect qualitatif et l'aspect quantitatif du temps.
Le premier concerne les informations symboliques permettant de décrire la relativité entre des événements, des actions ou des processus d'un point de vue temporel. Les schémas de représentation traitant cet aspect sont fondés sur la logique et plus particulièrement la logique dite "réifiée". Nous y distinguons l'algèbre d'intervalles de Allen[Allen 83] et l'algèbre d'instants de McDermott[McDermott 82].
Le second concerne les informations numériques permettant de situer un événement, une action ou un processus donnés dans ou par rapport à un référentiel temporel. Parmi les représentations traitant cet aspect nous distinguons les représentations fondées sur la recherche opérationnelle, où les informations temporelles sont traduites sous forme d'inéquations pouvant être résolues par l'algorithme du SIMPLEXE et celles utilisant une représentation par graphes temporels.
La séparation entre ces deux aspects n'existe pas dans la pratique. Dans nos activités de tous les jours, par exemple, nous combinons les deux types d'informations, numériques et symboliques, qui sont fortement imbriquées afin de décrire le monde en question. Le problème d'ordonnancement de tâches que nous venons d'évoquer constitue un autre exemple. Dans ce cas, les informations numériques spécifient les dates de début au plus tôt et de fin au plus tard des différentes tâches, et les informations symboliques décrivent des contraintes qualitatives de type disjonction temporelle de tâches, c'est à dire qu'une tâche donnée s'exécute nécessairement avant ou après une autre tâche. Enfin, des travaux ont été effectués ces dernières années afin de représenter les deux types d'informations dans un même modèle, nous parlerons dans ce cas là de représentations mixtes. Notre travail se situe dans ce cadre.

Notre premier objectif est de proposer une représentation du temps qui tienne compte des deux aspects, numérique et symbolique, dans un même modèle.
Après avoir défini un tel modèle nous devons être capable d'exploiter les informations qu'il représente et d'en tirer des conclusions. Nous devons par conséquent définir un système de raisonnement permettant de manipuler et d'interpréter les informations numériques et symboliques représentées dans notre modèle tout en tenant compte, dans des applications industrielles par exemple, du temps d'obtention des résultats. En effet, si cet outil a pour rôle d'ordonner des tâches dans un système de production donné, le temps mis par l'ordonnancement ne doit pas pénaliser le temps global d'exécution du système. Or proposant un modèle riche en expressi-

vité et capable de représenter des informations qualitatives et quantitatives, nous pénalisons ce temps. Nous sommes alors confrontés à des problèmes de complexité en temps lors de la manipulation de ces informations par un outil de raisonnement. Pour établir un tel outil en faisant face à ces problèmes, nous devons, en nous inspirant des techniques et méthodes permettant de traiter ce genre de problèmes, recourir à des heuristiques pour répondre aux besoins que nous venons d'évoquer et améliorer par conséquent les travaux effectués dans ce domaine.

2 Notre contribution

Notre but est de fournir un modèle expressif de représentation des informations temporelles qualitatives et quantitatives indépendant de tout domaine d'application ainsi qu'un outil de raisonnement capable de manipuler les informations représentées par ce modèle et fournir des résultats directement exploitables dans des délais de temps acceptables.
Afin de répondre à ces besoins, nous avons défini un modèle qui généralise l'algèbre de Allen[Allen 83] afin d'intégrer des informations métriques. Ce modèle se fonde sur cette algèbre pour l'aspect symbolique du temps. L'objet temporel que nous manipulons est l'événement correspondant au couple (ϕ, I) où ϕ est une assertion logique atemporelle et I l'intervalle de temps durant lequel ϕ est vraie. Les informations symboliques sont traduites sous forme de disjonctions de relations de base définies par Allen permettant de situer deux événements temporels entre eux. Les informations numériques sont exprimées sous forme de domaines de variation attachés à chaque événement et permettant de situer ce dernier dans ou par rapport à un référentiel temporel. Nous utilisons une représentation discrète du temps ce qui permet de représenter chaque domaine de variation d'un événement donné par un ensemble fini d'intervalles correspondant à toutes les occurrences possibles de l'événement.

Notre choix s'est porté sur l'algèbre de Allen puisqu'elle offre une expressivité riche des informations qualitatives. Cependant les problèmes traitant des systèmes fondés sur cette approche sont NP-complets.
Plusieurs travaux ont été effectués pour pallier ce problème de complexité. Ces travaux s'intéressent généralement à déterminer des sous classes de l'algèbre de Allen dans lesquelles la résolution de problèmes représentés par ces sous classes peut être obtenue par un algorithme polynômial[van Beek 90a][van Beek 92][Meiri 91][Nebel 95][Bessière 96]. Cependant, avec cette manière de procéder nous perdons en expressivité des informations qualitatives. Aucune de ses sous classes polynômiales ne contient, par exemple, les relations de précédence de Allen[Allen 83]. Ces relations sont néanmoins nécessaires pour exprimer des contraintes temporelles, notamment dans le domaine de la planification.
En ce qui nous concerne, nous voulons profiter de cette richesse d'expressivité de l'algèbre de Allen tout en faisant face à ce problème de complexité. Nous nous attaquons ainsi à une classe non polynômiale de problèmes et nous nous intéressons dans ce cas à la complexité en moyenne en temps. Le recours à des heuristiques est alors nécessaire pour résoudre ce genre de problèmes.

La solution que nous proposons est la suivante. Nous ramenons tout problème de contraintes temporelles représenté par notre modèle en un problème d'étiquetage pouvant être résolu efficacement par des techniques de satisfaction de contraintes.

Nous traduisons alors tout problème de contraintes temporelles en un graphe de contraintes qualitatives et numériques dans lequel les nœuds représentent les différents événements, les relations qualitatives entre événements sont attachées aux arcs et à chaque nœud est relié un ensemble d'intervalles correspondant au domaine de variation de l'événement.
Sur ce graphe nous appliquons des algorithmes de consistance locale pour résoudre le problème correspondant au graphe et exploiter les résultats dans des délais de temps acceptables. Pour faire face aux problèmes de complexité surtout lorsqu'il s'agit de problèmes de grande taille, nous avons défini un tel algorithme 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 avons proposé.

Nous avons également amélioré les algorithmes de consistance locale utilisés par notre algorithme en y introduisant des techniques que nous avons définies en étudiant les propriétés des contraintes temporelles.

3 Plan du rapport de thèse

Le rapport comporte 3 parties contenant chacune 4 chapitres. La première est divisée de la manière suivante :

- une étude de synthèse dans le domaine du raisonnement temporel permettant de dégager les différents concepts liés à la représentation des connaissances temporelles en vue de définir notre modèle de représentation des informations temporelles, et
- une présentation des différentes techniques permettant de concevoir un outil de raisonnement associé au modèle que nous proposons.

Nous présentons dans la deuxième partie d'une part notre modèle permettant de représenter des informations temporelles qualitatives et numériques et d'autre part l'outil de raisonnement qui lui est associé.

La troisième partie est d'abord consacrée à une étude expérimentale comparative des techniques que nous utilisons dans notre outil de raisonnement afin de dégager les avantages et inconvénients de chacune d'elles. Nous nous appuyons ensuite sur cette étude comparative pour améliorer notre outil de raisonnement et présentons des tests d'évaluation de performances de cet outil dans le traitement de problèmes de satisfaction de contraintes de natures différentes. Nous traitons en particulier les problèmes d'ordonnancement de tâches non-préemptives à la fin de cette partie.

Nous terminons enfin par une conclusion dans laquelle nous faisons le bilan de notre travail en soulignant notre contribution et les résultats que nous avons obtenus. Des prolongements possibles sont ensuite proposés en perspectives.

Première partie

État de l’art

Chapitre 1

Introduction

Le temps est d'une importance majeure dans de nombreuses applications en intelligence artificielle. En effet, un système intelligent doit pouvoir prendre en compte la composante temporelle dans son raisonnement sur un univers évolutif. L'élaboration, par exemple, d'un plan dans le domaine de la planification de tâches nécessite la prise en considération des durées de ces tâches ainsi que l'ordonnancement temporel le plus approprié étant donné leurs interactions dans le temps. De même, le contrôle de procédés industriels dans des systèmes d'aide à la supervision nécessite la prise en compte des différents états passés du procédé et des variables qui le font évoluer. Concevoir un système intelligent pour des tâches d'aide à la décision, de planification ou de contrôle de procédés industriels nécessite donc une représentation explicite de l'information temporelle sous ses deux formes symbolique et numérique et la prise en compte de connaissances temporelles spécifiques au domaine ou générales.

Nous distinguons deux types d'approches de représentation du temps : les approches fondées sur des représentations numériques du temps et celles qui reposent sur des représentations symboliques. Parmi les approches numériques, nous distinguons les approches classiques de recherche opérationnelle et les approches qui conjuguent des techniques d'intelligence artificielle avec la recherche opérationnelle.
Parmi les approches classiques nous distinguons les méthodes basées sur la programmation linéaire et celles utilisant une représentation par graphes temporels (méthodes du chemin critique). Ces méthodes sont efficaces dans des cas simples mais deviennent insuffisantes pour des problèmes complexes.
De nouvelles approches d'intelligence artificielle ont alors été fondées dans le but de pallier ces problèmes. Ces approches se basent sur une représentation des informations temporelles par des intervalles numériques. Plus précisément, les informations temporelles numériques sont interprétées sous forme de contraintes tendant à situer des événements, actions ou processus temporels de durée donnée dans ou par rapport à un référentiel temporel. Nous parlerons alors de domaines de variation de ces événements (fenêtres temporelles dans lesquelles les événements doivent se produire).

Pour les approches symboliques, les premiers travaux traitant cet aspect symbolique du temps se sont basés sur la logique classique. Cette logique (propositionelle ou du premier ordre) ne donne cependant aucun statut particulier au temps. Ce dernier est en effet pris en compte

comme un terme argument d'un prédicat.
Plus récemment, deux autres types de logiques ont traité l'aspect symbolique du temps. Ce sont les logiques modales temporelles et les logiques dites "réifiées ".

La logique modale temporelle constitue un cadre formel d'étude important pour la représentation et l'étude des propriétés du temps. Elle a donné lieu à des développements dans différents domaines tel que la preuve de programmes ou la vérification de circuits logiques. Dans cette logique, le temps représente une modalité associée au domaine d'interprétation d'une formule. La manipulation de ce dernier est effectuée via des règles d'inférence de la logique.

La logique réifiée, très utilisée en intelligence artificielle, associe au sein d'un prédicat global du genre $TRUE(P, I)$ un composant atemporel, la proposition P, à un composant temporel, I (I étant soit un intervalle soit un instant), pendant lequel P est considérée comme vraie. Parmi les approches se basant sur la logique réifiée, nous distinguons deux catégories : les approches fondées sur l'algèbre d'instants[McDermott 82] et les approches fondées sur l'algèbre d'intervalles [Allen 83].
Dans les approches fondées sur l'algèbre d'instants, l'entité de base est l'instant et les relations qualitatives possibles entre instants sont : $<, >$ et $=$.
Dans les approches fondées sur l'algèbre d'intervalles[Allen 83], les composants atemporels de base associés aux intervalles sont de trois types : "*les propriétés*", *"les événements"*, et "*les processus*". Les propriétés représentent l'aspect statique d'un procédé alors que les événements et les processus représentent les aspects dynamiques de celui-ci (décrivant les changements du procédé sur un intervalle de temps).

Après avoir défini une représentation des informations temporelles, nous devons être capable de les utiliser. Alors que les approches numériques classiques se basent sur la programmation linéaire pour raisonner sur les informations temporelles, les approches récentes, manipulant les domaines de variation d'événements représentés par des intervalles numériques, utilisent des techniques de satisfaction de contraintes afin d'assurer la propagation des différents changements sur ces domaines de variation. De manière analogue, les approches symboliques du temps basées sur la logique réifiée utilisent des techniques de propagation de contraintes pour raisonner sur les informations temporelles qualitatives représentées par des relations symboliques.

Dans le chapitre suivant nous présentons les différents travaux traitant l'aspect temporel sous ses deux formes numérique et symbolique. Nous décrivons les différentes représentations du temps et dégageons les avantages et inconvénients de chacune d'elles.
Dans le troisième chapitre nous parlerons des techniques de résolution de problèmes de satisfaction de contraintes. En effet, dans la plupart des représentations que nous allons présenter dans le second chapitre, l'information temporelle est interprétée sous forme de contraintes. Raisonner sur ces représentations nécessite par conséquent l'utilisation de techniques de satisfaction de contraintes. Nous présentons alors dans ce chapitre les différentes techniques permettant de résoudre des problèmes de satisfaction de contraintes, à savoir : les stratégies de parcours d'arbre ainsi que des algorithmes de satisfaction de contraintes permettant de réduire l'espace de recherche de solutions.

Chapitre 2

Représentation de l'information temporelle

Le temps est d'une grande importance dans de nombreuses applications en intelligence artificielle. Durant ces dernières années, plusieurs chercheurs se sont intéresses à la représentation de l'information temporelle. Cette représentation pouvant être de nature numérique ou symbolique, nous distinguons alors deux types d'approches : les approches qui s'intéressent à l'aspect numérique et celles qui s'intéressent à l'aspect symbolique. Parmi les approches numériques, nous distinguons celles fondées sur les techniques classiques de recherche opérationnelle et celles manipulant explicitement les domaines des variables. Parmi les approches symboliques, nous distinguons les approches fondées sur la logique du premier ordre, les approches fondées sur la logique modale temporelle et les approches fondées sur la logique dite "réifiée".
Nous présentons ces différentes approches dans ce qui suit en dégageant les avantages et les inconvénients de chacune d'elles.

2.1 Représentations numériques du temps

Les approches numériques consistent à représenter les informations temporelles par des nombres (des nombres réels généralement). Les événements temporels sont alors situés numériquement dans un référentiel temporel avec une éventuelle tolérance (fenêtres temporelles avec dates au plus tôt et au plus tard). Ces approches empruntent largement à la recherche opérationnelle. Certaines d'entre elles utilisent toutefois d'autres symboles que les nombres (nombres réels), pour dénoter en particulier une relation entre deux intervalles. Cependant, le maintien de cohérence temporelle passe toujours par une modification de nombres.
Il existe deux types d'approches numériques : les approches utilisant les techniques classiques de recherche opérationnelle et les approches manipulant explicitement les domaines d'intervalles.
Parmi les approches utilisant les méthodes classiques de recherche opérationnelle, nous distinguons les approches fondées sur l'algorithme du SIMPLEXE et les approches utilisant la

représentation par graphes temporels.
Les approches manipulant explicitement les domaines d'intervalles, quant à elles, conjuguent des techniques d'intelligence artificielle avec des méthodes de recherche opérationnelle.

Dans ce qui suit, nous présentons chacune de ces deux approches.

2.1.1 Approches fondées sur la recherche opérationnelle

a) Approches basées sur la programmation linéaire : Algorithme du SIMPLEXE

En associant un intervalle à chaque événement temporel, les approches basées sur la programmation linéaire permettent d'exprimer, sous forme d'inéquations linéaires en fonction des dates de début et de fin de chaque événement, la plupart des relations temporelles. L'algorithme du SIMPLEXE est alors utilisé sur les inéquations qui définissent les contraintes temporelles linéaires d'un problème donné afin de le résoudre (déterminer une solution au problème).

Malik et Binford[Malik 83] proposent un modèle uniforme pour la représentation et le raisonnement sur des informations temporelles et spatiales. En utilisant la programmation linéaire, les informations temporelles sont traduites sous forme d'inéquations linéaires représentant les relations entre les bornes des événements. L'application de l'algorithme du SIMPLEXE sur l'ensemble des inéquations permettra alors de vérifier la consistance du problème (vérifier si une solution existe) et de calculer les bornes, supérieure et inférieure, des différents événements temporels. Étant donné qu'il y a un isomorphisme entre le raisonnement temporel et spatial, la même méthode peut être utilisée pour des problèmes géométriques (planification de chemins dans un espace muni d'obstacles, etc ...).

Barrouil[Barrouil 84] propose une méthode permettant le contrôle et le maintien de cohérence de contraintes temporelles (imposées notamment dans des problèmes de gestion des ressources) lors de la génération de plans. L'algorithme du SIMPLEXE est alors utilisé pour vérifier, à chaque ajout d'une contrainte lors de l'élaboration du plan, la cohérence de cette dernière avec les contraintes précédentes.
Le principe de l'algorithme du SIMPLEXE est de maximiser une fonction de coût linéaire en présence de contraintes linéaires. Plus précisément, cet algorithme consiste à chercher un sommet (dans un polyèdre convexe définissant les contraintes linéaires) en améliorant à chaque étape, la valeur de la fonction de coût. Lorsque les contraintes sont incompatibles, un sommet de départ ne peut être trouvé. Dans le cas d'un problème de satisfaction de contraintes, il n'y a pas de fonction à optimiser. Si un sommet de départ est trouvé, les contraintes sont compatibles, sinon elles ne le sont pas.
Le problème qui se pose lors de la génération de plan est que les contraintes temporelles sont formulées une à une au cours du processus de génération. Deux possibilités se présentent alors lors de l'application de l'algorithme du SIMPLEXE :

1. appliquer l'algorithme du SIMPLEXE à chaque fois qu'une contrainte est créée ou
2. appliquer l'algorithme du SIMPLEXE une fois qu'un plan possible complet est défini.

Ces deux possibilités conduisent à des calculs volumineux.
Une meilleure application de l'algorithme du SIMPLEXE dans ce cas consiste à l'implanter

d'une manière recursive. À chaque fois qu'une nouvelle contrainte est créée, on vérifie si elle est cohérente avec le contexte courant composé de l'ensemble des contraintes précédentes et un sommet S du polyèdre. Dans le cas contraire, cette nouvelle contrainte est considérée comme une fonction de coût relativement aux contraintes précédentes. En maximisant la fonction de coût, nous pouvons déduire si la nouvelle contrainte est cohérente ou non avec les contraintes précédentes. Soit, par exemple, $L(T)$ cette nouvelle contrainte où T est le point de coordonnées $(t_1, \ldots, t_n)$. Si $L(S) \geq 0$ alors la nouvelle contrainte est cohérente avec le contexte courant sinon ($L(S) < 0$) il faut chercher unn nouveau sommet S' tel que $L(S') \geq 0$ ce qui peut se faire en considérant $L(x)$ comme une fonction de coût. La maximisation de $L(x)$ dans le contexte courant conduit à un point S'. Si $L(S')$ est négatif on conclut que la nouvelle contrainte n'est pas compatible avec les précédentes, dans le cas contraire on déduit que la nouvelle contrainte est compatible avec les précédentes au point S' qui sera le point S de l'itération suivante.
Notons enfin, que le processus de génération de plans se base sur le langage PROLOG pour énumérer les différentes propositions (disjonction de requêtes exprimées sous forme de contraintes) et gérer le retour arrière en cas d'échec.

Les méthodes utilisant la programmation linéaire permettent de faire les constatations suivantes :

- ces méthodes ne sont pas décrites par des graphes de contraintes temporels, on peut donc manipuler des contraintes n-aires intraduisibles dans le formalisme des graphes, et
- l'algorithme du SIMPLEXE propose une solution, par contre les domaines des variables ne sont pas explicites.

b) Méthodes du chemin critique

Dans ces méthodes([Vere 83], [Bell 84] et [Dean 87]) les informations temporelles sont exprimées par des contraintes linéaires simples, de la forme :

$$A - B \geq d,$$

A et B représentent des instants temporels et d un paramètre positif.

Ces contraintes étant binaires, un graphe temporel permet donc de les représenter.
Les nœuds d'un tel graphe dénotent les instants (début ou fin) des différents événements. Chaque arc est alors étiqueté par la plus petite distance entre les instants correspondant aux nœuds auxquels il est relié. Ces distances (minimales entre chaque paire de nœuds) sont déduites en calculant la valeur du plus long chemin reliant la paire de nœuds. Le graphe est alors inconsistant si et seulement si il contient un circuit de longueur strictement positive (figure 2.1).
De cette manière, nous pouvons calculer les durées minimales et maximales des différents intervalles. Soit, par exemple, D et F représentant respectivement le début et la fin d'un intervalle I donné, la durée minimale de l'intervalle I correspond au plus long chemin de D à F. La durée maximale correspondra alors à l'opposé du plus court chemin de F à D. Nous ajoutons dans ce dernier cas des arcs à valeur négative dont la valeur absolue dénote la distance maximale entre les nœuds.

Nous pouvons de manière analogue calculer la date au plus tôt des différents instants en introduisant dans le graphe un nœud O pour origine du graphe). La date au plus tôt d'un instant X donné est égale à la valeur du plus grand chemin entre O et M. L'opposé du plus court chemin de M à O est égale à la date au plus tard de M.

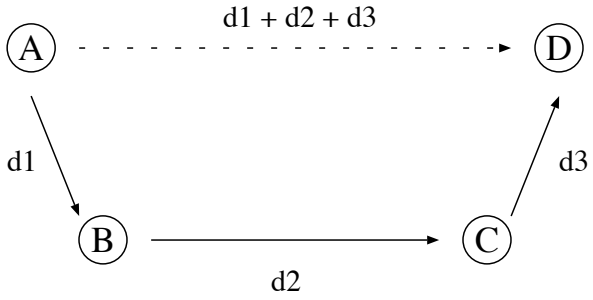


FIG. 2.1: *Chemin étiqueté*.

2.1.2 Approches manipulant explicitement les domaines des variables

Dans ce type d'approches[Vere 83][Pape 87], les variables représentent des intervalles de durée fixe dont le domaine de variation (appelé aussi fenêtre temporelle) est défini par des bornes numériques.

Des relations temporelles entre chaque paire de domaines sont alors utilisées pour vérifier la compatibilité des domaines. A l'inverse des problèmes d'étiquetage où les étiquettes ne vérifiant pas les relations temporelles sont éliminées, ce sont les domaines qui sont modifiés par compression de leurs bornes. Nous présentons dans le troisième chapitre, sous section 3.2.2, plus en détail cette dernière technique assurée par des méthodes de propagation de contraintes.

2.2 Représentations symboliques du temps

Les approches symboliques du temps sont fondées sur la logique. Trois types de logique ont permis la représentation symbolique du temps :

- La logique classique du premier ordre.
- La logique modale temporelle.
- La logique dite réifiée.

2.2.1 Le temps en logique classique

En logique classique, le temps est représenté par un terme argument d'un prédicat. Cette approche, mise en œuvre d'une façon restreinte dans le système WARPLAN[Warren 74] est extrêmement limitée. Elle ne donne aucun statut particulier au temps, et ne permet pas d'énoncer et d'exploiter dans le raisonnement des lois générales telle que la transitivité de la relation de

précédence ou l'antériorité des causes à effet.
Nous pouvons, dans ce même cadre, citer les travaux de Haugh[Haugh 87] dans lesquels il introduit la méthode TA des arguments temporels(TA pour Temporal Arguments). De la même manière que le système WARPLAN, la méthode TA consiste à introduire le temps comme un paramètre d'un prédicat du premier ordre. Ce paramètre temporel dénote le temps durant lequel ce prédicat doit être interprété. Par exemple, nous pouvons représenter l'information temporelle suivante : "Pierre et Marie se rencontrent le 12 Avril" par le prédicat $rencontre(Pierre, Marie, 12Avril)$. L'argument temporel n'ayant pas de statut particulier, la formule suivante : $rencontre(Pierre, Maire, Claude)$ est correcte.

2.2.2 Les logiques modales temporelles

En logique modale temporelle, le temps est pris en compte comme une modalité associée au domaine d'interprétation d'une formule. Sa manipulation se fait via des règles d'inférence de la logique ([Halpern 86]). Cette approche est souvent utilisée en informatique théorique, par exemple pour l'analyse et la spécification de programmes parallèles ou de protocoles.
Le langage utilisé est une extension du langage propositionnel ou des prédicats avec les opérateurs suivants :

Soit ϕ une formule propositionnelle.
$F\phi$: Il sera vrai une fois que ϕ.
$P\phi$: Il a été vrai au moins une fois que ϕ.
$G\phi$: ϕ sera toujours vrai.
$H\phi$: ϕ a toujours été vrai.

Le lien entre modalités se fait avec les opérateurs de possibilités et de nécessité : $\Diamond$ et $\Box$.

2.2.3 Les logiques réifiées

Les approches fondées sur la logique réifiée sont largement utilisées en intelligence artificielle. L'idée de la logique réifiée consiste à passer à un meta-langage, à partir d'un langage propositionnel ou de prédicat, où une formule du langage initial devient un terme dans le nouveau langage. De ce fait, dans ce nouveau langage nous pouvons raisonner sur des aspects particuliers de la validité des expressions du langage initial à travers l'utilisation du prédicat de vérité $TRUE$. Dans le cas du raisonnement temporel, le prédicat $TRUE$ prend comme argument une formule du langage propositionnel ou de premier ordre et une expression dénotant un objet temporel :

$$TRUE(formule\ atemporelle\ ,\ qualification\ temporelle)$$

Cette formule est représentée d'une manière plus simple par le couple $< \phi, t >$ où ϕ est une assertion logique atemporelle et t la qualification temporelle par rapport à laquelle ϕ est considérée.

La logique réifiée présente plusieurs avantages :

1. Elle accorde un statut spécial au temps.

2. Elle offre une puissance d'expression des différents aspects temporels. Nous pouvons par exemple ([Vila 94]) :

   – représenter l'axiome d'homogénéité [1] du prédicat $TRUE$, de la manière suivante :

   $$TRUE(p,T) \Leftrightarrow [\,\forall t\ \ in(t,T) \Rightarrow TRUE(p,t)\,]$$

   p étant une proposition, t et T des intervalles et in un prédicat exprimant le fait qu'un intervalle est contenu dans un autre intervalle, ou bien

   – exprimer des faits incompatibles : par exemple, nous pouvons exprimer la relation d'incompatibilité suivante :

   $$INCOMPATIBLE(dance(Jordi, Lolanda), dance(Jordi, Maria)).$$

   Cela permet d'établir l'axiome suivant qui stipule que deux faits incompatibles ne peuvent se chevaucher :

   $$\forall f_1, f_2\ \ HOLDS(f_1, time_1) \wedge HOLDS(f_2, time_2) \wedge INCOMPATIBLE(f_1, f_2)$$
   $$\Rightarrow \neg overlap(time_1, time_2)$$

   ou bien encore

   – exprimer la causalité : la propriété de causalité représente une connaissance très importante dans plusieurs domaines d'intelligence artificielle tel que le raisonnement qualitatif. Pour exprimer, par exemple, qu'un certain événement est la cause d'un fait donné, nous écrivons :

   $$CAUSES(start\_music, time_1, dance(Jordi, Lolanda), time_2).$$

   Nous pouvons de même établir la règle générale de causalité suivante :

   "Les effets ne peuvent pas précéder leur causes"

   par :

   $$\forall\, event,\ fact,\ time_1,\ time_2$$
   $$CAUSES(event, time_1,\ fact,\ time_2)\ \Rightarrow\ time_1 < time_2.$$

La logique réifiée a été largement utilisée dans les travaux sur le raisonnement temporel[McDermott 82][Allen 83][Dechter 91][Vilain 86][Kautz 91]. Cette approche permet d'établir un lien entre une assertion atemporelle et sa référence temporelle. Cette dernière consiste en un ensemble d'entités temporelles reliées entre elles par un ensemble de relations temporelles. Ces entités temporelles peuvent être soit des instants (points temporels) soit des intervalles de temps. Notons qu'on peut aussi avoir les deux types d'entités.
Les différents travaux de représentation du temps dans une logique réifiée se partagent donc entre deux grandes écoles : l'algèbre d'instants et l'algèbre d'intervalles.
Nous présentons dans ce qui suit, les différents formalismes de chaque école.

1. L'axiome d'homogénéité stipule qu'une proposition p est vraie sur un intervalle T donné si et seulement si elle est vraie sur tous ses sous intervalles

a) L'algèbre d'instants

Le travail le plus connu est celui de McDermott[McDermott 82]. Un instant étant défini comme une "*photographie instantanée de l'univers*", McDermott représente le temps par un ensemble infini et dense d'instants. L'idée étant de capturer l'aspect continu du temps.
Cette ensemble est partiellement ordonné par la relation de précédence vers le futur et linéaire sur le passé. Une assertion atemporelle est définie dans la logique réifiée par l'ensemble des instants t durant lesquels l'assertion est vrai :

$$TRUE(p,t)$$

Les seules relations possibles sur une ramification de la ligne temporelle sont les relations de précédence$(<,>,=)$. Un ensemble convexe et ordonné d'instants est un intervalle. Un intervalle non borné est une chronique(une chronique représente une histoire possible du monde). Un événement est un ensemble d'intervalles.
Une telle représentation offre l'avantage de pouvoir définir les dates et les durées des événements. Par contre il est impossible de considérer plus d'une seule ligne temporelle choisie parmi l'ensemble des ramifications vers le futur.

b) L'algèbre d'intervalles

Contrairement à l'approche précédente, l'élément principal est ici l'intervalle. Allen[Allen 84] considère que l'intervalle permet une meilleure simulation de la façon dont l'être humain perçoit le temps dans son raisonnement.
Pour Allen[Allen 84], les propriétés, les événements et les processus sont des entités temporelles fondamentales. Toutes sont associées à un intervalle. La différence de nature entre ces trois entités réside dans leur propriété de "*divisibilité*". Une propriété, notée $HOLD(p,t)$ (telle que "la voiture est bleue") est vraie sur un intervalle si et seulement si elle est vraie sur tous ses sous-intervalles. Par contre, un événement (tel que "la balle va de x_1 à x_2") ne peut être vrai sur deux intervalles dont l'un contient l'autre (le fait qu'un événement e se produit à un intervalle t est noté $OCCUR(e,t)$), dans ce cas l'événement e couvre l'intervalle t entièrement et ne peut survenir sur aucun de ses sous-intervalles. L'intervalle t est donc considéré comme étant le plus petit des intervalles dans lequel l'événement e se produit. Un processus se produisant sur un intervalle, noté $OCCURRING(p,t)$ (comme "Je mange") peut être vrai sur certains de ses sous intervalles. Contrairement aux événements, les processus peuvent se dérouler durant certains sous intervalles de leur intervalle, nous pouvons ainsi compter le nombre d'occurrences d'un événement mais pas celui d'un processus. Shoham[Shoham 86][Shoham 87][Shoham 88] a critiqué, quant à lui, la trichotomie : propriété, événement, processus de Allen. Il pense, en effet, que la distinction entre propriété et événement est insuffisante dans certaines applications et inutiles dans d'autres.

En utilisant ses trois entités fondamentales, Allen introduit aussi les notions de *causalité* et d'*action*.
La causalité stipule qu'un événement peut être la cause d'un autre événement à condition que ce dernier ne le précède pas. La causalité entre deux événements e_1 et e_2 survenant respectivement

sur les intervalles t_1 et t_2 est exprimée par :

$$ECAUSE(e_1, t_1, e_2, t_2)$$

D'un autre coté, une action est définie comme un événement ou un processus causé par un agent. Cette notion est exprimée de la manière suivante :

$$ACAUSE(a, e)$$

où a est l'agent et e l'événement ou le processus. À partir de la notion d'action, Allen introduit la notion de planification définie comme une séquence d'actions.

Allen[Allen 83] définit enfin un calcul sur les intervalles, basé sur treize primitives qui sont des relations temporelles binaires, mutuellement exclusives et qui permettent d'exprimer la position relative d'un intervalle par rapport à un autre. Ces relations sont illustrées dans le tableau ci-dessous :

Relation	Symbole	Symbole de la relation inverse	Exemple
X avant Y	P	$P^{\smile}$	XXX YYY
X égal Y	E	E	XXX YYY
X rencontre Y	M	$M^{\smile}$	XXXYYY
X recouvre Y	O	$O^{\smile}$	XXXX YYYY
X pendant y	D	$D^{\smile}$	XXX YYYYYY
X débute Y	S	$S^{\smile}$	XXX YYYYY
X termine Y	F	$F^{\smile}$	XXX YYYYY

Sur ces relations, Allen définit la règle de transitivité permettant de déduire la position relative de deux intervalles en fonction des relations les liant à un troisième intervalle.

Exemple :
Soit $I_1,\ I_2,\ I_3$ trois intervalles :
Si $I_1\ avant\ I_2$ et $I_2\ rencontre\ I_3$ **Alors** $I_1\ avant\ I_3$.

Allen définit alors un réseau de contraintes dans lequel les nœuds représentent les intervalles et les arcs chacun étiqueté d'une disjonction de relations de base de Allen représentent la relation temporelle qualitative entre les nœuds qui les concernent.
La vérification de la cohérence du réseau est assurée par un algorithme de propagation de contraintes qui utilise la table de transitivité(décrite ci-dessus) pour déduire toutes les relations possibles entre les intervalles et éliminer les relations qui rendraient le réseau incohérent. Dans

le chapitre suivant, nous décrirons plus en détail la propagation de contraintes sur un tel réseau.

La représentation de Allen permet de définir les différents positionnements possibles entre deux intervalles. Cette représentation est très recommandée soit dans des applications où l'information numérique n'est pas disponible soit dans les applications où cette dernière n'est pas importante. Il est cependant impossible de définir la date ou la durée d'une entité temporelle dans cette représentation. Koomen[Koomen 87] propose ensuite une extension de la représentation de Allen permettant d'intégrer le début et la fin des intervalles sous forme numérique.

c) Autres formalismes de la logique réifiée

Bien que les approches de Allen et de McDermott aient été largement utilisées en raisonnement temporel, aucune sémantique formelle de ces approches n'a été donnée. Shoham[Shoham87] soulève ce problème et propose une formalisation de la logique réifiée pour les assertions propositionelles et prédicatives. Il introduit, pour cela, un meta-prédicat $TRUE$ qui prend comme arguments, deux termes temporels et une proposition atemporelle (relation ayant pour arguments des termes atemporels).

Exemples :

$TRUE(10{:}00, 11{:}00, dance(Jordi, Lolanda))$
$\forall v_i,\ Dans(v_i, 10{:}00, 11{:}00) \Rightarrow TRUE(v_i, 11:00, origine(partenaire(Jordi), Italie))$

Vila[Vila 94] critique le formalisme de Shoham en dégageant ses faiblesses d'expressivité. En effet, en utilisant le prédicat $TRUE$, tous les prédicats et fonctions formant la partie atemporelle de ce prédicat sont interprétés sous les mêmes points temporels. La phrase suivante, par exemple, ne peut être exprimée par le prédicat $TRUE$:

"*Jordi danse aujourd'hui avec la fille qui portait hier une jupe rouge*"

Vila déduit alors que la logique de Shoham ne peut pas exprimer des connaissances temporelles générales tel que : "Les effets ne peuvent pas précéder leur causes", bien que Shoham l'ait utilisé comme argument pour le formalisme qu'il a proposée. Le formalisme de Shoham ayant beaucoup de similitude avec la logique modale (en effet, le prédicat $TRUE$, du point de vue sémantique, peut être considéré comme un opérateur modal qui prend deux expressions temporelles et génère un nœud opérateur modal), Vila se joint à Bacchus[Bacchus 91] et Reichgelt[Reichgelt 89] pour dire que ce formalisme ne peut pas être qualifié de logique réifiée. Reichgelt[Reichgelt 89] propose enfin la logique réifiée TRL (Temporal Reified Logic) utilisant comme langage objet, la logique modale temporelle classique avec l'opérateur $AT(t)$ et la définit comme un langage de premier ordre.

La représentation par intervalles a un pouvoir expressif plus riche que la représentation par instants. Elle possède néanmoins un inconvénient majeur : la vérification de cohérence d'un ensemble de relations entre intervalles est un problème NP-difficile([Vilain 86]). Nous devons par conséquent recourir à des algorithmes polynômiaux de propagation locale de contraintes qui

posent deux problèmes :

- un problème d'incomplétude : Ne permettant d'assurer qu'une cohérence locale, ces algorithmes peuvent accepter comme satisfiable un ensemble de relations qui ne le sont pas,
- et un problème de complexité : Avec une complexité $O(n^3)$, ces algorithmes sont coûteux pour des applications conséquentes.

En essayant de palier le premier problème, Vilain et Kautz[Vilain 86] proposent une sous classe de l'algèbre d'intervalles, appelée *algèbre d'intervalles restreinte*. Cette algèbre est équivalente à l'algèbre d'instants. Son avantage, valable aussi pour l'algèbre d'instants, est la possibilité de vérifier la cohérence globale avec un algorithme polynômial, mais cette approche repose sur une représentation purement symbolique.
En s'inspirant de cette approche, Ghallab[Ghallab 89] propose le modèle IxTeT(Treillis d'instants) basé sur un langage permettant la représentation d'instants, d'intervalles et de relations de l'algèbre d'intervalles restreinte. Toutes les relations sont traduites dans l'algèbre d'instants. Dans le modèle IxTeT, l'ensemble des contraintes et des événements temporels est représenté par un graphe dont les nœuds sont des instants et les arcs sont étiquetés par les deux relations : $\leq$ et $\neq$. Ces deux relations suffisent à exprimer les 8 relations suivantes permettant de situer deux instants quelconques :

- les 3 relations de base : $<, > et =$,
- et les 5 relations disjonctives : $(> \vee =), (< \vee =), (< \vee >).(< \vee > \vee =)$ et la relation vide $\circ$.

Dans cette approche, la vérification et le maintien de cohérence de relations temporelles sont assurés non pas par des techniques de propagation de contraintes mais par la recherche d'un chemin dans un treillis. Si une opération d'agrégation entre les nœuds appartenant à un circuit d'arcs du graphe étiquetés par la relation $<=$ est effectuée, le graphe se traduit par un treillis pour la relation $<=$.
La gestion, quant à elle, de relations temporelles utilise dans cette approche un algorithme complet et de faible complexité, une complexité en moyenne linéaire pour l'accès et l'ajout de nouvelles relations. La représentation des informations temporelles privilégie l'instant au plus bas niveau tout en permettant la représentation des relations restreintes entre intervalles au niveau d'un langage d'expression d'actions, d'effets, de tâches et d'événements.

2.3 Représentations mixtes

Les approches qualitatives du temps[Allen 83])[Vilain 86][McDermott 82] trouvent des difficultés a représenter et raisonner sur des données numériques. De leur coté, les approches numériques sont limitées lorsqu'il s'agit d'exprimer des informations qualitatives.
Ces dernières années, plusieurs travaux de recherche ont abordé le problème de représentation et de raisonnement sur le temps en utilisant des informations de nature différentes, qualitatives et numériques. Le facteur commun entre ces schémas est qu'ils sont fondés sur l'intervalle

considéré comme l'entité de base utilisée pour la propagation de contraintes.
Meiri[Meiri 91] propose une structure généralisant l'algèbre d'intervalles, l'algèbre de points et les réseaux métriques. Son modèle se compose d'un seul réseau supportant les deux types de contraintes, numériques et qualitatives. Dans ce modèle, les variables représentent soit des points temporels soit des intervalles. Les contraintes entre variables peuvent être soit numériques, les variables sont, dans ce cas, des points temporels, soit des relations symboliques binaires, les variables pouvant être dans ce cas des points temporels ou des intervalles.

Kautz et Ladkin[Kautz 91] proposent un modèle permettant d'intégrer des contraintes métriques et symboliques dans un système de raisonnement basé sur des contraintes. 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 treize primitives de Allen. Ces deux types de contraintes sont propagées dans 2 réseaux séparés, la gestion des deux types de contraintes étant assurée par des procédures de transfert entre les deux types d'informations.

Durant son travail de thèse au sein de l'équipe RFIA(CRIN&INRIA Lorraine) Hany Tolba[Tolba 91][Tolba 92] propose un modèle intégrant des informations qualitatives et quantitatives du temps. Deux réseaux de contraintes permettent alors de supporter les deux types d'informations. Pour vérifier et maintenir la cohérence des deux types d'informations, Tolba utilise une propagation de contraintes à deux niveaux, symbolique et numérique. Un échange d'information entre les ces deux types d'informations est assuré par un module appelé module de communication.

En gardant le principe d'utiliser deux réseaux différents pour les deux types de contraintes, nous rejoignons l'idée de Tolba et 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 que celui de Kautz et Ladkin, il permet, comme nous allons le montrer dans la section suivante, une homogénéité entre la représentation des deux types de contraintes (puisqu'on ne manipule que des intervalles) ce qui facilite le traitement de leur consistance et offre une puissance de calcul accrue aux algorithmes de consistance utilisés.

Chapitre 3

Satisfaction de contraintes

Le problème de satisfaction d'un ensemble de contraintes est un problème classique en informatique. Un grand nombre de problèmes, en informatique et spécialement en intelligence artificielle, peuvent être vus comme étant des cas particuliers de problèmes de satisfaction de contraintes[Nudel 90]. Ces problèmes vont de l'analyse de scènes en vision[Montanari 74][Mackworth 77][Davis 81] à l'analyse grammaticale en langage naturel en passant par la conception[Estman 72], l'ordonnancement[Rit 86], le raisonnement temporel[Allen 83][Allen 84][Dechter 91][Meiri 91] [Vilain 86][Kautz 91][Ladkin 92a] [Ghallab 89][Tolba 92][van Beek 96] et la biologie moléculaire[Gaspin 95].

Plusieurs travaux ont abordé le problème de satisfaction de contraintes notamment en recherche opérationnelle et en logique. Une approche plus récente, appelée CSP pour Constraint Satisfaction Problem, consiste à trouver un étiquetage consistant représentant un ensemble de valeurs de variables satisfaisant toutes les contraintes d'un problème donné.
Le problème d'existence d'un étiquetage consistant est NP-complet. La résolution de ce problème passe donc par des méthodes heuristiques, d'où le recours aux techniques de propagation locale de contraintes, dites de filtrage de données. Plusieurs chercheurs ont alors proposé des algorithmes de consistance locale permettant de supprimer certaines inconsistances et réduisant par conséquent l'espace de recherche de solutions ce qui facilite éventuellement la résolution ultérieure (déterminer une solution au problème de contraintes).
Ces algorithmes sont appelés algorithmes de satisfaction de contraintes (CSA pour Constraint satisfaction Algorithms).

Tout problème de satisfaction de contraintes peut être codé sous forme d'un graphe de contraintes. Un algorithme de consistance locale permet de propager les contraintes sur le graphe et d'assurer la cohérence, ou consistance locale de ce dernier. Montanari[Montanari 74] fût le premier à avoir spécifié formellement les techniques de propagation de contraintes utilisant les algorithmes CSA. La consistance locale du graphe est alors assurée par un algorithme appelé PC-1, PC pour path consistency ou consistance de chemins. Depuis, de nombreux travaux dans ce domaine ont vu le jour. Le succès de l'approche par propagation de contraintes est dû notamment au fait que de nombreux problèmes de contraintes se codent naturellement sous forme de réseaux de contraintes et sont résolus de manière efficace grâce aux techniques développées durant ces deux dernières décades.

3.1 Problème de satisfaction de contraintes

Soit N un ensemble de variables $\{X_1 \dots X_n\}$ tel que chaque variable x_i est définie sur un domaine discret $\{a_{i1} \dots a_{in}\}$ et R un ensemble de contraintes (nous nous limitons dans ce qui suit aux relations binaires[2]) sur un sous ensemble de ces variables. Un problème de satisfaction de contraintes (appelé CSP pour Constraint Satisfaction Problem) consiste à déterminer tous les ensembles de valeurs $\{a_{1j1} \dots a_{1jn}\}$ pour $\{X_1 \dots X_n\}$ 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 la recherche de solutions, des algorithmes de satisfaction locale de contraintes sont alors utilisés soit comme un filtre, ou étape de pre-traitement, avant l'étape de recherche de solutions, soit durant le processus de recherche. Ces algorithmes, appelés aussi algorithmes de consistance locale (ou aussi algorithmes de k-consistance), ont pour 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 suppriment toutes les inconsistances qui ne vérifient pas les contraintes entre les sous-ensembles de k variables appartenant à N. On parle alors de consistance d'arcs quand $k = 2$ et de chemins quand $k = 3$. 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 ensuite nécessaire pour résoudre le problème.

Nous présentons, dans ce qui suit, les différentes techniques de propagation de contraintes ainsi que les différentes stratégies permettant de diriger la recherche afin d'améliorer les performances du retour-arrière.

3.2 Résolution de problèmes de satisfaction de contraintes

La méthode naïve

La méthode naïve pour résoudre un problème de satisfaction de contraintes donné utilise le paradigme *générer-et-tester*. Ceci consiste à générer une composition de valeurs de variables du problème et à tester si cette composition vérifie toutes les contraintes du problème ou non jusqu'à obtention de la solution. La première composition de valeurs satisfaisant les contraintes est une solution au problème. Le nombre de combinaisons considéré dans ce cas est le cardinal du produit cartésien des domaines de toutes les variables du problème.

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

La méthode du retour arrière

Cette méthode, plus efficace que la méthode précédente, utilise la technique du retour arrière. A l'inverse de la méthode précédente, les variables sont ici instanciées séquentiellement. À chaque fois qu'une variable est instanciée, on teste si le nouveau sous ensemble de variables instanciées vérifie l'ensemble des contraintes qui les relient. Si une inconsistance est détectée, un retour arrière est effectué sur la dernière variable instanciée, une autre valeur du domaine de cette variable est alors choisie. Le retour arrière permet donc d'éliminer un sous espace du produit cartésien des valeurs de toutes les variables à chaque fois qu'une instanciation partielle viole les contraintes du problème.

Bien que cette méthode soit meilleure que la méthode précédente en ce sens où elle effectue moins de tests pour déterminer une solution au problème, les performances du retour arrière sont affectées par le fait que le processus de recherche dans l'espace de solutions potentielles échoue souvent pour les mêmes raisons. Ce phénomène est appelé *thrashing*[Gashing 79].
L'une des causes du *thrashing* est l'inconsistance de nœuds[Mackworth 77] :

> *Si une certaine valeur a appartenant au domaine D_i de la variable X_i ne satisfait pas la contrainte linéaire sur cette dernière, chaque instanciation de la variable X_i par a provoque un échec de l'algorithme de recherche.*

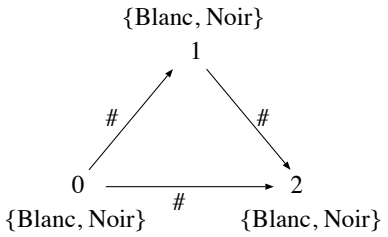
Une seconde cause est l'inconsistance d'arcs [Mackworth 77] :

> *Supposons que les variables d'un problème donné soient instanciées dans l'ordre suivant $X_1,\ X_2,\ ...X_i, \ldots X_j, \ldots X_n$. Supposons aussi que l'instanciation de la variable X_i par la valeur a viole la contrainte binaire entre X_i et X_j(quelque soit la valeur de X_j). Après avoir instancié la variable X_i par a, le processus de recherche échoue après chaque instanciation de la variable X_j car aucune valeur de X_j n'est compatible avec la valeur de a de X_i. Cet échec est répété pour toutes les combinaisons des valeurs de variables v_k ($i < k < j$).*

Comme l'inconsistance de nœuds ou d'arcs, l'inconsistance de chemins (figure 3.1) peut également être une cause du *thrashing*.
Soit le graphe présenté en figure 3.1 illustrant un exemple de problème de coloriage de graphe[3], à chaque nœud du graphe est associé un domaine de couleurs admissibles à priori. Bien que le graphe vérifie la consistance d'arcs, aucune valeur n'est possible sur le nœud 2 si l'on fixe 2 valeurs consistantes sur les nœuds 0 et 1.

Le problème de *thrashing* dû à l'inconsistance de nœuds peut être évité en éliminant, avant l'étape de recherche, toute valeur du domaine D_i de chaque variable X_i qui ne satisfait pas la relation unaire sur cette variable.

3. Le problème de coloriage de graphe consiste à étiqueter chaque nœud du graphe par une couleur de son domaine de manière à ce que deux nœuds voisins n'aient pas la même couleur.



Bien que la consistance d'arcs est vérifiée, ce graphe ne vérifie pas la consistance de chemins (3-Consistance).

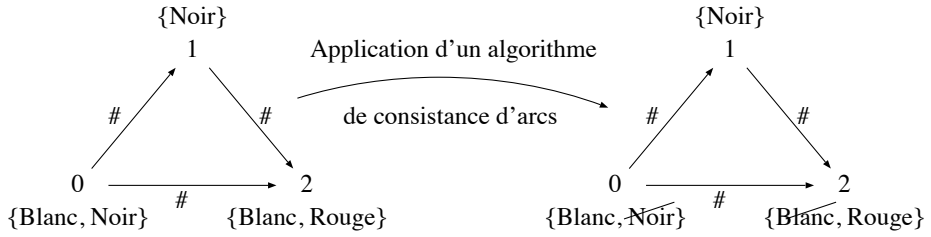
FIG. 3.1: *Consistance d'arcs et de chemins sur un problème de coloriage de graphe.*

Le *thrashing* du à l'inconsistance d'arcs (ou de chemins) peut être résolu en réduisant le graphe de contraintes, avant l'étape de recherche de solutions, en un graphe correspondant respectant la consistance d'arcs (ou de chemins). Ceci peut être obtenu en utilisant les techniques de consistance d'arcs ou de chemins que nous allons présenter dans les sous sections suivantes.

3.2.1 Consistance d'arcs dans un réseau de contraintes

a) La consistance d'arcs

Soit le réseau de contraintes $G = (N, R)$ (défini en sous section 3.1). Ce réseau est arc-consistant si tous ses arcs sont arc consistants. Un arc (i, j) est arc consistant si et seulement si pour chaque valeur a appartenant au domaine D_i de la variable X_i , il existe une valeur b appartenant au domaine D_j de la variable X_j tel que la contrainte binaire entre la paire de variables (X_i, X_j) soit satisfaite. On dit alors que a appartenant à D_i est supporté par b 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 3.2).



En comparant les noeuds 0 et 1 on déduit que 0 ne peut pas être Noir, puis en comparant 0 et 2 on déduit que 2 ne peut pas être Blanc.

FIG. 3.2: *Application d'un algorithme de consistance d'arcs sur un graphe de contraintes.*

b) Les algorithmes de révision d'arcs

Les premiers algorithmes de consistance d'arcs, appelés "*algorithmes de révision d'arcs*", se basent sur la fonction $REVISE$, ci-après, 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).

Fonction $REVISE(i, j)$
Début
$REVISE \leftarrow faux$
Pour `chaque valeur` $a \in D_i$ **Faire**
Si $\neg compatible(a, b)$
`pour toute valeur` $b \in D_j$ **Alors**
`enlever` a `de` D_i
$REVISE \leftarrow vrai$
Fin-Si
Fin-Pour
Fin

Pour rendre chaque arc du graphe consistant, il ne suffit pas d'appliquer la fonction $REVISE$ une seule fois sur chaque arc. En effet, lorsque la fonction $REVISE$ réduit le domaine d'une certaine variable X_j, cette fonction doit être appliquée une autre fois sur l'arc (X_i, X_j) (Bien que ce dernier ait été déjà rendu consistant) car certaines valeurs de X_j supports de valeurs de X_i ont pu être supprimées.
L'algorithme AC-1 ci-après, proposé par Mackworth[Mackworth 77], permet d'assurer la consistance d'arcs sur un réseau de contraintes.

Algorithme AC-1

Début
`1.Soit le graphe` $G = (U, E)$
`2.` `(`U`: ensemble d'arcs,` E`: ensemble de noeuds)`
`3.` $Q \leftarrow \{(i, j) \mid (i, j) \in U\}$
`4.` `(liste contenant initialement tous les arcs de` G`)`
`5.` **Répéter**
`6.` $change \leftarrow faux$
`7.` **Pour** `chaque` $(i, j) \in Q$ **Faire**
`8.` $change \leftarrow (REVISE(i, j) \vee change)$
`9.` **Fin-Pour**
`10.` **Jusqu-a** $\neg change$
Fin

Dans cet algorithme, à chaque fois que la fonction $REVISE$ élague des valeurs du domaine d'une variable donnée, elle est appliquée de nouveau sur tous les arcs du graphe. Ce qui constitue un inconvénient de l'algorithme étant donné que seuls certains arcs sont concernés par cet élagage. La fonction $REVISE$ ne doit donc s'appliquer que sur ces derniers arcs, et non sur tous les arcs du graphe. D'ou le principe de l'algorithme AC-3[Mackworth 77] suivant

qui utilise une structure de données pour stocker, à chaque fois que le domaine de valeurs d'un nœud donné a été modifié, tous les arcs ayant pour origine ce dernier nœud.
La structure de données est initialisée à tous les arcs du graphe de contraintes.

Algorithme AC-3

Début
1.Soit le graphe $G = (U, E)$
2. (U: ensemble d'arcs, E: ensemble de noeuds)
3. $Q \leftarrow \{(i,j) \mid (i,j) \in U\}$
4. (liste contenant initialement tous les arcs de G)
5. **Tant-Que** $Q \neq Nil$ **Faire**
6. $Q \leftarrow Q - \{(i,j)\}$
7. **Si** $REVISE(i,j)$ **Alors**
8. $Q \leftarrow Q \sqcup \{(k,i) \mid (k,i) \in U \wedge k \neq j\}$
9. **Fin-Si**
10. **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].

Notons que Mackworth[Mackworth 77] a proposé l'algorithme AC-2 qui représente un cas particulier de l'algorithme AC-3 dans lequel les nœuds sont traités dans un ordre déterminé.
L'idée derrière l'algorithme AC-2 est que chaque nœud sera traité une seule fois durant l'application de la consistance d'arcs. Les nœuds sont alors explorés dans un ordre déterminé. A chaque traitement d'un nœud i donné, tous les arcs (k,m) tel que $k,m \leq i$ sont rendus consistants. Pour ce faire, deux listes Q et Q' permettent de gérer les différents arcs à traiter lors de l'exploration du nœud i de la manière suivante : tous les arcs ayant pour origine le nœud i et pour extrémité un nœud déjà exploré sont mis dans la liste Q. De même, tous les nœuds ayant pour extrémité le nœud i et pour origine un nœud déjà exploré sont mis dans la liste Q'. Les arcs contenus dans la liste Q sont traités un à un et à chaque fois qu'un domaine de valeurs d'un nœud donné est modifié, tous les arcs ayant pour origine ce dernier nœud sont mis dans la liste Q' pour être traités par la suite.

L'algorithme AC-2 pose néanmoins, par rapport à AC-3, l'inconvénient suivant : Si le graphe de contraintes possède des cycles de plus de deux arcs, les listes Q et Q' peuvent contenir le même arc en même temps (ce dernier arc sera alors révisé inutilement).

L'algorithme AC-2 est décrit comme suit :

Algorithme AC-2

Début
1. Soit le graphe $G = (U, E)$
2. (U: ensemble d'arcs, E: ensemble de noeuds)
3. **Pour** $i \leftarrow 1$ jusqu'a n **Faire**
4. $Q \leftarrow \{(i,j) \mid (i,j) \in U, j < i\}$
5. $Q' \leftarrow \{(j,i) \mid (j,i) \in U, j < i\}$
6. **Tant-Que** $Q' \neq Nil$ **Faire**
7. **Tant-Que** $Q \neq Nil$ **Faire**
8. $Q \leftarrow Q - \{(k,m)\}$
9. **Si** $REVISE(k,m)$ **Alors**
10. $Q' \leftarrow Q' \sqcup \{(p,k) \mid (p,k) \in U \wedge p \leq i \wedge p \neq m\}$
11. **Fin-Si**
12. **Fin-Tant-Que**
13. $Q \leftarrow Q$
14. $Q' \leftarrow Nil$
15. **Fin-Tant-Que**
16. **Fin-Pour**

Fin

c) Algorithmes de maintien de support

Mohr et Henderson proposent un algorithme ayant une complexité $O(ed^2)$[Mohr 86] dans le pire des cas. Il s'agit de l'algorithme AC-4. Cet algorithme est le premier d'une nouvelle catégorie d'algorithmes appelés "*algorithmes de maintien de support*".
AC-4, présenté ci-après, se déroule en deux étapes. La première consiste à initialiser les différentes structures de données contenant les informations relatives à chaque valeur de variables. Chaque valeur a d'une variable i donnée (étiquette (i,a)) possède :

- un compteur, par variable j du réseau avec $j \neq i$, contenant le nombre de valeurs supports, de l'étiquette (i,a), appartenant à j (nombre d'étiquettes (j,b) compatibles avec (i,a)), et

- une liste $S_{a,i}$ contenant toutes les étiquettes de nœuds (j,b) ayant (i,a) comme support.

Une structure $SUPP$ va contenir, après l'étape d'initialisation, toutes les étiquettes de nœuds à supprimer (i.e les étiquettes dont un de leurs compteurs vaut zéro).

La seconde étape, appelée étape d'élagage, permet de supprimer toutes les étiquettes contenues dans $SUPP$ et de mettre à jour les structures compteurs et les listes supports des valeurs de toutes les variables. À chaque fois que le compteur d'une valeur devient nul, cette dernière est mise dans $SUPP$ pour être traitée par la suite. L'algorithme continue jusqu'à ce que toutes les étiquettes contenues dans $SUPP$ soient traitées.

Bien que AC-4 soit meilleur que AC-3 dans le pire des cas, Wallace[Wallace 93] montre que AC-3 présente un meilleur comportement que AC-4 dans le cas général. En effet, l'une des particularités de l'algorithme AC-3 est qu'il est dépendant de l'ordre dans lequel les nœuds sont traités. Un choix judicieux de cet ordre s'appuyant sur des heuristiques données peut réduire considérablement le nombre de tests effectués par cet algorithme et améliorer par conséquent ses performances[Wallace 92].
Ces heuristiques ont pour but de minimiser le nombre de tests des mêmes domaines de valeurs en éliminant les valeurs inconsistantes le plus tôt possible. Les paires de nœuds ayant une contrainte restrictive sont donc révisées en premier.
Bien que la première étape de AC-4 soit aussi dépendante de l'ordre de traitement des nœuds (ce qui n'est pas le cas de la deuxième étape), Wallace montre, en se basant sur des tests expérimentaux, que si nous utilisons les mêmes heuristiques pour les deux algorithmes, AC-3 présente de meilleurs résultats que AC-4 dans le cas général. Ceci se justifie par le fait que l'algorithme AC-3, appliqué à chaque paire de nœuds (i, j), consiste à chercher un support pour chaque valeur de i en la testant avec les valeurs de j jusqu'à trouver un support, le test ne se fait généralement pas pour toutes les valeurs de j. Par contre, AC-4 assure la consistance d'arcs sur la paire de nœuds (i, j) en déterminant le nombre de supports dans j pour chaque valeur de la variable i. Toutes les valeurs de j doivent donc être testées d'où l'inconvénient de AC-4 par rapport à AC-3 dans le cas général.

Dans la suite de ce rapport, nous dégagerons d'autres avantages de AC-3 par rapport à AC-4. En effet, nous avons utilisé ces deux algorithmes dans notre outil de raisonnement afin de résoudre des problèmes de satisfaction de contraintes temporelles. Afin de faire face aux problèmes de complexité auquel nous nous attaquons, nous avons ajouté des heuristiques aux deux algorithmes s'appuyant sur des propriétés des contraintes temporelles que nous manipulons. Nous avons alors constaté que ces heuristiques améliorent beaucoup plus les performances de AC-3 que ceux de AC-4. Des tests effectués de comparaison des deux algorithmes, présentés en troisième partie, confirmeront ces constatations.

Algorithme AC-4

Notations :

N : Ensemble de nœuds.
U : Ensemble d'arcs.
E_i : Ensemble des étiquettes du nœud i.
(i, a) : Étiquette a du nœud i.
R_2 : Relation binaire.
$R_2(i, a, j, b) \Rightarrow (i, a)$ compatible avec (j, b).
$C[a, i, j]$: Compteur des étiquettes du nœud j
compatible avec (i, a)
$S_{a,i} = \{(j, b) \mid (i, a)$ compatible avec $(j, b)\}$
/* Ensemble des étiquettes de nœuds ayant (i, a) comme support */
$SUPP$: Liste contenant toutes les étiquettes de nœuds à supprimer.

Initialisation :

Début
  $SUPP \leftarrow \emptyset$
  Pour `chaque` $(i,a) \in N\mathtt{x}E_i$ **Faire**
    $S_{a,i} \leftarrow \emptyset$
  Pour `chaque` $(i,j) \in U$ **Faire**
    Pour `chaque` $a \in E_i$ **Faire**
      $Total \leftarrow 0$
      Pour `chaque` $b \in E_j$ **Faire**
        Si $R_2(i,a,j,b)$ **Alors**
          $Total \leftarrow Total + 1$
          $S_{b,j} \leftarrow S_{b,j} \cup \{(i,a)\}$
        Fin-Si
      Si $Total = 0$ **Alors**
        $E_i \leftarrow E_i - \{a\}$
        $SUPP \leftarrow SUPP \cup \{(i,a)\}$
      Sinon $C[a,i,j] \leftarrow Total$
      Fin-Si
Fin

Élagage :

Début
  Tant-Que $SUPP \neq \emptyset$
    `Choisir` (k,c) `dans` $SUPP$
    $SUPP \leftarrow SUPP - \{(k,c)\}$
    Pour `chaque` $(j,b) \in S_{c,k}$ **Faire**
      $C[b,j,k] \leftarrow C[b,j,k] - 1$
      Si $C[b,j,k] = 0$ **Alors**
        $E_j \leftarrow E_j - \{b\}$
        $SUPP \leftarrow \cup \{(j,b)\}$
      Fin-Si
Fin

En tirant profit des propriétés de certains types de contraintes, Deville et van Hentenryck [Deville 91][van Hentenryck 92] proposent l'algorithme AC-5 ayant une complexité en temps de $O(ed)$ pour les contraintes fonctionnelles, anti-fonctionnelles et monotones, mais qui ce réduit à AC-3 ou AC-4 dans le cas général.

Afin d'améliorer la complexité moyenne en temps de AC-4 tout en gardant la même complexité au pire cas, Bessière propose l'algorithme AC-6[Bessière 94]. AC-6 pallie le problème de complexité en espace de AC-4[4] et effectue moins de tests que ce dernier

4. La complexité en espace est de $O(ed)$ pour AC-6 alors qu'elle est de $O(ed^2)$ pour AC-4.

dans le cas général pour assurer la consistance d'arcs. Plus récemment, Bessière et ses co-auteurs[Bessière 95] proposent une amélioration de AC-6, il s'agit de l'algorithme AC-7. AC-7 (présenté ci-dessous) 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"[5], AC-7 élimine, en effet, certains tests de consistance effectués par les autres algorithmes (AC-3, AC-4 et AC-6).

Algorithme AC-7

Notations :

N : Ensemble de nœuds.
U : Ensemble d'arcs.
E_i : Ensemble des étiquettes du nœud i.
/* Les étiquettes de chaque nœud sont triées dans un ordre donné */
(i,a) : Étiquette a du nœud i.
R_{ij} : Relation binaire entre les nœuds i et j.
$R_{ij}(a,b) \Rightarrow (i,a)$ compatible avec (j,b).
$S[(i,j),a]$: Ensemble des étiquettes du nœud j supportées par (i,a)
$SeekSupportStream$: Liste de toutes les valeurs pour lesquels on cherchera un support
$DeletionStream$: Liste contenant toutes les étiquettes à supprimer
M : Matrice Booléenne, $N \text{x} Max(Card(E_i))$
$M[i,a] =$ Vrai $\Rightarrow a \in E_i$ (Faux sinon)
$inf\_support$: $inf\_support[(i,j),a]$ contient le plus petit support de a dans j

Fonction $SeekNextSupport((i,j),a,b)$
Début
1 **Si** $b > last(E_j)$ **Alors** retourner Faux
2 **Tant-Que** $\neg M[j,b]$ **Faire** $b \leftarrow b+1$
3 **Si** $inf\_support[(j,i),b] \leq a$ **Alors**
4 **Si** $R_{ij}(a,b)$ **Alors** retourner Vrai
5 **Si** $b < last(E_j)$ **Alors**
$b \leftarrow next(b,E_j)$
Aller à la ligne 3
6 retourner Faux
Fin

5. 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.

Fonction $SeekInferableSupport((i,j),a,b)$
/* retourne vrai si une valeur b est trouvé */
Début
1 **Si** $S[(i,j),a] = Nil$ **Alors** retourner Faux
2 $b \leftarrow$ un élément de $S[(i,j),a]$
3 **Si** $M[j,b]$ **Alors** retourner vrai
4 Éffacer b de $S[(i,j),a]$
5 Retourner à la ligne 1
Fin

Début
$DeletionStream \leftarrow Nil$
$SeekSupportStream \leftarrow Nil$
Pour chaque $(i,a) \in N \mathbf{x} E_i$ **Faire** $M[i,a] \leftarrow true$
Pour chaque $(i,j) | R_{ij}$ existe **Faire**
Pour chaque $a \in E_i$ **Faire**
$S[(i,j),a] \leftarrow Nil$
$inf\_support[(i,j),a] \leftarrow 1$ /* 1ère valeur de E_j */
inserer $[(i,a),j]$ in $SeekSupportStream$
Répéter Jusqu'a done
Si $DeletionStream \neq Nil$ **Alors**
Retirer (j,b) de $DeletionStream$
Pour chaque $i | R_{ij}$ existe **Faire**
Pour chaque $a \in S[(j,i),b]$ **Faire**
Supprimer a de $S[(j,i),b]$
inserer $[(i,a),j]$ dans $SeekSupportStream$
Sinon Si $SeekSupportStream \neq Nil$ **Alors**
Retirer $[(i,a),j]$ de $SeekSupportStream$
Si $M[i,a]$ **Alors**
Si $SeekInferableSupport((i,j),a,c)$ **Alors**
mettre a dans $S[(j,i),c]$
Sinon $c \leftarrow inf\_support[(i,j),a]$
Si $SeekNextSupport((i,j),a,c)$ **Alors**
mettre a dans $S[(j,i),c]$
$inf\_support[(i,j),a] \leftarrow c$
Sinon
Supprimer a de E_i, $M[i,a] \leftarrow Faux$
Mettre (i,a) dans $DeletionStream$
Sinon done
Fin

Citons enfin les travaux de Lhomme[Lhomme 93] qui propose des techniques de consistance faible appelée "B-consistance d'arcs" pour des contraintes numériques, Lhomme travaillant sur des intervalles numériques plutôt que sur des valeurs discrètes. La B-consistance d'arcs peut être vue comme une forme de consistance d'arcs restreinte aux seules bornes des intervalles.

3.2.2 Consistance de chemins dans un réseau de contraintes

Un réseau de contraintes vérifie la consistance de chemins si et seulement si tous ses chemins sont chemin-consistants. Un chemin $X_0 \ldots, X_m$ est chemin consistant si et seulement si pour chaque couple de valeurs $(a, b) \in X_0 \text{x} X_m$ vérifiant la contrainte entre X_0 et X_m il existe une valeur pour chaque variable X_k $(1 \leq k \leq m-1)$ telle que chaque couple de valeurs $(c, d) \in X_i \text{x} X_{i+1}$ $(0 \leq i \leq m-1)$ vérifie la contrainte entre X_i et X_{i+1}. Une manière de vérifier la consistance de chemins sur un réseau de contraintes consiste à assurer la 3-consistance (consistance de chemins de longueur 2) sur ce dernier. La 3-consistance implique, en effet, la consistance de chemins [Montanari 74][Tsang 94].
Un réseau de contraintes vérifie la 3-consistance si et seulement si la règle suivante est respectée :

$$\forall i, j, k \in [1, n],\ \forall (v_i, v_j) \in R_{ij}, \exists v_k \in D_k \mid (v_i, v_k) \in R_{ik} \wedge (v_k, v_j) \in R_{kj}$$

Pour rendre un graphe 3-consistent, il suffit d'enlever toutes les valeurs v_i appartenant au domaine D_i (ou toutes les valeurs v_j appartenant à D_j) qui n'appartiennent à aucun couple (v_i, v_j) vérifiant la règle citée ci-dessus. Cela revient à supprimer tous les couples (v_i, v_j), de la relation R_{ij} qui ne vérifient pas la règle de 3-consistance citée précédemment.
Cette opération peut être vue comme le calcul de la relation entre X_i et X_j induite par R_{ik} et R_{kj}. D'où le principe de l'algorithme de fermeture transitive proposé par Montanari[Montanari 74] sous le nom de PC-1. L'algorithme PC-1 (voir ci-après) est appliqué sur chaque triplet de nœuds jusqu'à ce qu'il n'y ait plus de changement dans les relations.

Algorithme PC-1

Début
  $Y^n \leftarrow R$
  Répéter
    $Y^0 \leftarrow Y^n$
    Pour $k \leftarrow 1$ `jusqu'a` n **faire**
      Pour $i \leftarrow 1$ `jusqu'a` n **faire**
        Pour $j \leftarrow 1$ `jusqu'a` n **faire**
          $Y_{ij}^k \leftarrow Y_{ij}^{k-1}\ \&\ Y_{ik}^{k-1}\ .\ Y_{kj}^{k-1}$
  Jusqu'a $Y^n = Y^0$
  $Y \leftarrow Y^n$
Fin

R est la matrice représentant le graphe de contraintes. Chaque relation R_{ij} de R est définie par une matrice booléenne ayant en lignes, les valeurs de la variable i et en colonnes les valeurs de la variable j. $R_{ij}(a, b) = 1$ si le couple de valeurs (a, b) satisfait la relation R_{ij}, 0 sinon.

L'intersection "&" et la composition "." sont alors définies de la manière suivante.

Intersection

Soit deux relations R'_{ij} et $R"_{ij}$ entre deux variables i et j, l'intersection $R_{ij} = R'_{ij} \& R"_{ij}$ consiste à calculer toutes les entrées de la matrice booléenne correspondant à R_{ij} comme suit :
$\forall r, s \; R_{ij,rs} = R'_{ij,rs} \wedge R"ij, rs$
où $\wedge$ est l'operateur de multiplication binaire ("et" logique).

Composition

La composition $R_{ij} = R_{ik} \; . \; R_{kj}$ est calculée comme suit :
$\forall \; r, s \; R_{ij,rs} = \bigvee_{t=1}^{Card(D_j)} (R_{ij,rt} \wedge R_{jk,ts})$
où $\vee$ et $\wedge$ sont respectivement l'addition et la multiplication binaire et D_j le domaine de valeurs de la variable j.

Une version améliorée de cet algorithme est proposée par Mackworth[Mackworth 77] sous le nom de PC-2 (voir ci-après). Cet algorithme 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.

Algorithme PC-2

Début
1. $Q \leftarrow \{(i,k,j) \mid (i \leq j), \neq (i = k = j)\}$
2. `(liste contenant initialement tous les chemins à reviser)`
3. **Tant-Que** $Q \neq Nil$ **Faire**
4. $Q \leftarrow Q - \{(i,k,j)\}$
5. **Si** $REVISE(i,k,j)$ **Alors**
6. $Q \leftarrow Q \sqcup RELATED\_PATHS(i,k,j)$
7. **Fin-Si**
8. **Fin-Tant-Que**

Fin

Procedure REVISE(i, k, j)

Début
$Z \leftarrow Y_{ij} \;\; \& \;\; Y_{ik} \;\; . \;\; Y_{kk} \;\; . \;\; Y_{kj}$
Si $Z = Y_{ij}$ **Alors** `retourner FAUX`
Sinon $Y_{ij} \leftarrow Z$`; Retourner VRAI`
Fin

Procédure RELATED_PATHS(i, k, j)

Début
Si $i < j$ **Alors** `retourner`
$\{(i,j,m)|(i \leq m \leq n),(m \neq j)\} \sqcup \{(m,i,j)|(1 \leq m \leq j),(m \neq i)\}$
$\sqcup\{(j,i,m)|(j \leq m \leq n)\} \sqcup \{(m,j,i)|(1 \leq m \leq i)\}$
Sinon `Retourner`
$\{(p,i,m)|(1 \leq p \leq m),(1 \leq m \leq n), \neq (p = i = m), \neq (p = m = k)\}$
Fin

Dans la prochaine section, nous présenterons les versions des algorithmes PC-1 et PC-2 appliquées aux relations temporelles.
Mohr et Henderson [Mohr 86] appliquent le principe de l'algorithme AC-4 à la consistance de chemins, et proposent l'algorithme PC-3. En déterminant un contre exemple mettant en cause la correction de l'algorithme PC-3, Han et Lee[Han 88] présentent une version corrigée de ce dernier, il s'agit de l'algorithme PC-4. Ce dernier algorithme ne présente, par contre, pas de différence de principe avec PC-3.

3.2.3 Stratégies de recherche de solutions

Comme nous l'avons précisé auparavant, un algorithme de consistance d'arcs ou de chemins ne résout pas un problème de contraintes mais réduit son espace de recherche. Ceci améliore l'étape effective de recherche de solutions avec retour arrière généralement nécessaire pour déterminer une solution au problème.
En plus de leur utilisation dans l'étape de pre-traitement (filtrage) dans de la résolution d'un problème de contraintes, les algorithmes de consistance d'arcs (ou de chemins) peuvent être aussi utilisés dans l'étape de recherche avec retour arrière ce qui minimise le nombre de retour arrière et augmente les performances de l'algorithme de recherche.
Nous allons voir, à travers les stratégies permettant de diriger cette recherche, comment sont utilisés ces algorithmes de consistance partielle ou locale. Les algorithmes utilisés dans les stratégies que nous allons présenter sont des algorithmes de consistance d'arcs (en effet, ces derniers sont beaucoup moins coûteux que les algorithmes de consistance de chemins).

Les techniques de recherche que nous présentons se basent sur les idées suivantes :

- anticiper le futur afin que l'étiquetage présent soit cohérent,
- regarder le futur afin de ne pas être obligé de revenir sur le passé,
- se souvenir de ce qui a été fait dans le passé pour ne pas tomber dans les mêmes erreurs, et
- commencer là ou il n'y a plus de chance d'échouer afin d'atteindre la solution le plus tôt possible.

Nous nous basons sur les travaux de Nudel[Nudel 88] pour présenter et comparer ces techniques que nous utiliserons par la suite dans notre système de propagation de contraintes.

Nudel[Nudel 88] compare les algorithmes suivants :

- Générer et tester (GT pour *generate and test*),
- Algorithme de retour arrière(BT pour backtracking),
- Forward-Checking(FC),
- Partial Look-ahead(PL),
- Full Look-ahead(FL), et
- Really full Look-ahead(RFL).

Comme nous allons le voir dans ce qui suit, ces algorithmes diffèrent dans leur "degré d'utilisation" de l'algorithme de consistance d'arcs.

Algorithme générer et tester(GT)

Il s'agit de l'algorithme naïf de recherche de solutions qui génère, jusqu'à obtention d'une solution, une combinaison de valeurs (appartenant au produit cartésien des domaines de variables) et teste si cette combinaison respecte ou non toutes les contraintes du problème. L'algorithme de consistance d'arcs n'est pas utilisé dans ce cas.

Algorithme de retour arrière(BT)

Dans cet algorithme, la règle de consistance d'arcs est appliquée entre le nœud courant(nœud qui vient d'être instancié) et les nœuds déjà instanciés (dans ce qui suit, nous appellerons ces nœuds les nœuds passés et les nœuds non instanciés les nœuds futurs). Ceci a l'avantage de réduire le domaine de valeurs du nœud courant ce qui évite les retour-arrières dus à des valeurs de ce nœud, inconsistantes avec les valeurs des nœuds passés.

Algorithmes Look-ahead

Les algorithmes que nous allons citer sont des variantes d'une technique plus générale appelée *Look-ahead*. Cette technique se base sur les deux conditions suivantes qui seront appliquées lors de la recherche de solutions[Haralick 80] :

1. chaque nœud futur doit avoir au moins une valeur compatible avec le nœud courant et les nœuds passés, et
2. chaque nœud futur doit avoir au moins une valeur compatible avec celles de tous les autres nœuds futurs.

1- Algorithme Forward Checking(FC) : Dans cet algorithme, seule la première condition du Look-ahead est appliquée. La règle de consistance d'arcs est alors appliquée entre le nœud courant instancié par une valeur choisie et tous les nœuds futurs. Si un nœud futur donné se retrouve avec un domaine vide, un retour arrière est effectué et une autre valeur est choisie pour le nœud courant.

2- Algorithme Full-Look-ahead(FL) : Dans cet algorithme, les deux conditions du look-ahead sont appliquées. En plus d'être appliquée entre le nœud courant (instancié par une valeur donnée) et les nœuds futurs, la règle de consistance d'arcs est aussi appliquée entre les nœuds futurs entre eux ce qui permet d'éliminer plus de valeurs inconsistantes.

3- Algorithme Partial-Look-ahead(PL) : Cet algorithme est une variante du Full-Look-ahead dans laquelle la règle de consistance n'est pas appliquée entre les nœuds futurs mais entre chaque nœud futur et ses propres nœuds futurs.

4- Algorithme Real Full Look-ahead(RFL) : Dans cet algorithme, l'algorithme de consistance d'arcs est appliqué sur toutes les variables à chaque fois qu'une variable donnée est instanciée.

Comme nous venons de le montrer, la principale différence entre ces algorithmes est le degré d'utilisation de la consistance d'arcs. En effet, la consistance d'arcs est totalement appliquée dans le Real full Look-ahead, partiellement dans le Partial Look ahead, le Forward Check, le Full Look ahead et le Backtrack et elle n'est pas utilisée dans de la recherche de solutions par la méthode générer et tester.

Nudel[Nudel 88] a comparé les cinq algorithmes sur les exemples des n reines[6] et des n reines "confus"[7]. Sur ces exemples, l'algorithme du Forward-check présente les meilleurs résultats. Nudel se joint alors à Haralick et Elliot[Haralick 80], Gashing[Gashing 78], Dechter et Meiri[Dechter 89] pour conclure qu'il est plus avantageux d'appliquer la consistance d'arcs mais seulement d'une manière limitée (Forward-Check). Les résultats de tests que nous présenterons dans la troisième partie de ce rapport confirmeront ces constatations.

3.3 Propagation de contraintes en raisonnement temporel

Le raisonnement temporel constitue l'un des champs de l'intelligence artificielle dans lesquels les techniques de propagation de contraintes sont largement utilisées. En effet, les informations temporelles (symboliques ou numériques) se codent naturellement sous forme de relations (ou contraintes). Nous allons voir dans ce qui suit, l'utilisation des techniques de propagation de contraintes sur les deux types de relations(symboliques et numériques).

6. L'exemple des n reines consiste à placer n reines dans un échiquier n x n de manière à ce qu'aucune reine ne puisse être attaquée par une autre reine

7. L'exemple des n reines "confus" consiste à placer n reines, chaque reine par ligne, de manière à ce que chaque reine attaque l'autre, chaque paire de reines doit être placée sur la même colonne ou sur la même diagonale

3.3.1 Propagation de contraintes symboliques

En section 2.2.3 nous avons évoqué les travaux de Allen[Allen 83][Allen 84] dans lesquels il définit un calcul sur les intervalles basé sur 13 relations permettant de spécifier toutes les positions relatives entre deux intervalles de temps données. Une relation symbolique entre deux intervalles de temps est alors une disjonction de ces relations de base.
Allen définit un réseau de contraintes dans lequel les nœuds représentent les intervalles et les arcs sont étiquetés par les relations disjonctives entre les nœuds correspondants.
Pour assurer la consistance partielle (consistance de chemins), Allen utilise un algorithme de consistance de chemins. La règle de 3-consistance (cf section 3.2.2), permettant de réduire les différentes relations qualitatives disjonctives en supprimant de chaque relation disjonctive les relations de base de Allen inconsistantes, sera définie comme suit :

$$\forall i,j,k \in [1,n] \quad R_{ij} = R_{ij} \cap R_{ik} \circ R_{kj}$$

Le calcul de la composition de deux relations disjonctives permettant de déduire la position relative de deux intervalles en fonction des relations les liant à un troisième intervalle utilise la table de transitivité d'Allen (cf figure 7.2).
La figure 3.3 présente l'application de la règle de 3-consistance sur un réseau de contraintes temporelles.

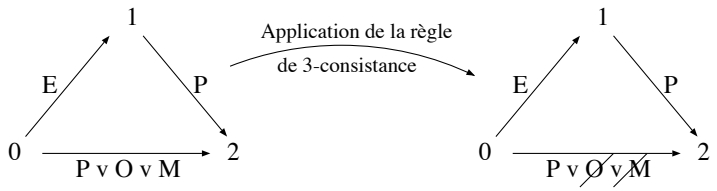


FIG. 3.3: *Consistance de chemins dans un réseau de contraintes temporelles.*

Nous présentons dans ce qui suit l'application de l'algorithme PC-2[Mackworth 77] aux relations temporelles [van Beek 92] :

Algorithme PC-2

Début
1. $Q \leftarrow \{(i,j,k),(k,i,j) \mid 1 \leq i < j \leq n,\ \ 1 \leq k \leq n,\ \ k \neq i,\ \ k \neq j\}$
2. **Tant-Que** $Q \neq NIL$ **Faire**
3. `Retirer` (i,k,j) `de` Q
4. $t \leftarrow C_{ij} \cap C_{ik}\ .\ C_{kj}$
5. **Si** $(t \neq C_{ij})$ **Alors**
6. $C_{ij} \leftarrow t$
7. $C_{ji} \leftarrow INVERSE(t)$
8. $Q \leftarrow Q \cup \{(i,j,k),(k,i,j) \mid 1 \leq k \leq n,\ \ k \neq i\ \ et\ \ k \neq j\}$
9. **Fin-Si**
10. **Fin-Tant-Que**

Fin

Une version modifiée de cet algorithme, proposée par vanBeek et Manchak[van Beek 96] est présentée en sous section 7.2.2.

Ladkin et Reinefeld[Ladkin 92a] utilisent la composition de matrices relationnelles pour appliquer l'algorithme PC-1 sur les relations temporelles. Une matrice relationnelle est une matrice de relations qualitatives représentant le réseau de contraintes (de taille n x n où n est le nombre de variables du réseau) et telle que chaque entrée M_{ij} correspond à la relation qualitative R_{ij} entre la paire de nœuds (i, j). Le calcul matriciel permettra alors d'appliquer la règle de 3-consistance afin de réduire les différentes relations symboliques (entrées de la matrice) :

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

La règle de transitivité sur laquelle se base l'algorithme PC-1 pour assurer la consistance de chemins est traduite comme suit :

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

où :

+ : Opérateur de conjonction entre deux relations disjonctives.
$*$: Opérateur de composition de deux relations disjonctives(basé sur la table de transitivité d'Allen[Allen 83]).

L'algorithme PC1 est présenté alors comme suit :

Début
  Répéter
    $M \leftarrow M \cdot M \circ M$
  jusqu'a $M = M \circ M$
  `Retourner` $M \neq 0$
Fin

L'algorithme est appliqué plusieurs fois sur la matrice M jusqu'a stabilité du réseau ($M = M \circ M$, dans ce cas le réseau vérifie la consistance de chemins) ou l'échec ($M = 0$, dans le cas où une entrée de la matrice M devient nulle).

Rappelons qu'un algorithme de consistance de chemins assure une consistance partielle du problème. Une étape de recherche de solutions est généralement nécessaire pour déterminer un scénario (solution symbolique constituée d'un ensemble de relations de base d'Allen) consistant.
La figure 3.4 montre un exemple du problème des trois sommets[Allen 83]. Bien que la

3-consistance soit vérifiée pour chaque triplet de nœuds, le réseau est inconsistant.

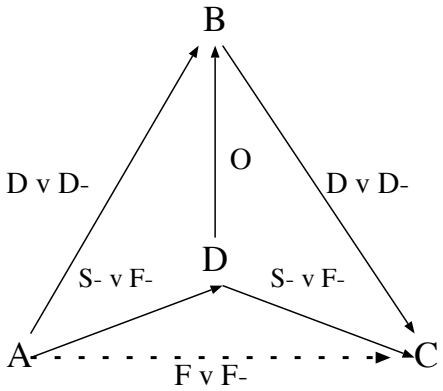


FIG. 3.4: *Problème des 3-sommets*.

La vérification de la consistance d'un réseau de contraintes symbolique est un problème NP-complet. Plusieurs travaux ont alors été effectués afin de déterminer des classes de réseaux de contraintes qualitatives dans lesquels la consistance de chemins implique la consistance du problème. La résolution de tels réseaux sera alors effectuée avec une complexité polynômiale en temps. Nous présentons dans ce qui suit les différentes classes de réseaux temporels et identifions celles où la consistance de chemins implique la consistance du problème.

Hiérarchie des réseaux qualitatifs

Un réseau de contraintes qualitatives est appelé réseau IA (IA pour Interval Algebra ou algèbre d'intervalles) si les contraintes qualitatives correspondantes sont des disjonctions de relations de base de Allen. Notons que tout réseau qualitatif peut être représenté par un réseau IA. La classe de réseaux IA contient $2^{13} = 8192$ relations. Un réseau IA tel que toutes les contraintes peuvent être traduites en relations de l'algèbre d'instants sur les extrémités des intervalles (contraintes ne contenant pas la disjonction $P \vee P^{\smile}$, cette dernière disjonction ne peut être exprimée dans l'algèbre d'instants) est appelé réseau PA (PA pour Point Algèbre ou algèbre de points) ou "*réseau pointisable*". La sous-classe des réseaux PA contient 188 relations.
Un réseau de contraintes PA pouvant être traduit en "*réseau de relations convexes*" de l'algèbre d'instants est appelé réseau CPA (CPA pour Convex Point Algebra ou algèbre convexe de points) ou "*réseau convexe*". Une relation convexe, en algèbre de points, est une conjonction de relations de base prises dans l'ensemble $\{<, >, \leq, \geq, =\}$ (donc ne contenant pas la relation $\neq$). La sous-classe des réseaux CPA contient 83 relations[8].
Un réseau IA tel que les contraintes peuvent être traduites en conjonction de clauses de Horn utilisant les relations $\{\leq, =, \neq\}$ (appelées aussi clauses ORD-HORN), exprimant les relations entre les points extrémités des intervalles, est appelé réseau N.B[Nebel 95](appelé aussi "*réseau Hornisable*" ou réseau ORD-HORN). La sous-classe de "*réseaux Hornisables*" est une sur-classe stricte de la sous-classe de "*réseaux pointisables*". Elle contient 868 relations ce qui représente 10% des relations de la classe IA.

8. Une énumération des relations des sous-classes PA et CPA est présentée dans [van Beek 90b]

La figure 3.7 illustre la hiérarchie des différentes classes de réseaux.

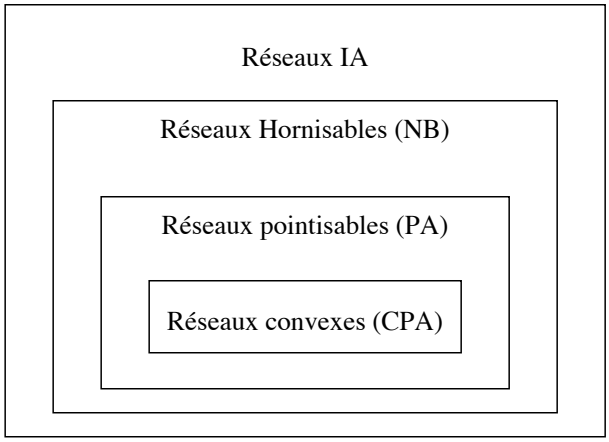


FIG. 3.5: *Hiérarchie des réseaux de contraintes qualitatives.*

La résolution de réseaux IA est un problème NP-complet. Nebel et Burckert[Nebel 95] démontrent que la sous-classe ORD-HORN est la plus grande sous-classe de réseaux IA dans laquelle la consistance de chemins implique la consistance du problème de contraintes. C'est donc la plus grande sous-classe des réseaux "traitables" (dans lesquels la consistance de réseaux de contraintes peut être obtenue avec un algorithme polynômial). Étant donné que les sous-classes PA et CPA sont des sous-classes de la sous-classe ORD-HORN, la consistance de chemins implique aussi la consistance du réseau de contraintes dans ces deux sous-classes. La consistance de chemins implique par contre la consistance globale uniquement pour la sous-classe CPA.
Suite aux travaux de Ligozat[Ligozat 90][Ligozat 91], Bessière, Isli et Ligozat[Bessière 96] ont déterminé deux sous-classes de la sous-classe ORD-HORN dans lesquelles la consistance de chemins implique la consistance globale. Une de ses deux sous-classes contient 558 relations et couvre donc 60 % des relations de la classe ORD-HORN.

Le fait de passer des réseaux CPA aux réseaux IA permet de gagner en expressivité. En effet, les réseaux IA permettent d'exprimer tous les réseaux qualitatifs temporels, ce qui n'est pas le cas des réseaux ORD-HORN et encore moins des réseaux PA et CPA. Par contre, le problème de vérification de la consistance des réseaux IA est un problème NP-complet alors qu'il est polynômial en temps pour les trois autres classes de réseaux. Pour ces derniers, ce problème est résolu par un algorithme de consistance de chemins (de complexité polynômiale). Van Beek[van Beek 90a][van Beek 92] propose, par exemple, un algorithme permettant de vérifier la consistance de chemins et donc la consistance globale avec une complexité $O(n^2)$ en temps pour les réseaux CPA.

En ce qui nous concerne, nous avons choisi les réseaux IA pour représenter les informations qualitatives. Nous gagnons de ce fait en expressivité étant donné que les réseaux IA permettent de représenter tous les réseaux temporels qualitatifs. Cependant, avec ce choix nous nous confrontons au problème de complexité. Pour palier ce problème, nous nous sommes basés sur les techniques de propagation de contraintes (algorithmes de consistance locale et

stratégies de recherche de solutions) que nous avons étudiées pour définir un algorithme de résolution de contraintes. Pour améliorer l'algorithme de résolution que nous proposons et faire face aux problèmes de complexité, nous avons étudié les propriétés des contraintes temporelles pour définir des heuristiques qui seront utilisées par les algorithmes de consistance locale que nous appliquons.

3.3.2 Propagation de contraintes numériques

Les approches numériques manipulent explicitement des nombres, généralement des intervalles de temps de durée fixe ou variable.

Dans la section 2.2.2 nous avons évoqué une représentation de variables temporelles par des intervalles de durée fixe et dont le domaine est défini par des bornes numériques[Pape 87][Vere 83]. Des relations temporelles entre chaque paire de domaines sont alors utilisées pour vérifier la compatibilité des domaines. Dans le cas de domaines à valeurs discrètes, la vérification de la compatibilité de deux variables liées par une relation temporelle se traduit par l'élimination des étiquettes impossibles. Comme on ne peut pas énumérer d'étiquettes dans le cas de domaines continus (vu qu'il y a dans ce cas une infinité d'étiquettes), on modifiera les bornes du domaine.
Soit, par exemple, deux variables temporelles A et B représentées par deux intervalles donnés et tel que A précède B . Soit D_A(resp D_B) le début au plus tôt de A(resp de B). La relation $D_B \geq D_A + d_A$ doit donc être vérifiée d'où la compression du domaine de B (voir figure 3.6).

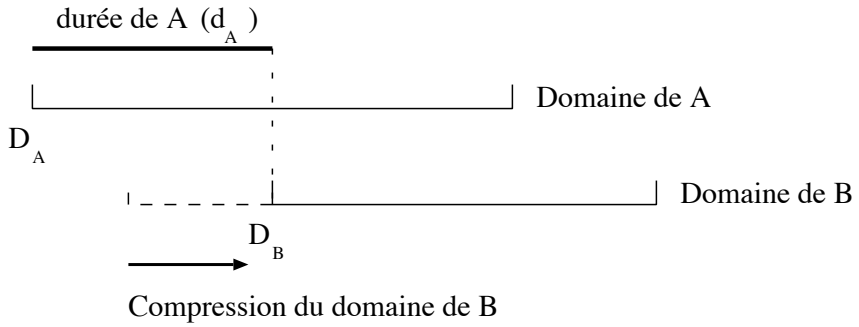


FIG. 3.6: *Effet du domaine de A sur le domaine de B : A P B*

Si A et B appartiennent à un graphe de contraintes temporelles, la compression du domaine de B va donc entraîner d'autres compressions. Un algorithme de consistance d'arcs permettra alors de propager ces modifications jusqu'à stabilité du réseau.

Rit[Rit 86][Rit 88] propose une méthode originale qui repose sur une interprétation géométrique des problèmes de contraintes temporelles. Les objets temporels sont des événements contraints numériquement par l'ensemble de leurs occurrences possibles appelé DOP (pour Domaine d'Occurrences Possibles ou SOPO pour Set Of Possible Occurrences). Un DOP est un ensemble d'intervalles fermés et de durée variable.

Les événements (auxquels nous associons un DOP pour chacun d'entre eux) sont reliés entre eux par des relations temporelles (disjonction de relations de base d'Allen).
Un DOP est représenté graphiquement sous forme d'une région dans le plan des occurrences défini par les axes $(debut, fin)$. Une occurrence d'un SOPO donné est représentée par un point appartenant au demi plan limité par la première bissectrice d'équation $debut = fin$.

La propagation de contraintes est assurée par l'algorithme de filtrage de Waltz[Waltz 75] pour éliminer tous les intervalles inconsistants de chaque DOP.
Pour ce faire, Rit définit les notions suivantes :

Région permise par une occurrence et une relation : Soit o une occurrence et R une relation de base d'Allen. La relation R peut être interprétée par rapport à l'occurrence o par l'ensemble des occurrences qui vérifient la relation R avec o :

$$P(o, R) = \{\ x \mid R(o, x)\ vrai\ \}$$

La figure 3.7 présente l'interprétation de toutes les relations de base d'Allen par rapport à une certaine occurrence o.

Région permise par une occurrence et une relation disjonctive : C'est l'union de toutes les régions permises par l'occurrence et chaque relation de base de la relation disjonctive.

Région permise par un DOP et une relation : C'est la réunion des régions permises par toutes les occurrences du DOP.

Raffinement : C'est l'opération ρ d'intersection d'un DOP O_2 avec la région permise par le DOP O_1 et la relation R_{12} :

$$\rho(O_1, R_{12}, O_2) = P(O_1, R_{12}) \cap O_2$$

Pour propager les contraintes, Rit s'inspire de l'algorithme de filtrage de Waltz[Waltz 75] et propose un algorithme basé sur la notion de raffinement et permettant d'éliminer les occurrences impossibles. Cet algorithme consiste à calculer l'intersection de chaque domaine avec les régions qui lui sont permises tant qu'il y a lieu de le faire. Cet algorithme n'élimine que les occurrences impossibles. Il n'est cependant pas complet dans le sens où il ne les élimine pas toutes.

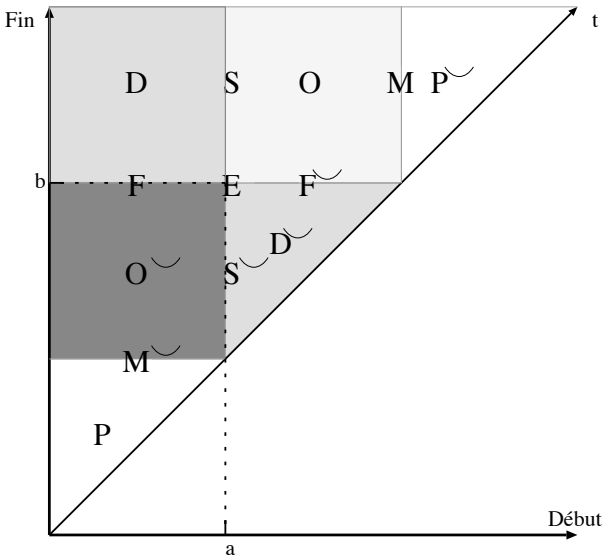


FIG. 3.7: *Interprétation géométrique des primitives d'Allen à l'aide de régions permises.*

Chapitre 4

Conclusion

Nous avons présenté plusieurs schémas de représentation de l'information temporelle en faisant la distinction à chaque fois entre deux aspects : l'aspect symbolique et l'aspect numérique.

Les représentations symboliques du temps sont fondées sur la logique. Nous en distinguons trois catégories :

- la logique classique ne donnant aucun statut particulier au temps.
- la logique modale temporelle, généralement utilisée en informatique théorique,
- et enfin la logique réifiée, largement utilisée en intelligence artificielle.

La logique réifiée se divise en deux classes :

- l'algèbre d'instants [McDermott 82] dans laquelle l'instant est défini comme une "*photographie instantanée de l'univers*",
- et l'algèbre d'intervalles [Allen 83][Allen 84] dans laquelle l'intervalle est l'élément principal permettant de représenter des entités fondamentales tel que les événements, les processus et les propriétés.

Notre choix s'est porté sur la logique réifiée et plus précisément l'algèbre d'intervalles pour représenter l'aspect symbolique du temps dans le modèle que nous proposons. En effet, cette approche offre une richesse d'expression ainsi qu'une élégance de traitement des relations symboliques disjonctives, bien que les problèmes traitant des systèmes fondés sur cette approche soient NP-complets (ce qui n'est pas le cas des systèmes fondés sur l'algèbre de points, cf section 3.3.1).

Tout en profitant des avantages de cette approche, nous pouvons palier les difficultés que nous venons de citer en utilisant les techniques de propagation de contraintes ainsi que les stratégies de résolution de problèmes de contraintes présentées dans le troisième chapitre de cette partie.

Parmi les représentations qui traitent l'aspect numérique, certaines sont exclusivement fondées sur la recherche opérationnelle, d'autres conjuguent des techniques d'intelligence artificielle avec la recherche opérationnelle. Notons enfin, une méthode originale proposée par Rit[Rit 86][Rit 88] qui repose sur une interprétation géométrique.
Bien que les techniques basées sur la recherche opérationnelle soient efficaces pour des cas simples, elles s'avèrent inefficaces dans des situations plus compliquées. Elles trouvent, par exemple, des difficultés à exprimer des contraintes non linéaires.
D'autres méthodes, plus récentes, se basent sur les intervalles numériques pour supporter l'aspect métrique du temps. Ces intervalles numériques varient à l'intérieur de fenêtres temporelles, domaines de variation avec date de début au plus tôt et date de fin au plus tard.
L'interprétation des informations numériques ainsi représentées nécessite l'utilisation de techniques de satisfaction de contraintes. La vérification de la consistance de ces informations passe dans ce cas par la manipulation des domaines d'intervalles (par compression de ces domaines). Des algorithmes de propagation de contraintes sont alors utilisées.

Nous avons choisi cette approche comme point de départ pour supporter l'aspect numérique du temps dans le modèle que nous proposons.
En effet, ayant opté pour l'intervalle comme entité de base de notre modèle, nous l'utilisons pour représenter les différents événements temporels. Chaque événement est contraint numériquement à l'intérieur d'une fenêtre temporelle ayant une date de début au plus tôt et une date de fin au plus tard. Ces fenêtres temporelles constituent les domaines de variation des différents événements. Nous nous basons cependant sur un domaine discret du temps. De ce fait, ce sont les intervalles que nous manipulons, étant donné que nous avons dans ce cas des domaines finis d'intervalles, et non pas les domaines d'intervalles.
Cela nous permet de ramener l'interprétation des informations numériques en un problème d'étiquetage qui s'adapte bien aux techniques de satisfaction de contraintes que nous avons présenté au chapitre trois.

Après avoir défini un modèle permettant de supporter des informations numériques et symboliques, nous devons établir un outil de raisonnement permettant d'exploiter ces informations et répondre aux besoins suivants :

- vérifier la consistance globale d'un réseau de relations temporelles représentant un problème de contraintes donné,
- déterminer un scénario cohérent (une solution) dans le cas où le réseau est consistant, et
- offrir une exécution rapide et un temps de réponse acceptable permettant d'exploiter directement les différents résultats.

Pour ce faire, nous nous sommes basés sur l'étude des différentes techniques de propagation de contraintes temporelles que nous avons présentée dans le troisième chapitre pour définir un algorithme de résolution de problèmes de satisfaction de contraintes temporelles que nous présentons dans la seconde partie du rapport.
Cet algorithme utilise une technique de recherche de solutions avec retour arrière. Afin d'améliorer ses performances, des algorithmes de consistance locale sont utilisés comme filtre pour

supprimer certaines inconsistances durant la recherche ce qui permet de réduire la combinatoire de la recherche arborescente de solutions.
Parmi les algorithmes de consistance locale que nous avons étudiés, nous avons choisi les algorithmes AC-3, AC-4 et AC-7 pour assurer la consistance d'arcs et PC-1, PC-2 pour assurer la consistance de chemins.

Parmi les stratégies de recherche de solutions que nous avons étudiées, nous avons opté pour la stratégie Real Full Look ahead. Cependant, bien que cette dernière permette d'enlever le plus d'inconsistances d'arcs (par rapport aux autres stratégies étudiées) durant la recherche, des tests que nous avons effectués et présentés dans la troisième partie de ce rapport montrent qu'elle est moins performante qu'une stratégie utilisant partiellement un algorithme de consistance locale, à savoir la stratégie Forward Checking. Nous rejoignons donc Haralick, Gashing et Dechter [Haralick 80][Gashing 78] [Dechter 89] pour affirmer que cette dernière stratégie est la meilleure parmi toutes les stratégies que nous avons étudiées.

Deuxième partie

Modélisation des informations temporelles et raisonnement

Chapitre 5

Introduction

Depuis plusieurs années, les tentatives de prise en compte du temps sous toutes ses formes ont été nombreuses et variées notamment en intelligence artificielle. Certains travaux traitent l'aspect numérique du temps alors que d'autres se fondent sur l'aspect symbolique.
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 d'Allen. Deux réseaux permettent alors 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 supporter les deux types de contraintes, Tolba[Tolba 91][Tolba 92] propose un modèle généralisant l'algèbre d'intervalles afin d'intégrer les contraintes métriques. Il utilise alors une propagation de contraintes à deux niveaux : symbolique et numérique. Pour chaque type de contraintes a été implanté un propagateur de contraintes. A chaque fois que la propagation d'un type de contraintes arrive à un point de stabilité, les résultats obtenus sont transmis à l'autre propagateur. Cette tache est assurée par un module appelé module de communication.

Le travail de Tolba constitue un point de départ du modèle que nous proposons. Comme nous l'avons précisé au chapitre quatre de la partie précédente, l'événement étant l'objet temporel que nous manipulons, ce dernier est représenté par un intervalle numérique. Les fenêtres temporelles servent de domaines de variation des événements et permettent de contraindre nu-

mériquement ces derniers. Étant donné que nous nous basons sur un domaine discret du temps, le domaine de variation de chaque événement contient un ensemble fini d'intervalles. Ceci nous permet de ramener le traitement des informations temporelles supportées par notre modèle en un problème d'étiquetage pouvant aisément être résolu par les techniques de propagation de contraintes présentées dans la première partie de notre thèse.
Par rapport aux approches quantitatives qui se basent sur les intervalles numériques pour supporter les informations métriques, la différence entre ces approches et le modèle que nous proposons se situe dans la manière de représenter et surtout d'exploiter ces informations. Alors que ce sont les domaines d'intervalles qui sont manipulés dans ces approches (par compression des domaines), ce sont les intervalles contenus dans les domaines des événements et non pas les domaines de ces derniers qui sont manipulés dans notre modèle. En effet, nous utilisons une représentation discrète et non pas continue du temps comme c'est le cas de ces méthodes. Nous traduisons alors le domaine de variation d'un événement en un ensemble fini d'intervalles ce qui permet de traiter facilement le cas de domaines non convexes (domaines contenant des intervalles de temps durant lesquels un événement ne peut avoir lieu).
La différence entre le modèle de Tolba et le nôtre se situe dans la manière de raisonner sur les informations temporelles. Tolba se base sur l'algorithme GAC-4[Mohr 88](généralisation de l'algorithme AC-4 pour supporter et manipuler des contraintes n-aires) pour assurer la propagation de contraintes sur les deux types de réseaux.
Pour pouvoir appliquer cet algorithme, le graphe de contraintes symboliques est transformé en un hyper-graphe dans lequel chaque nœud est étiqueté par toutes les relations possibles entre la paire de nœuds correspondante du graphe initial. La figure 5.1 illustre la transformation d'un graphe de contraintes symboliques en un hyper-graphe.

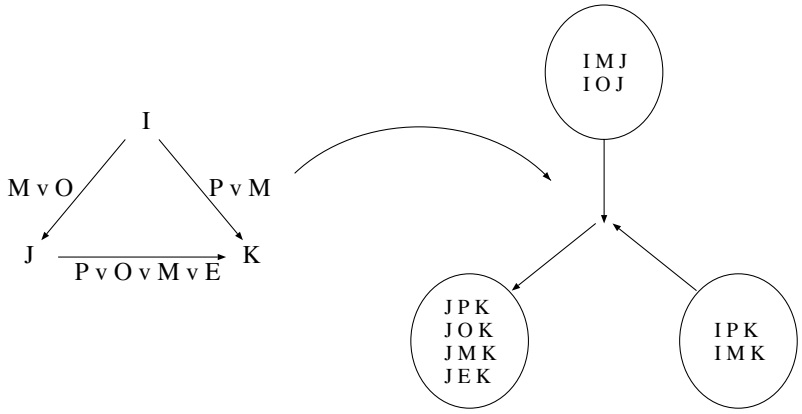


FIG. 5.1: *Transformation d'un réseau de contraintes symboliques en un hyper-graphe*

De manière analogue, le graphe de contraintes numériques est transformé en un hyper-graphe dans lequel chaque nœud est étiqueté par toutes les paires d'occurrences possibles des domaines des deux nœuds pour chaque relation de base de Allen. La figure 5.2 illustre la transformation d'un graphe de contraintes numériques en un hyper-graphe.

En plus du coût de transformation d'un graphe de contraintes en un hyper-graphe, le processus d'énumération des étiquettes symboliques et numériques est très coûteux. De plus, bien que AC-4 ait une complexité intéressante en temps au pire cas, il n'en est pas de même dans le cas

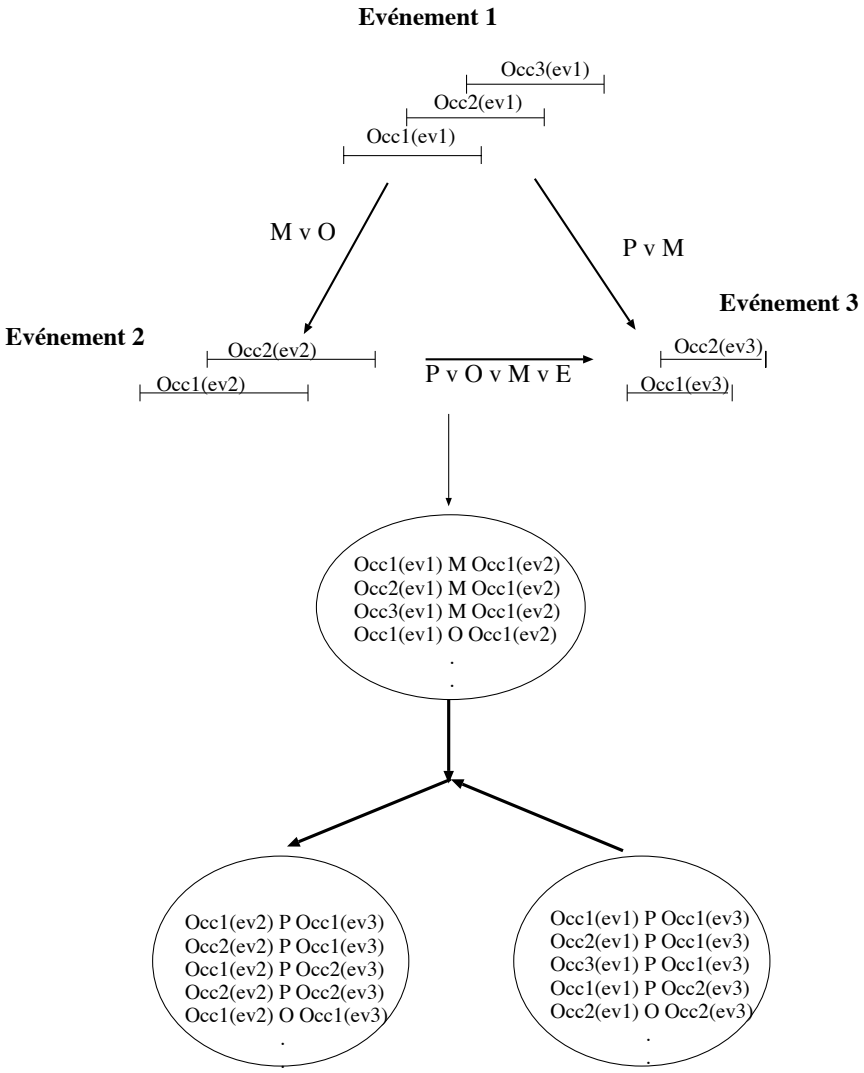


FIG. 5.2: *Transformation d'un réseau de contraintes numériques en un hyper-graphe*

général (il est par exemple moins performant que AC-3 [Wallace 93]). Nous confirmerons cette remarque dans les tests que nous allons présenter dans la troisième partie de notre rapport.
Nous considérons enfin que la propagation de contraintes à deux niveaux n'est généralement pas la meilleure solution pour résoudre des problèmes de contraintes temporelles symboliques et numériques. En effet, d'après les tests que nous avons effectués et présentés dans la troisième partie de notre rapport, le temps mis dans l'étape de propagation symbolique afin de réduire le nombre de relations qualitatives utilisés dans l'étape d'élagage lors de la propagation numérique, n'est pas compensé lors de la recherche de solutions numériques. Toutefois d'autres tests présenté dans la même partie du rapport montrent que la propagation symbolique est souvent nécessaire pour détecter les inconsistances de problèmes de contraintes temporelles.

Dans le chapitre six nous présentons notre modèle de représentation des informations temporelles qualitatives et quantitatives. Ces dernières seront définies par des contraintes supportées par un graphe temporel.

Nous présentons dans le chapitre sept l'algorithme général que nous avons proposé et qui permet de résoudre des problèmes de contraintes temporelles représentés par le modèle que nous proposons. Afin d'améliorer les performances de cet algorithme, des techniques de satisfaction de contraintes sont utilisées.
Nous terminons enfin par une conclusion sur le modèle ainsi que l'outil de raisonnement que nous proposons.

Chapitre 6

Modélisation des informations temporelles

L'objet temporel de base que nous considérons dans notre modèle de représentation est l'événement défini par le couple (ϕ, I) où ϕ est une assertion logique atemporelle et I l'intervalle de temps durant lequel ϕ est vraie. À partir de cet événement nous définissons les informations symboliques et numériques de la manière suivante :

- l'information symbolique situe ou relie qualitativement deux événements temporels entre eux,
- l'information numérique permet de situer un événement par rapport à un référentiel temporel.

Les informations temporelles qualitatives et quantitatives, ainsi définies, se codent naturellement sous forme de contraintes. En effet, l'information symbolique se traduit facilement en une relation permettant de situer deux événements temporels entre eux. Nous pouvons, par exemple, traduire la connaissance suivante :
"*La tâche 1 se termine avant le début de la tâche 2*" par la relation :

$$tache_1 \ avant \ tache_2 \text{ où}$$

$tache_1$ et $tache_2$ sont deux événements décrivant les intervalles de temps durant lesquels les tâches 1 et 2 sont effectuées.
De manière analogue, les informations numériques se traduisent aisément sous forme de domaines de variation permettant de contraindre numériquement les différents événements. Soit, par exemple, l'information suivante :
"*La tâche 1 doit s'exécuter après 6 heures et au plus tard avant 10 heures*".
Cette connaissance peut se traduire par :

$$tache_1 \in [\, 6h, 10h \,]$$

Ayant choisi l'intervalle temporel comme entité de base, nous avons opté pour l'algèbre d'intervalles définie par Allen[Allen 83] pour représenter l'aspect symbolique du temps.

Comme nous l'avons signalé auparavant, cette approche offre une richesse d'expression ainsi qu'une souplesse de traitement des relations symboliques disjonctives. De plus, ces relations sont facilement compréhensibles par les utilisateurs, elles peuvent donc être intégrées dans des systèmes plus évolués.
Les domaines de variation des événements temporels définissent les contraintes métriques. Nous les représentons par des fenêtres temporelles ayant une date de début au plus tôt et une date de fin au plus tard. Nous nous basons cependant sur un domaine discret du temps. Les domaines des différents événements seront alors constitués d'ensembles finis d'éléments. Ceci nous permet, en plus de traiter le problème des domaines non convexes que nous avons évoqués en introduction de cette partie, de ramener tout problème de contraintes temporelles représenté par notre modèle en un problème d'étiquetage pouvant être efficacement résolu par des techniques de satisfaction de contraintes.

6.1 Notions de base : Définitions

6.1.1 Droite temporelle

Le modèle que nous proposons est basé sur une représentation discrète du temps. Nous définissons une référence temporelle **Tr** (cf figure 6.1) 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.

6.1.2 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.

6.1.3 Fenêtre temporelle : Domaines d'Occurrences Possibles

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 figures 6.1 et 6.3). 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 entre les dates de début de chaque paire d'intervalles adjacents,
- et la durée de chaque intervalle du DOP.

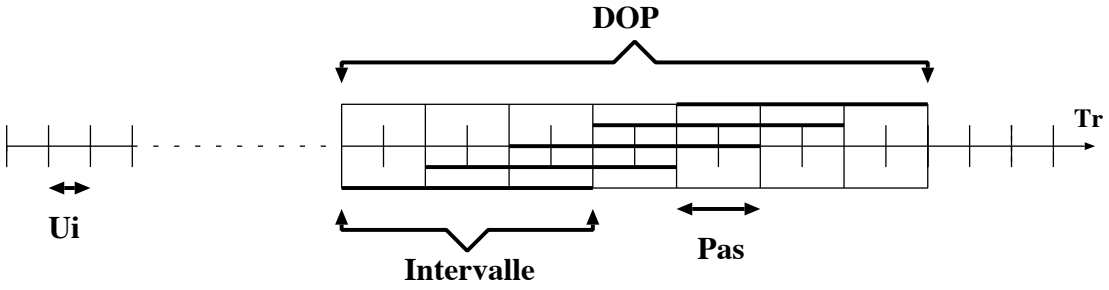


FIG. 6.1: *Représentation d'une contrainte numérique par un DOP*

6.2 Le langage de représentation

6.2.1 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 une assertion logique atemporelle et l'intervalle de temps durant lequel cette assertion logique est vraie : $Evt = (p, I)$. Par la suite on confondra entre la qualification temporelle de l'événement et l'événement lui même.

6.2.2 Les contraintes qualitatives : Relations symboliques entre événements

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 $I_1 \ r_1 \vee r_2 \vee \cdots \vee r_n \ I2$ 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.

La figure 6.2 illustre un exemple d'une relation symbolique entre 2 événements.

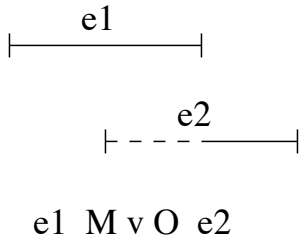


FIG. 6.2: *Relation symbolique entre deux événements*

Exemple 1

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). Cette information peut être représentée par les relations qualitatives suivantes :

$$T_1 \ (P \vee P^{\smile}) \ T_2, \ T_1 \ (P \vee P^{\smile}) \ T_3 \ \text{ et } \ T_2 \ (P \vee P^{\smile}) \ T_3$$

6.2.3 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).
La figure 6.3 illustre le DOP suivant : $[12, 88, 4, 4]$.

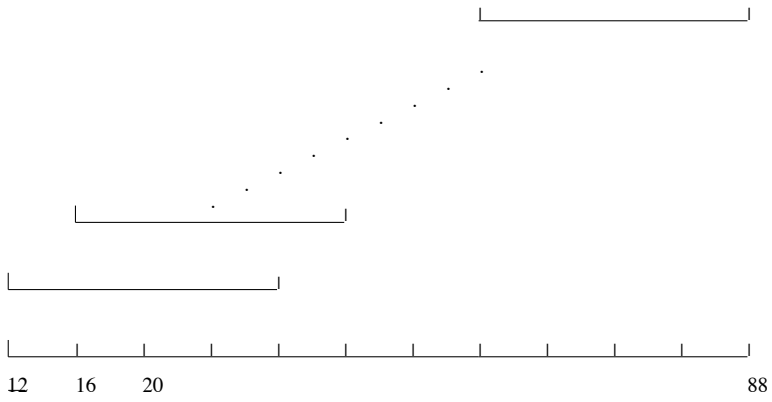


FIG. 6.3: *DOP : Ensemble d'occurrences possibles d'un événement*

Soit e_i un événement contraint numériquement par le DOP suivant :

$$DOP(i) = [inf_i, \ sup_i, \ p_i, \ d_i],$$

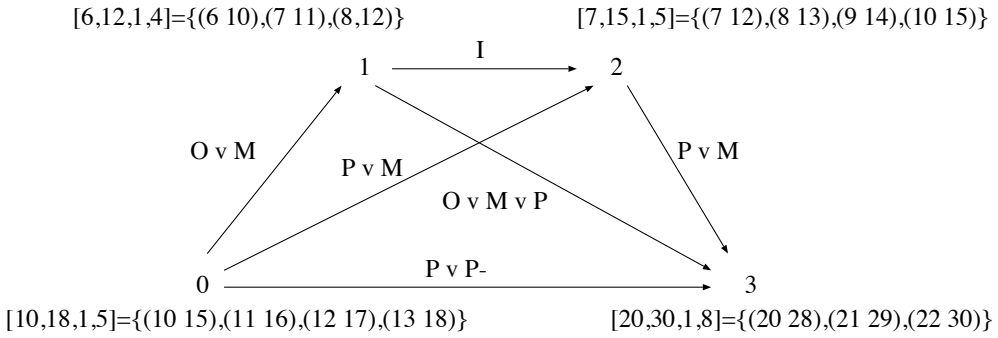
nous définissons l'ensemble E_i des occurrences possibles de e_i, à partir de $DOP(i)$, comme suit :
$E_i = \{occj \mid debut(occ_j) = inf_i + k * p_i,$
$fin(occ_j) = debut(occ_j) + d_i * p_i,$
$fin(occ_j) <= sup_i,$
$k \in [0, \frac{sup_i - inf_i}{p_i} - d_i] \cap \aleph\}$ où
$debut$ et fin sont des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

Exemple 2 :

Soit la tâche T (de durée 4h) devant être effectuée après 1 heure et au plus tard avant 10 heures. Cette tâche peut être représentée par un événement contraint numériquement par le DOP $[1, 9, 1, 4]$ et aura donc pour ensemble d'occurrences : E = {(1 5), (3 7), (5 9)}.

6.3 Réseau de contraintes 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. La figure 6.4 donne un exemple de graphe de contraintes symboliques et numériques tel que nous venons de le spécifier.



I : Relation universelle(disjonction des 13 relations d'Allen).
O : Overlap, M : Meet, P : Precedes, Pi : precedes inverse.

FIG. 6.4: *Graphe de contraintes symboliques et numériques*.

Exemple 3 :

Nous reprenons l'exemple des trois tâches (exemple 1, page 60). Soit une quatrième tâche devant être exécutée avant la tâche 1 et après la tâche 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.

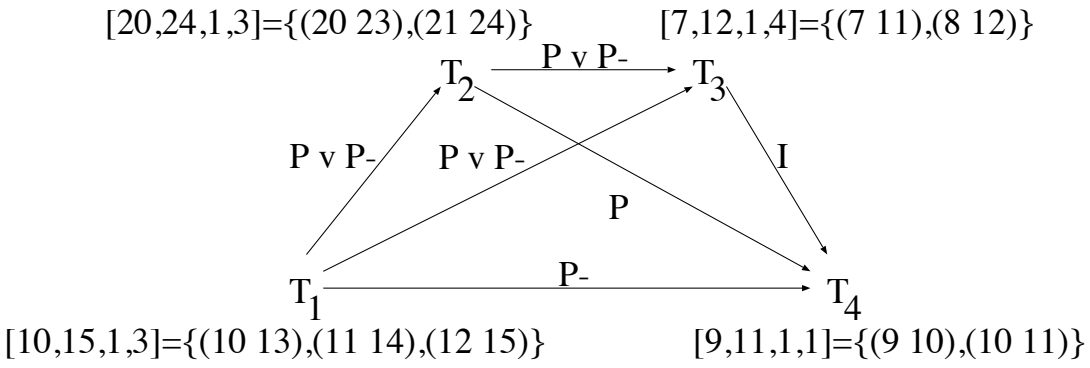


FIG. 6.5: *Graphe temporel représentant un problème d'ordonnancement de tâches.*

Ce problème d'ordonnancement peut être représenté par le graphe temporel de la figure 6.5.

Ce graphe est rendu complet par l'ajout d'arcs étiquetés avec la relation universelle I (disjonction des 13 relations de base d'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).

6.4 Propagation de contraintes numériques et symboliques

Nous avons présenté dans la section précédente un graphe de relations symboliques augmenté par des contraintes métriques. À partir de ce graphe temporel, nous devons être capable d'exploiter les différentes informations supportées qui expriment un problème de contraintes donné. Une des tâches principales est la vérification de la consistance de ce graphe. Pour ce faire, nous avons opté pour des techniques de propagation de contraintes. Les informations que nous manipulons étant de deux types, nous utiliserons alors deux types de propagation : une propagation symbolique et une propagation numérique. Les deux types de propagation sont complètement indépendants ce qui permet de rendre notre représentation plus flexible et plus modulaire. Cela nous permet d'utiliser l'une ou l'autre des deux propagations, sinon les deux en même temps.

6.4.1 Propagation de contraintes numériques

Soit le graphe de la figure 6.5. D_i représente le domaine d'occurrences de la tâche i.
Nous constatons, par exemple, que les valeurs $(10\ 13)$ et $(11\ 14)$ de D_1 sont inconsistantes puisque la tâche 1 doit être appliquée après la tâche 4 et que les seules valeurs possibles de cette dernière tâche sont $(9\ 10)$ et $(10\ 11)$. Nous devons par conséquent éliminer les valeurs $(10\ 13)$ et $(11\ 14)$ de D_1.

Pour détecter et supprimer ces inconsistances, nous avons testé toutes les valeurs de la tâche 1 par rapport à celles de la tâche 4 en utilisant la contrainte symbolique $P^{\smile}$. Toute valeur de la tâche 1, respectivement de la tâche 4, n'ayant pas de support dans D_4 (respectivement D_1) est alors supprimée.
De plus, l'élimination de valeurs dans D_1 (respectivement D_4) va induire d'autres suppressions sur les domaines des tâches en relation directe avec la tâche 1, respectivement la tâche 4. Tous ces domaines vont par conséquent être mis à jour. Ceci implique d'autres suppressions sur le graphe.

Cette opération d'élagage entre chaque paire de nœuds et la gestion de mise à jour d'étiquettes à travers le graphe peut être aisément assurée par des algorithmes de consistance d'arcs. Nous avons donc opté pour de tels algorithmes afin d'assurer la propagation de contraintes numériques. Ces algorithmes utilisent comme fonction d'élagage la traduction en équations arithmétiques des relations symboliques utilisées. La relation $P^{\smile}$ est par exemple exprimée par :

$$I_1 \; P \; I_2 \; \Leftrightarrow fin(I_1) \; < \; debut(I_2)$$

La table 3.1 du chapitre suivant présente toutes les traductions en équations arithmétiques des relations de base d'Allen.

6.4.2 Propagation de contraintes symboliques

Nous considérons cette fois les relations symboliques du graphe de la figure 6.5. En prenant simultanément les trois tâches T_1, T_2 et T_4 nous avons :

$$T_1 \; (P \; \vee \; P^{\smile}) \; T_2, \;\; T_1 \; (P \; \vee P^{\smile}) \; T_4 \;\; \text{et} \;\; T_2 \; (P \; \vee \; P^{\smile}) \; T_4$$

Le fait de fixer, par exemple, la relation P entre T_1 et T_2 provoque une incohérence si l'on considère les 3 nœuds simultanément. La relation P ne peut donc appartenir à aucune solution symbolique. Elle doit donc être supprimée. La contrainte qualitative entre T_1 et T_2 se réduit par conséquent à la relation $P^{\smile}$.
La règle de 3-consistance présentée en 3.3.1 permet de réduire d'une manière élégante et efficace les différentes contraintes symboliques entre chaque paire de nœuds. Nous optons donc pour un algorithme de consistance de chemins, basé sur la règle de 3-consistance, pour assurer la propagation de contraintes symboliques sur le graphe temporel.

Chapitre 7

Raisonnement

Nous avons présenté dans le chapitre précédent notre modèle de représentation des informations temporelles qualitatives et quantitatives. L'événement étant l'entité temporelle que nous manipulons, les contraintes qualitatives sont définies par les relations symboliques permettant de situer deux événements entre eux. Les contraintes numériques sont représentées par des domaines de variation des événements temporels. Étant donné que nous nous basons sur une représentation discrète du temps, ces domaines seront constitués d'ensembles finis d'éléments. Ceci nous permet de ramener tout problème de contraintes représenté par notre modèle en un problème d'étiquetage pouvant être résolu efficacement par des techniques de satisfaction de contraintes citées dans la première partie du rapport.
Notre choix s'est donc porté sur ces techniques pour définir notre outil de raisonnement sur ces informations temporelles. Nous proposons par conséquent un algorithme de résolution de problèmes de satisfaction de contraintes temporelles, qualitatives et quantitatives, tirant profit des techniques de propagation de contraintes.
Nous présentons dans ce chapitre l'algorithme général que nous proposons ainsi que les différentes techniques que nous avons utilisées afin d'améliorer les performances de ce dernier.

7.1 Résolution de problèmes de satisfaction de contraintes temporelles symboliques et numériques

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 (voir chapitre précédent).
L'algorithme classique permettant de résoudre un tel problème est un algorithme de recherche avec retour arrière. Cependant, comme nous l'avons précisé dans le chapitre trois, un tel algorithme pose certains inconvénients surtout lorsque l'espace de recherche est très grand.
Afin de palier ce problème et améliorer l'efficacité de cet algorithme nous utilisons des al-

gorithmes de consistance locale (consistance arcs et de chemins) dans la première étape de l'algorithme général appelée étape de pre-traitement, avant l'application de la deuxième étape appelée étape de recherche, afin de réduire, en éliminant certaines inconsistances, l'espace de recherche et éviter notamment le phénomène de "*trashing*" dû à l'inconsistance d'arcs ou de chemins que nous avons évoqué en section 3.2 de la première partie du rapport et qui affecte les performances de l'algorithme de recherche.

Les contraintes temporelles que nous manipulons étant de deux types, nous utilisons, comme nous l'avons précisé dans le chapitre précédent, les algorithmes de consistance d'arcs sur les contraintes métriques et les algorithmes de consistance de chemins sur les relations symboliques.
En s'inspirant des stratégies de recherche de solutions présentées en sous section 3.2.3 de la première partie de ce rapport, nous pouvons aussi utiliser les algorithmes de consistance d'arcs ou de chemins durant le processus de recherche de solutions ce qui augmentera les performances de notre algorithme de recherche.
La figure 7.1 présente l'algorithme général de résolution de problèmes de contraintes temporelles symboliques et numériques que nous proposons[Mouhoub 96b].

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. 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 et de chemins durant le processus de recherche.
Dans le cas de problèmes consistants l'algorithme détermine une, plusieurs ou toutes les solutions numériques.

L'algorithme général se déroule en deux phases :

– une phase de pré-traitement et

– une phase de recherche de solutions.

Dans ce qui suit, nous détaillons ces deux phases.

1- Phase de pré-traitement

Dans cette phase nous appliquons d'abord l'algorithme de consistance de chemins 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 permet de réduire le nombre de relations de base 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.
L'application de la consistance de chemins pose néanmoins certains problèmes. Tout d'abord, le graphe est rendu complet après cette application. Toutes les contraintes qualitatives qui

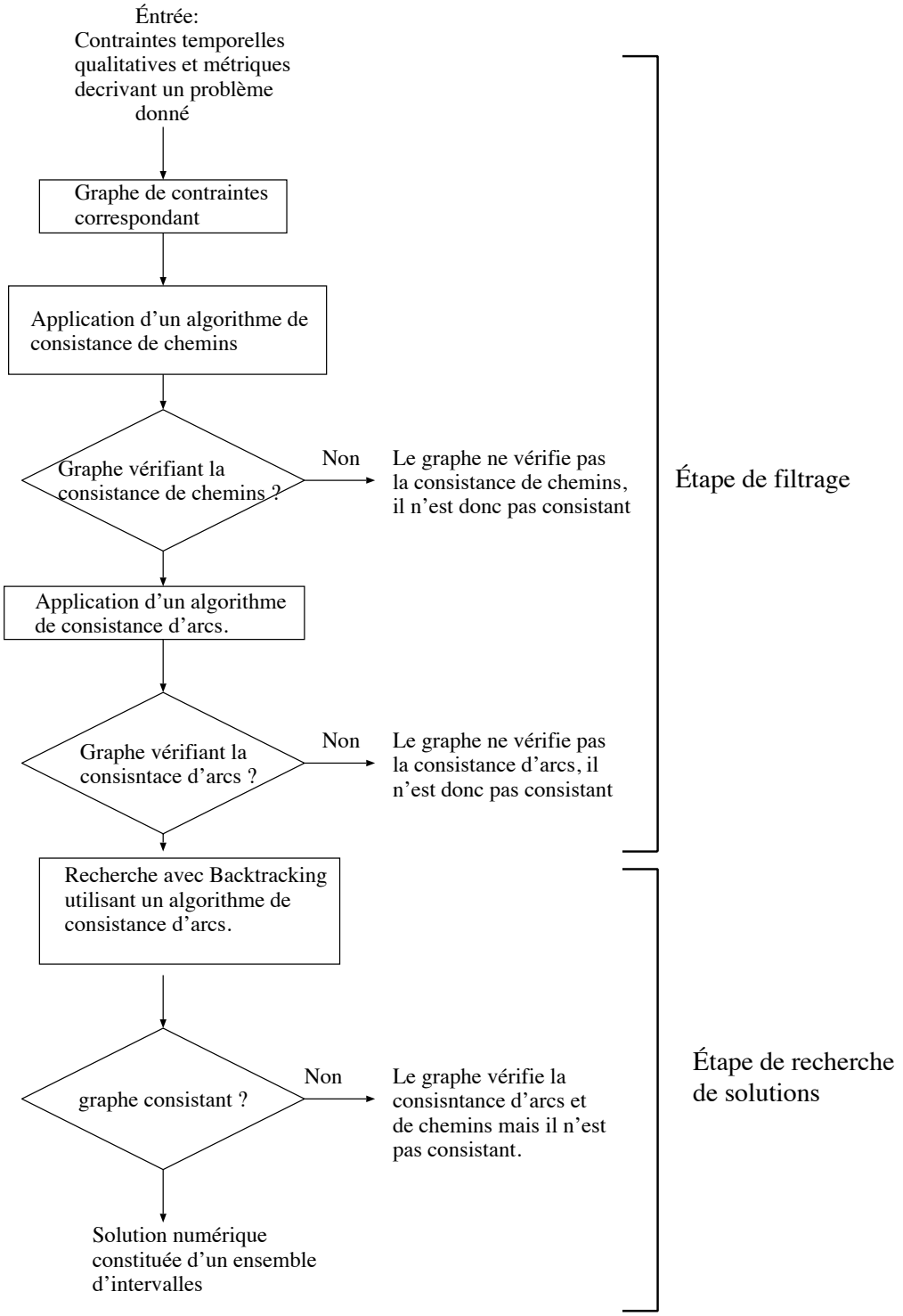


FIG. 7.1: *Algorithme de résolution de problèmes de contraintes temporelles.*

n'existaient pas entre les paires d'événements sont alors créées. De plus, bien que la consistance de chemins permet de réduire le nombre de primitives de Allen par relation qualitative, le temps consommé lors de la vérification de la consistance de chemins n'est pas compensé durant l'étape de recherche de solutions. Cette constatation est vérifiée expérimentalement par les tests que nous avons effectués et présentés dans la partie suivante du rapport.

Dans la deuxième partie de la phase de pre-traitement nous appliquons un algorithme de consistance d'arcs. 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. Nous proposons dans ce qui suit deux méthodes de recherche qui diffèrent au niveau du type de propagation utilisé.

Première méthode :

Cette méthode utilise des algorithmes de consistance d'arcs durant le processus de recherche de solutions. Elle se déroule comme suit :

> *Choisir un nœud donné du graphe et instancier la variable correspondante par 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 par 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.

Fonction $SEARCH\_ALLSOL\_NUM()$
Début
$etiquette \leftarrow etiq\_noeud[0]$
Pour chaque élément $e \in etiquette$ **Faire**
$ACC\_ALL(nr\_ac, etiq\_noeud, solnbre(), 0, e)$
Finpour
Si $solnbre() \neq 0$ **Alors** retourner $Vrai$
Sinon retourner $Faux$
Fin
Fonction $ACC\_ALL(nr\_ac, etiq\_noeud, nbsol, i, e)$
Début
Si $nbsol > nbsol\_max$ **Alors** retourner $Faux$
$existe\_sol \leftarrow Faux$
$etiq\_noeud\_sauve \leftarrow etiq\_noeud$
$etiq_i \leftarrow etiq\_noeud[i]$
$etiq\_noeud[i] \leftarrow e$
Si $taille(etiq_i) = 1$ **ou** $AC(nr\_ac)$ **Alors**
/* L'appel à l'algorithme de consistance d'arcs n'est effectué
que si le domaine du nœud i contient plus d'une valeur */
Si $i = n - 1$ **Alors**
/* Une solution a été trouvée */
$resultats\_num[nbsol][i] \leftarrow e$
$etiq\_noeud \leftarrow etiq\_noeud\_sauve$
retourner $Vrai$
Sinon
$etiquette \leftarrow etiq\_noeud[i+1]$
/* Explorer le prochain nœud */
Pour tout élément e_j de $etiquette$ **Faire**
Si $ACC\_ALL(etiq\_noeud, nbsol, i+1, e_j)$ **Alors**
/* Une solution a été trouvée */
Si $\neg existesol$ **Alors** $existesol \leftarrow Vrai$ **Finsi**
$resultats\_num[nbsol][i] \leftarrow e$
$nbsol \leftarrow nbsol + 1$
Finsi
Tantque $nbsol < solnbre()$ **Faire**
$resultats\_num[nbsol][i] \leftarrow e$
$nbsol \leftarrow nbsol + 1$
Fintantque
Finpour
Finsi
Finsi
$etiq\_noeud \leftarrow etiq\_noeud\_sauve$
retourner existesol
Fin

Nous avons implanté cette méthode de la manière suivante :
la fonction $SEARCH\_ALLSOL\_NUM$, par appel à la fonction ACC_ALL, assure la recherche de solutions numériques. La variable globale $nbsol\_max$ (fonction ACC_ALL) permet de fixer le nombre de solutions numériques à déterminer. Dans le cas où nous voulons vérifier la consistance du graphe (déterminer une solution numérique) $nbsol\_max$ est initialisée à 1. Le vecteur $etiq\_noeud$ contient les domaines de valeurs des différents nœuds. La fonction AC assure la consistance d'arcs par appel à l'un des trois algorithmes : AC-3, AC-4 ou AC-7 (suivant la valeur de nr_ac). Le tableau $resultats\_num$ contient les différentes solutions numériques déterminées.

Deuxième méthode :

Dans cette méthode, nous utilisons, en plus d'un algorithme de consistance d'arcs, un algorithme de consistance de chemins. Plus exactement, pour vérifier la consistance d'un graphe de contraintes nous déterminons d'abord, en appliquant un algorithme de recherche de solutions symboliques utilisant un algorithme de consistance de chemins, une solution symbolique puis nous cherchons une solution numérique pour cette solution symbolique, en procédant de la même façon que dans la première méthode. Si une telle solution est trouvée, nous concluons que le graphe est consistant, sinon nous déterminons une deuxième solution symbolique et cherchons une solution numérique correspondante. Nous procédons ainsi tant qu'une solution numérique n'a pas été trouvée ou qu'il reste des solutions symboliques à explorer. Si une solution numérique ne peut être trouvée pour toutes les solutions symboliques, nous concluons que le graphe est inconsistant.

L'algorithme de recherche de solutions symboliques que nous appliquons est celui proposé par Ladkin et Reinefeld[Ladkin 92a] [Ladkin 92b]. Cet algorithme utilise une stratégie de recherche avec retour arrière selon le principe du Forward Checking. Un algorithme de consistance de chemins est appliqué durant l'étape de recherche de solutions pour réduire l'espace de recherche. Ladkin et Reinefeld utilisent l'algorithme PC-1 pour assurer la consistance de chemins. En ce qui nous concerne, nous appliquons soit l'algorithme PC-1 soit l'algorithme PC-2.
La recherche de solutions symboliques se déroule comme suit :

> *Choisir un arc donné du graphe et fixer une primitive de Allen, parmi les primitives composant la relation symbolique disjonctive, sur cet arc. Appliquer un algorithme de consistance de chemins sur le graphe. Si le nouveau graphe vérifie la consistance de chemins, choisir un autre arc et fixer une autre primitive de Allen sur cet arc, répéter le processus jusqu'a fixer les primitives sur tous les arcs du graphe, on obtiendra alors un scénario consistant (une solution symbolique). Si à une étape donnée de la recherche le graphe ne vérifie plus la consistance de chemins, un retour arrière est effectué, on fixera alors une autre primitive sur le dernier arc choisi. Si on n'arrive pas à fixer les primitives sur chaque arc du graphe, un scénario consistant ne peut être obtenu, nous concluons que le graphe ne vérifie pas la consistance symbolique. Il n'est donc pas consistant.*

Nous l'avons implanté de la manière suivante :
la fonction $SEARCH\_ALLSOL\_SYMB$, par appel à la fonction $CC\_ALLSYMB$, as-

Fonction $SEARCH\_ALLSOL\_SYMB()$
Début
$etiquette \leftarrow M[0,1]$
Pour tout élément e_k de $etiquette$ **Faire**
$CC\_allsymb(M, 0, 1, e_k, 0, solnbre())$
Finpour
Si $solnbre() \neq 0$ **Alors** retourner $Vrai$
Sinon retourner $Faux$
Fin

Fonction $CC\_ALLSYMB(M, i, j, e, depth, nbsol)$
Début
Si $nbsol > nbsol\_symb\_max$ **Alors** retourner $Faux$
$existe\_sol \leftarrow Faux$
$MSAUVE \leftarrow M$
$etiq_{ij} \leftarrow M[i,j]$
$M[i,j] \leftarrow e$
$next \leftarrow choix\_ij(0, 2, n)$
Si $etiq_{ij}$ est une relation atomique ou $PC(nr\_pc, M)$ **Alors**
Si $\neg next.existe$ **Alors**
/* Une solution symbolique a été touvée */
$resultats\_symb[nbsol][depth] \leftarrow e$
$M \leftarrow MSAUVE$
retourner Vrai
Sinon
$etiquette \leftarrow M[next.i, next.j]$
Pour tout élément e_next de $etiquette$ **Faire**
Si $CC\_allsymb(M, n, next.i, next.j, e\_next, depth + 1, nbsol)$ **Alors**
/* Une solution symbolique a été trouvée */
Si $\neg existesol$ **Alors** $existesol \leftarrow Vrai$ **Finsi**
$resultats\_symb[nbsol][depth] \leftarrow a_k$
$nbsol \leftarrow nbsol + 1$
Finsi
Tantque $nbsol < solnbre()$ **Faire**
$resultats\_symb[nbsol][depth] \leftarrow a_k$
$nbsol \leftarrow nbsol + 1$
Fintantque
Finpour
Finsi
Finsi
$M \leftarrow MSAUVE$
retourner existesol
Fin

sure la recherche de solutions symboliques. La variable globale $nbsol\_symb\_max$ (fonction ACC_ALL) permet de fixer le nombre de solutions symboliques à déterminer. Dans le cas où nous voulons vérifier la consistance globale du graphe de contraintes symboliques (déterminer une solution symbolique) $nbsol\_symb\_max$ est initialisée à 1.
La fonction PC assure la consistance de chemins par appel à l'un des deux algorithmes : PC-1 ou PC-2 (suivant la valeur de nr_ac). Le tableau $resultats\_symb$ contient les différentes solutions symboliques déterminées.

Afin d'améliorer la recherche de solutions symboliques, nous utilisons les mêmes heuristiques que celles de la recherche de solutions numériques :

- Nous ordonnons les arcs à traiter dans l'ordre croissant du nombre de primitives de base attachés à chaque arc. Pour chaque arc choisi, nous commençons par sélectionner les atomes correspondant aux primitives "point-meet"[9] étant donné que ces dernières possèdent peu d'entrées dans la table de transitivité de Allen et réduisent donc le nombre de primitives de Allen engendrées par la composition de relations.
- L'appel à l'algorithme de consistance de chemins n'est pas effectué si l'arc choisi contient une seule primitive de Allen.

7.2 Propagation symbolique : Consistance de chemins dans un graphe de contraintes temporelles

7.2.1 Définitions et notions de base

Pour appliquer les algorithmes de consistance de chemins, nous représentons notre graphe de contraintes symboliques par une matrice relationnelle. Soit M cette matrice, chaque entrée de M va correspondre à la relation qualitative R_{ij} entre la paire de nœuds (i, j) (voir 3.3.1 de la première partie du rapport pour la définition d'une matrice relationnelle).

Exemple :

Soit le graphe temporel de la figure 6.5. La matrice relationnelle représentant ce graphe est décrite comme suit :

	1	2	3	4
1	E	$P \vee P^{\smile}$	$P \vee P^{\smile}$	$P^{\smile}$
2	$P \vee P^{\smile}$	E	$P \vee P^{\smile}$	P
3	$P \vee P^{\smile}$	$P \vee P^{\smile}$	E	I
4	P	$P^{\smile}$	I	E

Nous présentons dans ce qui suit quelques notions que nous utiliserons par la suite pour la propagation de contraintes symboliques.

9. Une primitive "point-meet" est une relation telle que les intervalles correspondants commencent ou terminent en même temps

1- Relation symbolique disjonctive

Rappelons qu'une relation symbolique entre deux événements temporels donnés est une disjonction de primitives de base de Allen.
Soit $B\_Allen = \{E, P, D, O, M, S, F, F^{\smile}, S^{\smile}, M^{\smile}, O^{\smile}, D^{\smile}, P^{\smile}\}$ l'ensemble des treize primitives de Allen. La relation symbolique entre deux événements i et j est notée comme suit :

$$R_{ij} = r_1 \vee \ldots r_n \text{ avec } 1 \leq n \leq 13$$

Nous notons $Set\_relation(R_{ij})$ l'ensemble des primitives de la relation R_{ij}.

2- Composition de deux relations disjonctives

La composition de deux relations disjonctives est définie comme suit :

$$R_i \circ R_j = (\sum_{k=1}^{n} r_{ik}) \circ (\sum_{p=1}^{m} r_{jp}) = \sum_{k=1}^{n} \sum_{p=1}^{m} r_{ik} \circ r_{jp}$$

Le calcul des $r_{ik} \circ r_{jp}$ est obtenue à partir de la table de transitivité de Allen présentée en figure 7.2.

	E	P	$P^{\smile}$	D	$D^{\smile}$	O	$O^{\smile}$	M	$M^{\smile}$	S	$S^{\smile}$	F	$F^{\smile}$
E	E	P	$P^{\smile}$	D	$D^{\smile}$	O	$O^{\smile}$	M	m	S	s	F	$F^{\smile}$
P	P	P	I	u	P	P	u	P	u	P	P	u	P
$P^{\smile}$	p	I	$P^{\smile}$	$v^{\smile}$	$P^{\smile}$	$v^{\smile}$	$P^{\smile}$	$v^{\smile}$	$P^{\smile}$	$v^{\smile}$	$P^{\smile}$	$P^{\smile}$	$P^{\smile}$
D	D	P	$P^{\smile}$	D	I	u	$v^{\smile}$	P	$P^{\smile}$	D	$v^{\smile}$	D	u
$D^{\smile}$	$D^{\smile}$	v	$u^{\smile}$	n	$D^{\smile}$	$z^{\smile}$	$y^{\smile}$	$z^{\smile}$	$y^{\smile}$	$z^{\smile}$	$D^{\smile}$	$y^{\smile}$	$D^{\smile}$
O	O	P	$u^{\smile}$	y	v	x	n	P	$y^{\smile}$	O	$z^{\smile}$	y	x
$O^{\smile}$	$O^{\smile}$	v	$P^{\smile}$	z	$u^{\smile}$	n	$x^{\smile}$	$z^{\smile}$	$P^{\smile}$	z	$x^{\smile}$	$O^{\smile}$	$y^{\smile}$
M	M	P	$u^{\smile}$	y	P	P	y	P	a	M	M	y	P
$M^{\smile}$	$M^{\smile}$	v	$P^{\smile}$	z	$P^{\smile}$	z	$P^{\smile}$	b	$P^{\smile}$	z	$P^{\smile}$	$M^{\smile}$	$M^{\smile}$
S	S	P	$P^{\smile}$	D	v	x	z	P	$M^{\smile}$	S	b	D	x
$S^{\smile}$	s	v	$P^{\smile}$	z	$D^{\smile}$	$z^{\smile}$	$O^{\smile}$	$z^{\smile}$	m	b	s	$O^{\smile}$	$D^{\smile}$
F	F	P	$P^{\smile}$	D	$u^{\smile}$	y	$x^{\smile}$	M	$P^{\smile}$	D	$x^{\smile}$	F	a
$F^{\smile}$	$F^{\smile}$	P	$u^{\smile}$	y	$D^{\smile}$	O	$y^{\smile}$	M	$y^{\smile}$	O	$D^{\smile}$	a	$F^{\smile}$

$x = P \vee O \vee M$
$y = D \vee O \vee S$
$z = D \vee O^{\smile} \vee F$
$a = E \vee F \vee F^{\smile}$
$b = E \vee S \vee S^{\smile}$
$u = P \vee O \vee M \vee D \vee S$
$v = P \vee O \vee M \vee D^{\smile} \vee F^{\smile}$
$n = E \vee F \vee D \vee O \vee S \vee F^{\smile} \vee D^{\smile} \vee O^{\smile} \vee S^{\smile}$

FIG. 7.2: *Table de transitivité de Allen*

Exemple :

$$\begin{aligned} R_1 &= P \vee M \\ R_2 &= S \vee O \\ R_1 \circ R_2 &= (P \vee M) \circ (S \vee O) \\ &= (P \circ S) \vee (P \circ O) \vee (M \circ S) \vee (M \circ O) \\ &= P \vee P \vee M \vee P \\ &= P \vee M \end{aligned}$$

3- Intersection de deux relations disjonctives

Soit :

$$R_1 = \sum_{k=1}^{n} r_{ik}, \quad R_2 = \sum_{p=1}^{m} r_{jp}$$

L'intersection des deux relations R_1 et R_2 est définie comme suit :

$$R_1 \cap R_2 = \sum_{s=1} r_s \text{ où } r_s \in Set\_relation(R_1) \cap Set\_relation(R_2)$$

4- Inverse d'une relation disjonctive, inverse d'une composition de deux relations disjonctives

Soit :

$$R_i = \sum_{k=1}^{n} r_{ik}, R_j = \sum_{p=1}^{m} r_{jp}$$

L'inverse de la relation R_i est défini comme suit :

$$R_i^{\smile} = (\sum_{k=1}^{n} r_{ik})^{\smile} = \sum_{k=1}^{n} r_{ik}^{\smile}$$

L'inverse de la composition $R_i \circ R_j$ est défini comme suit :

$$\forall r_i,\ r_j \in B\_Allen\ :\ (r_i \circ r_j)^{\smile} = r_j^{\smile} \circ r_i^{\smile} \quad \text{d'où}$$

$$(R_i \circ R_j)^{\smile} = R_j^{\smile} \circ R_i^{\smile}$$

5- Représentation des relations qualitatives disjonctives par des entiers

Nous avons défini en donnant un ordre aux treize primitives de Allen, toute relation disjonctive de la manière suivante :

$$R = \sum_{i=1}^{13} e_i r_i$$

avec $e_i = 1$ si r_i est contenue dans R et $e_i = 0$ sinon. Cela nous a permis de représenter, en pratique, toute relation disjonctive par un vecteur binaire :

$$R = e_1 e_2 \dots e_{13}$$

avec $e_i = 1$ si la primitive de Allen correspondante existe dans la relation R et $e_i = 0$ sinon.
En donnant l'ordre suivant aux primitives de Allen : $E, P, D, O, M, S, F, F^{\smile}, S^{\smile}, M^{\smile}, O^{\smile}, D^{\smile}, P^{\smile}$, nous aurons par exemple :
E : 0000000000001, P : 0000000000010, POM : 0000000011010

Ainsi, nous pouvons, en pratique, représenter toute relation disjonctive qualitative par un entier décimal égal à la conversion du vecteur binaire correspondant.
La matrice relationnelle M de l'exemple 1 sera transformée en la matrice d'entiers suivante :

	1	2	3	4
1	1	6	6	4
2	6	1	6	1
3	6	6	1	8191
4	1	4	8191	1

Le premier avantage offert par cette représentation est le gain en espace mémoire lors du stockage de la matrice représentant le graphe symbolique. En effet, chaque entrée de la matrice est représentée par un entier qui tient sur 2 octets de mémoire.
Le deuxième avantage est le gain en temps de calcul des opérations d'intersection, de composition et d'inverse de relations qualitatives disjonctives. Les algorithmes de consistance de chemins se basent sur ces opérations pour assurer la cohérence du réseau symbolique. Nous définissons ces opérations comme suit :

a) Intersection et composition de 2 relations :

Soit les relations R_i et R_j définies comme suit :

$$R_i = e_{i1} \dots e_{i13}, \;\; R_j = e_{j1} \dots e_{j13}$$

L'intersection des deux relations R_i et R_j ainsi définies sera alors :

$$R_i \cap R_j = (e_{i1} \dots e_{i13}) \otimes (e_{j1} \dots e_{j13}) \;\text{ où :}$$

$\otimes$ est l'opérateur de multiplication binaire (et logique).

La composition des deux relations R_i et R_j est définie comme suit :

$$R_i \circ R_j = (e_{i1} \dots e_{i13}) \circ (e_{j1} \dots e_{j13}) = \sum_{k=1}^{13} \sum_{p=1}^{13} e_{ik} \circ e_{jp}$$

où l'opérateur $\sum$ utilise l'addition binaire (le "ou" logique).

b) Inverse d'une relation :

Grâce à l'ordre que nous avons donné à nos primitives de base, nous pouvons calculer l'inverse d'une relation disjonctive représentée par un entier de la manière suivante : Soit $R = e_1 \dots e_{13}$.

Début
  $R^{\smile} = e'_1 \dots e'_n \leftarrow 0$
  Pour i `allant de` 1 `à` 13 **faire**
    Si $e_i = 1$ **Alors**
      $e'_{13-i} \leftarrow 1$
    Fin-Si
  Fin-Pour
Fin

7.2.2 Algorithme de consistance de chemins pour le raisonnement temporel

Nous avons opté pour deux algorithmes de consistance de chemins afin d'assurer la propagation symbolique. Le premier est l'algorithme PC-1 proposé par Ladkin et Reinefeld[Ladkin 92a][Ladkin 92b] que nous avons présenté en sous section 3.3.1 de la première partie du rapport. Cet algorithme manipule directement des matrices relationnelles. La figure 7.3 illustre la procédure de calcul de la composition de deux matrices ($M \circ M$) que nous avons implantée.

Début
Pour $i = 1$ `à` N **Faire**
  Pour $j = i + 1$ `à` N **Faire**
    Début
      $M^2_{ij} \leftarrow I$
      Pour $k = 1$ `à` N `et` $M^2_{ij} \neq 0$ **Faire**
        $M^2_{ij} \leftarrow M^2_{ij} \cap M_{ik} \circ M_{kj}$
      $M^2_{ji} \leftarrow inverse(M^2_{ij})$
    Fin
  $M^2_{ii} \leftarrow E$
Fin

FIG. 7.3: *Calcul de la composition $M \circ M$.*

Le deuxième algorithme est l'algorithme PC-2 proposé par vanBeek[van Beek 96] qui représente une version modifiée de celle que nous avons présentée en sous section 3.3.1 . Nous l'avons appliqué aux matrices relationnelles de la manière suivante (cf figure 7.4).

Début
$L \leftarrow \{(i,j) | 1 \leq i < j \leq n\}$
Tant-Que $L \neq Nil$ **Faire**
  `Retirer` (i,j) `de` L
  Pour $k \leftarrow 1$ `à` n `et` $k \neq i,j$ **Faire**
    $t \leftarrow M_{ik} \cap M_{ij} \circ M_{jk}$
    Si $t = 0$ **Alors**
      `retourner Faux`
    Fin-Si
    Si $t \neq M_{ik}$ **Alors**
      $M_{ik} \leftarrow t$; $M_{ki} \leftarrow inverse(t)$; $L \leftarrow L \cup \{(i,k)\}$
    Fin-Si
    $t \leftarrow M_{kj} \cap M_{ki} \circ M_{ij}$
    Si $t = 0$ **Alors**
      `retourner Faux`
    Fin-Si
    Si $t \neq M_{kj}$ **Alors**
      $M_{kj} \leftarrow t$; $M_{jk} \leftarrow inverse(t)$; $L \leftarrow L \cup \{(k,j)\}$
    Fin-Si
  Fin-Pour
Fin

FIG. 7.4: *Algorithme PC-2*

PC-1 et PC-2 utilisent 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 composition des relations qualitatives. Ce calcul étant très coûteux, nous avons effectué une étude basée sur les propriétés des relations temporelles qualitatives afin de déterminer les cas suivants où un tel calcul peut être évité :

Soit R_1 et R_2 deux relations qualitatives.

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

Durant l'étape de recherche de solutions symboliques, nous utilisons une version incrémentale de l'algorithme PC-2. La liste L est initialisée non pas à tous les arcs du graphe mais uniquement à l'arc sur lequel nous fixons une primitive de Allen.

7.3 Propagation numérique : Consistance d'arcs dans un graphe de contraintes temporelles

Nous avons choisi les techniques de consistance d'arcs pour assurer la propagation numérique. Ces techniques sont assurées par des algorithmes de consistance d'arcs.
Soit un graphe temporel qualitatif et numérique, nous rappelons qu'un algorithme de consistance d'arcs consiste à assurer la consistance sur chaque arc du graphe. Étant donné un graphe temporel, pour chaque arc (i, j), tel que les nœuds i et j représentent deux événements donnés, un tel algorithme élimine toute occurrence du nœud i, respectivement du nœud j, qui n'est supportée par aucune occurrence du nœud j, respectivement du nœud i (i.e toute occurrence du nœud i, respectivement du nœud j, qui ne vérifie pas la contrainte qualitative attachée à l'arc (i, j) avec toutes les occurrences du nœud j, respectivement du nœud i).
Dans notre cas, pour tester la compatibilité des différentes occurrences, l'algorithme de consistance d'arcs utilise une fonction basée sur la traduction, en équations arithmétiques, des relations qualitatives attachées à chaque arc du graphe. Soit $compatible$ cette fonction d'élagage, nous la définissons comme suit :

Fonction $compatible(Rel, occur_1, occur_2)$
  Début
    Pour chaque $r_k \in Rel$ **Faire**
      Si(($en\_relation(r_k, occur_1, occur_2)$ et $\neg$ $rel\_inverse(r_k)$)
        ou ($en\_relation(r_k, occur_2, occur_1)$ et $rel\_inverse(r_k)$)) **Alors**
          retourner vrai
      Fin-Si
    FinPour
    retourner faux
  Fin

La fonction $en\_relation$ utilise la table présentée en figure 7.5 pour vérifier si deux occurrences données respectent ou non une certaine primitive de Allen. Les relations inverses (déterminées par la fonction booléenne $rel\_inverse$) sont traitées en permutant les arguments $occur_1$ et $occur_2$.

Primitive de Allen	*Equation correspondante*
$X\ P\ Y$	$debut(X) < debut(Y)$
$X\ E\ Y$	$debut(X) = debut(Y)$ $fin(X) = fin(Y)$
$X\ M\ Y$	$fin(X) = debut(Y)$
$X\ D\ Y$	$debut(X) > debut(Y)$ $fin(X) < fin(Y)$
$X\ O\ Y$	$debut(X) < debut(Y)$ $fin(X) > debut(Y)$ $fin(X) < fin(Y)$
$X\ F\ Y$	$fin(X) = fin(Y)$ $debut(X) > debut(Y)$
$X\ S\ Y$	$debut(X) = debut(Y)$ $fin(X) < fin(Y)$

FIG. 7.5: *Table de traduction des primitives de Allen en équations arithmétiques*

La figure 7.6 illustre un exemple d'application d'un algorithme de consistance d'arcs sur un graphe de contraintes temporelles qualitatives et métriques.

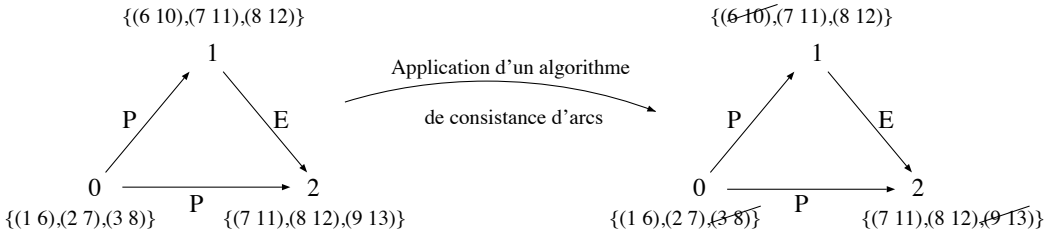


FIG. 7.6: *Consistance d'arcs dans un graphe temporel.*

Pour assurer la consistance d'arcs, nous avons opté pour l'algorithme de révision d'arcs AC-3 et les algorithmes de maintien de support AC-4 et AC-7 (voir sous section 3.2.1 pour plus

de détails sur ces algorithmes).

Nous présentons dans ce qui suit des améliorations que nous avons apporté aux 3 algorithmes de consistance d'arcs que nous utilisons.

Améliorations apportées aux algorithmes de consistance d'arcs pour la propagation numérique

1. Les contraintes temporelles qualitatives et numériques que nous manipulons possèdent certaines propriétés que nous pouvons exploiter afin d'améliorer les performances des algorithmes de consistance d'arcs que nous utilisons.
   Chaque contrainte numérique est représentée par une fenêtre temporelle définie par l'ensemble des occurrences possibles de l'événement qu'elle contraint. Cet ensemble d'occurrences est croissant par rapport aux dates de début de chaque événement. L'événement, par exemple, contraint numériquement par la fenêtre $[1, 10, 1, 4]$ a pour ensemble d'occurrences $\{(1\ 5), (3\ 7), (5\ 9)\}$. Nous pouvons, de ce fait, 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 :

   $$[P^{\smile} \in R_{ij} \wedge occ_{ik}\ P^{\smile}\ occ_{jl}] \Rightarrow \forall m > k \quad occ_{im}\ P^{\smile}\ occ_{jl} \qquad (7.1)$$

   où $occ_{ik},\ occ_{im} \in D_i$ et $occ_{jl} \in D_j$.

   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 $m > k$).

2. Toutes les listes utilisées dans les trois algorithmes de consistance d'arcs sont maintenues en ordre croissant de leurs éléments (si les éléments sont des couples représentant des arcs, par exemple, l'ordre est celui des premiers éléments de chaque couple). Ceci 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.

Étant donné que les trois algorithmes manipulent une liste contenant les arcs à réviser, nous pouvons appliquer la deuxième technique aux trois algorithmes que nous utilisons. Cependant, comme nous allons le montrer, l'application de la première technique aux algorithmes AC-4 et AC-7 est assez complexe.
Nous présentons donc dans ce qui suit, l'application de la propriété (7.1) aux trois algorithmes.

Application de la propriété (7.1) à l'algorithme AC-3

Afin d'appliquer la propriété (7.1), nous modifions la fonction $REVISE$, présentée en sous section 3.2.1, sur laquelle se base l'algorithme AC-3 pour rendre chaque arc du graphe consistant, de la manière suivante :

Fonction $REVISE(i, j)$
Début
$REVISE \leftarrow faux$
Pour `chaque occurrence` $occ_{ik} \in D_i$ **Faire**
Si $\neg compatible(R_{ij}, occ_{ik}, occ_{jl})$
Pour `tout` $occ_j \in D_j$ **Alors**
`retirer` occ_{ik} `de` D_i
$REVISE \leftarrow vrai$
Sinon
Si $P^{\smile} \in R_{ij} \ \wedge compatible(P^{\smile}, occ_{ik}, occ_{jl})$ **Alors**
`retourner` $REVISE$
`/* Ne pas tester les autres occurrences de` D_i `*/`
Fin-Si
Fin-Pour
Fin

Application de la propriété (7.1) à l'algorithme AC-4

Contrairement à l'algorithme AC-3 qui consiste à chercher un support pour chaque valeur de variable, le principe de l'algorithme AC-4 est de compter le nombre de support pour chaque valeur de variable. La propriété (7.1) augmente d'une manière significative l'efficacité de l'algorithme AC-3 étant donné que à partir du moment où l'on sait qu'une certaine valeur vérifie cette propriété on ne teste plus les autres valeurs de la variable (toutes ces variables auront alors pour support celui de la dernière valeur testée). Dans le cas de l'algorithme AC-4, si une certaine valeur vérifie la propriété (7.1), les valeurs qui restent seront toujours testées étant donné que le but dans cet algorithme est de déterminer le nombre de supports pour chaque valeur de variable. L'application d'une telle propriété n'est donc pas utile pour l'algorithme AC-4.

Application de la propriété (7.1) à l'algorithme AC-7

De même que pour l'algorithme AC-4, la mise en œuvre de la propriété (7.1) dans l'algorithme AC-7 est assez compliquée et n'améliore pas les performances de ce dernier.

Consistance d'arcs pour la recherche de solutions numériques

Afin d'appliquer les algorithmes de consistance d'arcs dans l'étape de recherche de solutions de notre algorithme général, nous avons rendu ces algorithmes incrémentaux ce qui permet de minimiser le nombre de tests effectués par ces algorithmes et améliorer par conséquent les performances de l'algorithme général. En effet, après l'étape de pre-traitement de l'algorithme général, le graphe de contraintes est rendu arc consistant. De ce fait, à chaque instanciation de variables durant l'étape de recherche de solutions, il n'est pas nécessaire d'appliquer l'algorithme de consistance d'arcs sur tous les arcs du graphe mais uniquement sur ceux qui ont pour extrémité le nœud sur lequel nous venons de fixer une valeur.

Soit i ce nœud. Pour rendre l'algorithme AC-3 incrémental, il suffit simplement d'initialiser la liste Q, contenant tous les arcs à tester, non pas à tous les arcs du graphe mais uniquement à ceux ayant pour extrémité le nœud i :

$$Q \leftarrow \{(k,i) \mid k \neq i \text{ et } (k,i) \in U\}.$$

Pour rendre l'algorithme de maintien de support AC-4 (resp AC-7) incrémental, nous procédons comme suit :

1. Après avoir rendu le graphe de contraintes arc-consistant, initialiser toutes les structures de données contenant les informations sur toutes les valeurs de variables (cette initialisation est effectuée une seule fois durant l'étape de recherche de solutions).

2. Durant l'étape de recherche de solutions, à chaque fois qu'une variable i est instanciée par une valeur a donnée, nous commençons d'abord par sauvegarder toutes les structures de données. La liste contenant les nœuds à supprimer sera ensuite initialisée comme suit :

   La liste $SUPP$ (resp $DeletionStream$) des étiquettes à supprimer de l'algorithme AC-4 (resp AC-7) est initialisée à toutes les étiquettes de la variable i excepté la valeur a :

   $$SUPP(\text{resp } DeletionStream) \leftarrow \{(i,b) \mid b \in E_i \vee b \neq a\}$$

   Après application de l'algorithme de consistance d'arcs AC-4 (resp AC-7) si le graphe ne vérifie plus la consistance d'arcs, un retour arrière est effectué, nous restaurons alors les anciennes structures de données avant de fixer une autre valeur sur la variable i.

Le fait de rendre AC-4 et AC-7 incrémentaux présente l'inconvénient suivant : on est obligé à chaque affectation d'une variable donnée de sauver toutes les structures de données. Ceci sanctionne considérablement le temps global de recherche de solutions. D'ou l'avantage de AC-3 par rapport à AC-4 et AC-7 dans l'étape de recherche de solutions.

Chapitre 8

Conclusion

Nous avons présenté dans cette partie notre modèle de représentation des informations temporelles qualitatives et numériques. L'événement étant l'objet de base que nous manipulons, l'aspect numérique du temps est représenté par une fenêtre temporelle permettant de situer cet événement par rapport à un référentiel temporel. Cette fenêtre temporelle permet de décrire toutes les occurrences possibles de l'événement (domaine de variation de l'événement constitué d'un ensemble d'intervalles numériques). L'aspect symbolique du temps est représenté par des relations qualitatives permettant chacune de situer relativement deux événements temporels entre eux.

Avec une telle représentation, nous traduisons tout problème de contraintes temporelles en un graphe où les nœuds représentent les différents événements, et où les arcs sont étiquetés par des relations qualitatives entre événements. A chaque nœud est attaché un ensemble d'intervalles représentant le domaine de variation de l'événement correspondant (fenêtre temporelle). Ainsi, nous ramenons tout problème de contraintes temporelles en un problème d'étiquetage pouvant être aisément résolu par des techniques de satisfaction de contraintes.
Nous avons présenté dans le troisième chapitre de cette partie un algorithme général de résolution de problèmes de contraintes utilisant une stratégie de recherche de solutions avec retour arrière. Des algorithmes de consistance d'arcs et de chemins sont alors utilisés pour réduire l'espace de recherche de solutions et augmenter en conséquence les performances de l'algorithme général. Parmi ces algorithmes, nous avons opté pour les algorithmes de consistance d'arcs afin d'assurer la propagation de contraintes numériques et les algorithmes de consistance de chemins pour assurer la propagation de contraintes symboliques.

Les algorithmes de consistance d'arcs que nous avons choisis sont AC-3, AC-4 et AC-7. Bien que l'algorithme AC-3 possède la complexité en temps la moins bonne au pire cas, il présente comme nous allons le montrer dans la troisième partie de ce rapport de bons résultats (comparé aux algorithmes AC-4 et AC-7) dans le cas général. En effet, cet algorithme de révision d'arcs vérifie la consistance locale du graphe sur lequel il est appliqué en manipulant simplement une liste contenant les différents arcs à vérifier. Nous pouvons donc facilement rajouter des heuristiques dont certaines sont tirées à partir de la nature du problème à traiter et qui sont dans notre cas les propriétés des contraintes temporelles. Ces heuristiques permettent d'améliorer considérablement les performances en temps d'exécution d'un tel algorithme. De

plus, AC-3 peut être facilement rendu incrémental durant l'étape de recherche de solutions ce qui améliore le temps global de la recherche en utilisant cet algorithme.
Afin de déterminer lequel de ces trois algorithmes est le mieux adapté pour la propagation de contraintes temporelles, nous comparons dans la troisième partie de ce rapport les trois algorithmes de consistance d'arcs aussi bien dans la phase de pré-traitement de l'algorithme général que dans la phase de recherche de solutions.

Nous avons choisi les algorithmes PC-1 et PC-2 pour assurer la propagation de contraintes symboliques. Alors que PC-1 utilise le calcul matriciel pour assurer la 3-consistance, PC-2 utilise une liste contenant les arcs sur lesquels nous devons appliquer la règle de 3-consistance. Nous pouvons, de ce fait, facilement rendre PC-2 incrémental lorsqu'il est appliqué à la recherche de solutions, en initialisant à chaque étape de la recherche, la liste des nœuds à vérifier à ceux qui sont en relation avec l'arc qui vient d'être changé. Une comparaison des deux algorithmes PC-1 et PC-2 est présentée dans le chapitre 10.

Dans la partie suivante de ce rapport, nous présentons en premier les différents tests de comparaison des différents algorithmes de consistance d'arcs et de chemins que nous utilisons. Cette comparaison a pour but de déterminer lequel de ces algorithmes est le mieux approprié pour l'une ou l'autre des deux phases de l'algorithme général, dans les deux types de propagation.
Nous présentons ensuite des tests de comparaison des différentes stratégies de recherche de solutions. En effet, nous avons présenté un algorithme général dans lequel nous appliquons des algorithmes de consistance locale afin de réduire l'espace de recherche, mais nous n'avons pas précisé pour le moment jusqu'a quel degré ces algorithmes seront appliqués. Le compromis étant entre le temps gagné par la réduction de l'espace de recherche et le temps perdu par ces algorithmes appliqués pour assurer cette réduction.

Troisième partie

Étude comparative des techniques de propagation de contraintes temporelles et évaluation de notre outil de raisonnement

Chapitre 9

Introduction

Nous avons présenté dans la partie précédente un algorithme général permettant de résoudre des problèmes de contraintes temporelles qualitatives et numériques. Cet algorithme se déroule en deux étapes : une étape de pre-traitement et une étape de recherche de solutions avec retour en arrière. Des algorithmes de consistance d'arcs et de chemins sont utilisés dans les deux étapes de l'algorithme général afin d'améliorer ses performances en réduisant l'espace de recherche de solutions.

Nous avons opté pour les algorithmes AC-3, AC-4 et AC-7 afin d'assurer la consistance d'arcs. Pour déterminer lequel de ces algorithmes est le plus performant du point de vue temps d'exécution, nous avons effectué des tests de comparaison de ces algorithmes de consistance d'arcs aussi bien dans la phase de pré-traitement que dans la phase de recherche de solutions.
De même, nous avons effectué des tests de comparaison des algorithmes de consistance de chemins PC-1 et PC-2 que nous avons choisis, pour déterminer lequel de ces deux algorithmes est le plus performant pour assurer la consistance de chemins dans les deux phases de l'algorithme général.

Dans notre algorithme général, nous utilisons un algorithme de consistance de chemins pour réduire le nombre de primitives de Allen par relation symbolique ce qui minimise le nombre de tests effectués par la suite par les algorithmes de consistance d'arcs qui se basent sur ces relations qualitatives pour élaguer les différentes valeurs de variables. Cependant, le temps perdu pour assurer la consistance de chemins est-il compensé par la suite lors de la recherche de solutions ?
Afin de répondre à cette question nous avons effectué des tests dans le but de déterminer si la consistance de chemins est utile pour la recherche de solutions numériques.

La stratégie de recherche de solutions que nous utilisons jusqu'a maintenant est le "*Real Full Look ahead*" (cf 3.2.3 de la première partie du rapport) c'est à dire que l'algorithme de consistance locale est appliqué, à chaque fois qu'une variable donnée est instanciée par une valeur de son domaine, sur tout le graphe de contraintes. Cependant, bien que cette méthode permette d'enlever plus d'inconsistances que les autres stratégies présentées dans la première partie de ce rapport, est-elle la meilleure stratégie de recherche de solutions ?
Pour répondre à cette question, nous avons effectué des tests de comparaison des différentes

stratégies que nous avons évoqués dans la première partie de notre rapport. Notons que ces stratégies diffèrent dans leur degré d'utilisation des algorithmes de consistance locale. Tous les tests cités ci-dessus sont présentés dans le chapitre dix.

En tirant profit des résultats de ces tests de comparaison, nous avons défini un outil de raisonnement basé sur un algorithme efficace de résolution de problèmes de contraintes temporelles. Des évaluations de performances de cet outil sont présentées dans le chapitre 11. Nous présentons également dans ce chapitre un cas particulier de problèmes de contraintes temporelles que nous avons traités. Ce sont les problèmes d'ordonnancement de tâches non préemptives.

Chapitre 10

Étude comparative des techniques de propagation de contraintes temporelles

Nous présentons dans ce chapitre une étude comparative des différentes techniques que nous avons utilisées dans notre algorithme général que nous avons présenté dans la partie précédente de notre rapport. Cette étude comparative 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 la deuxième section de ce chapitre. Ces tests sont réalisés sur une station SUN SPARC 20 ayant 64 Mo de mémoire vive. Dans ce qui suit, nous présentons une caractéristique principale des problèmes de contraintes que nous considérons pour effectuer et interpréter tous les tests présentés dans cette partie.

10.1 Caractéristique des problèmes de contraintes : Dureté de contraintes

Les problèmes CSP peuvent être caractérisés par la dureté de leur contraintes que nous définissons comme suit :

Définition

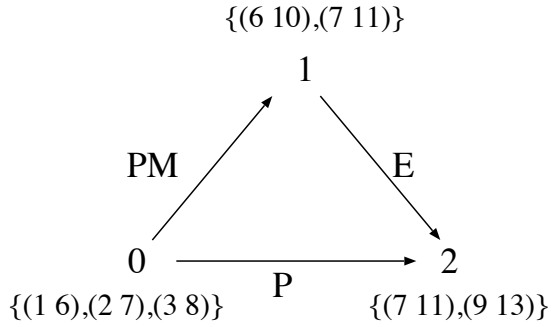
Soit une contrainte donnée entre deux variables. La dureté de cette contrainte est la fraction du nombre de couples, du produit cartésien du domaine de ces deux variables, qui ne respectent pas cette contrainte[Sabin 94].
La dureté d'un problème de contraintes est la moyenne des duretés de toutes ses contraintes.

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

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

Exemple :

La figure 10.1 décrit le calcul de la dureté d'un problème de contraintes temporelles représenté par un graphe temporel.



Soit Dureté(Cij) la dureté de la contrainte attachée a l'arc (i,j).
Nous aurons alors :
Dureté(C01) = 1/2, Dureté(C12) = 3/4, Dureté(C02) = 1/3.
Dureté du problème = (1/2 + 3/4 + 1/3)/3 = 19/36

FIG. 10.1: *Calcul de la dureté d'un problème de contraintes.*

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.

10.2 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 déterminés à partir des deux paramètres suivants :

1- l'Horizon : Date avant laquelle tous les événements doivent être réalisés, et

2- Nb-Allen : Nombre maximal de primitives d'Allen par relation symbolique.

L'algorithme de génération des problèmes consistants de contraintes se déroule comme suit :

1- 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 : $\{ (x, y) / x < y\}$). Ces N couples d'intervalles générés aléatoirement constituent la solution numérique à partir de laquelle nous construisons un ensemble de contraintes numériques et symboliques (cf figure 10.2).

2- Génération des contraintes symboliques

À partir de la solution numérique, nous déterminons la solution symbolique correspondante constituée d'un ensemble de relations de base d'Allen vérifiant la solution numérique.
À chaque atome (primitive de Allen) de la solution symbolique, nous ajoutons un nombre de primitives de Allen 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.

3- 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.

La figure 10.2 décrit les différentes étapes de génération d'un problème de contraintes temporelles symboliques et numériques.

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.

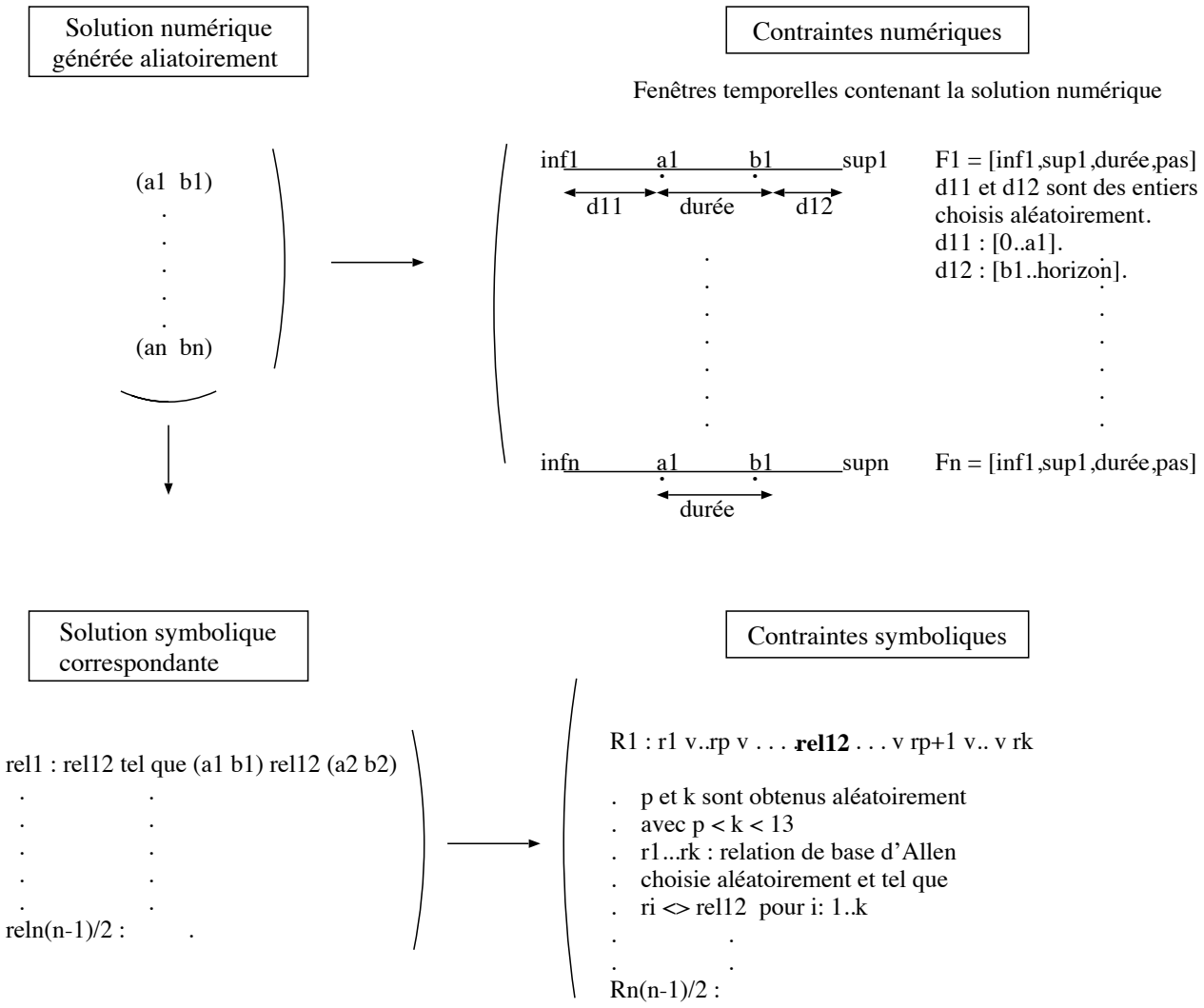


FIG. 10.2: *Description des étapes de génération d'un problème de contraintes temporelles.*

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 ce chapitre 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. Les tests sont réalisés sur une station SUN SPARC 20 ayant 64Mo de mémoire vive.

10.3 Comparaison des algorithmes de consistance d'arcs AC-3, AC-4 et AC-7

Nous présentons dans ce qui suit des tests de comparaison des algorithmes de consistance d'arcs dans les deux phases de notre algorithme général. Dans les deux types de tests, l'algorithme général que nous appliquons n'utilise pas de propagation de contraintes symboliques.

10.3.1 Comparaison de AC-3, AC-4 et AC-7 dans l'étape de filtrage

Un algorithme de consistance d'arcs est utilisé dans le but de réduire tout problème de contraintes symbolique et numérique en le rendant arc-consistant.
La figure 10.3 présente les temps CPU moyens mis par chacun des trois algorithmes pour rendre arc-consistant des problèmes de contraintes temporels générés aléatoirement et caractérisés par la dureté de leur contraintes.

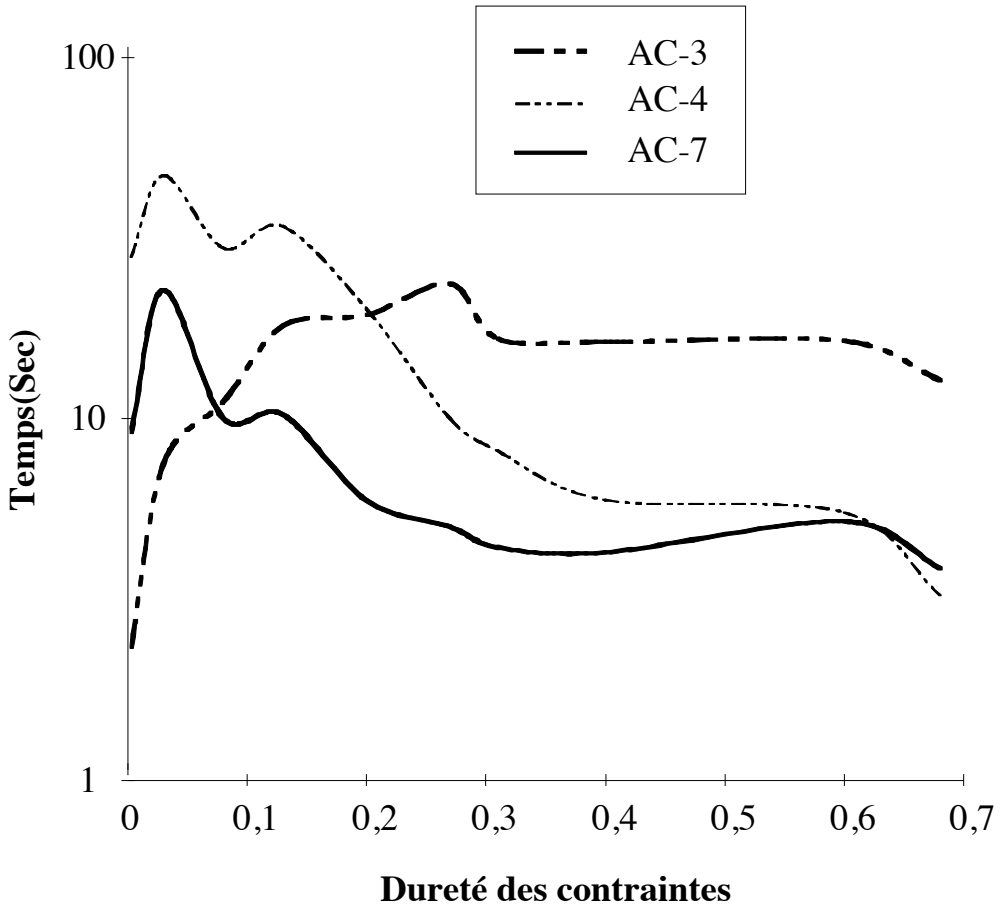


FIG. 10.3: *Performances en temps CPU mis pour assurer la consistance d'arcs.*

Dans la plupart des cas (pour la plupart des valeurs de la dureté) AC-4 et AC-7 sont plus

rapides que AC-3. En effet, grâce à l'utilisation des structures de données contenant les informations sur les différentes valeurs de variables, les algorithmes AC-4 et AC-7 effectuent beaucoup moins de tests que AC-3.
Cependant, pour les petites valeurs de la dureté, c'est à dire dans le cas où il y'a très peu de valeurs à éliminer, AC-3 termine plus vite que AC-4 et AC-7. Ceci s'explique par le fait que AC-4 et AC-7 perdent beaucoup de temps à initialiser les structures de données qu'ils utilisent alors qu'il n'y a que peu de valeurs à élaguer.
Hormis le cas des problèmes ayant des petites valeurs de la dureté, AC-7 est l'algorithme le plus performant du point de vue temps d'exécution. Ceci est du à la propriété de bidirectionalité (cf section 3.2.1 du chap 3) utilisée par cet algorithme pour éviter certains tests que les algorithmes AC-3 et AC-4 effectuent afin d'assurer la consistance d'arcs. AC-7 est donc l'algorithme que nous choisirons pour assurer la consistance d'arcs dans la première phase de notre algorithme général.

10.3.2 Comparaison de AC-3, AC-4 et AC-7 dans l'étape de recherche de solutions

Nous comparons cette fois-ci les algorithmes AC-3, AC-4 et AC-7 dans l'étape de recherche de solutions. Les tests sont effectués sur des problèmes vérifiant la consistance d'arcs (problèmes générés aléatoirement sur lesquels nous avons appliqué un algorithme de consistance d'arcs).
Les résultats de la figure 10.4 présentent les temps CPU moyens mis pour déterminer la première solution numérique aux problèmes de contraintes en utilisant l'un des trois algorithmes de consistance d'arcs. La stratégie utilisée pour la recherche de solutions est le "*Real Full Look ahead*" (l'algorithme de consistance d'arcs est utilisé à chaque instanciation de variables sur tout le graphe de contraintes).

Nous constatons, d'après ces résultats, que AC-3 est plus performant que AC-4 et AC-7. L'écart est d'autant plus grand pour les petites valeurs de la dureté c'est à dire dans le cas où il y a beaucoup d'appels aux algorithmes de consistance d'arcs durant le processus de recherche de solutions. En effet, les trois algorithmes sont appliqués d'une manière incrémentale lors de la recherche de solutions, AC-4 et AC-7 perdent alors beaucoup de temps à sauvegarder les différentes structures de données contenant les informations des valeurs de variables à chaque fois que ces algorithmes sont appelés durant le processus de recherche de solutions. Ce problème ne se pose pas pour l'algorithme AC-3 qui ne fait qu'initialiser, à chaque fois qu'il est appelé, une liste par les différents arcs à réviser. Nous constatons aussi que le temps mis par les trois algorithmes décroît lorsque les valeurs de dureté augmentent. En effet, plus la valeur de la dureté est grande et moins il y a d'appel de l'algorithme de consistance d'arcs. La phase de pré-traitement est très importante dans ce cas (la plupart du temps de l'algorithme général de recherche est perdu dans cette étape) et permet de supprimer beaucoup d'inconsistances. L'espace de recherche est alors considérablement réduit. La solution numérique est obtenue généralement après quelques appels des algorithmes de consistance d'arcs durant le processus de recherche de solutions.

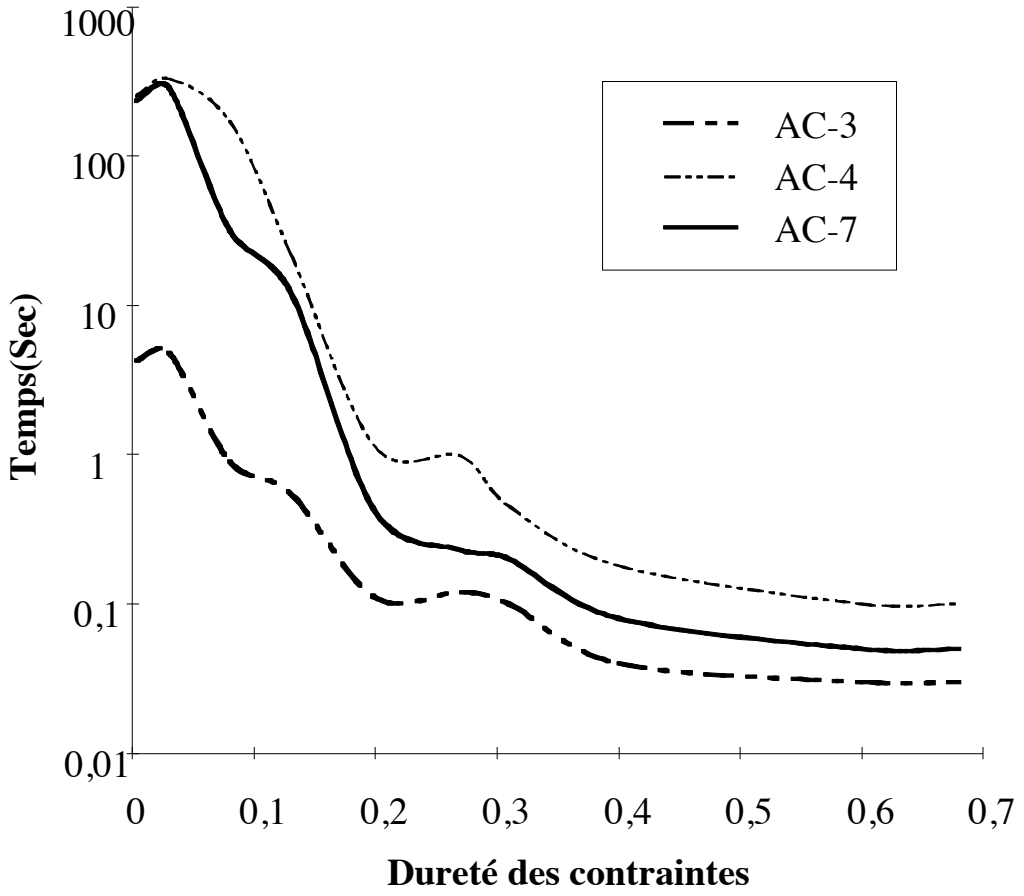


FIG. 10.4: *Performances de temps CPU mis par chacun des 3 algorithmes.*

10.4 Comparaison des algorithmes de consistance de chemins PC-1 et PC-2

Nous comparons dans ce qui suit les algorithmes de consistance de chemins PC-1 et PC-2 dans les deux étapes de l'algorithme général.
Dans la première étape de cet algorithme, PC-1 et PC-2 sont appliqués pour rendre chemin-consistant tout problème de contraintes symboliques et numériques. Cela permet de réduire le nombre de primitives de Allen par relation qualitative en enlevant certaines primitives inconsistantes (n'appartenant à aucune solutions symbolique).
Dans la deuxième étape, PC-1 et PC-2 sont appliqués pour déterminer une solution symbolique au problème de contraintes. Cette solution symbolique permet, par la suite, de déterminer la ou les solutions numériques correspondantes.

10.4.1 Comparaison de PC-1 et PC-2 dans l'étape de filtrage

Les résultats de la figure 10.6 présentent les temps CPU effectués par chacun des deux algorithmes de consistance de chemins pour rendre chemin-consistant des problèmes de contraintes

temporelles générés aléatoirement.

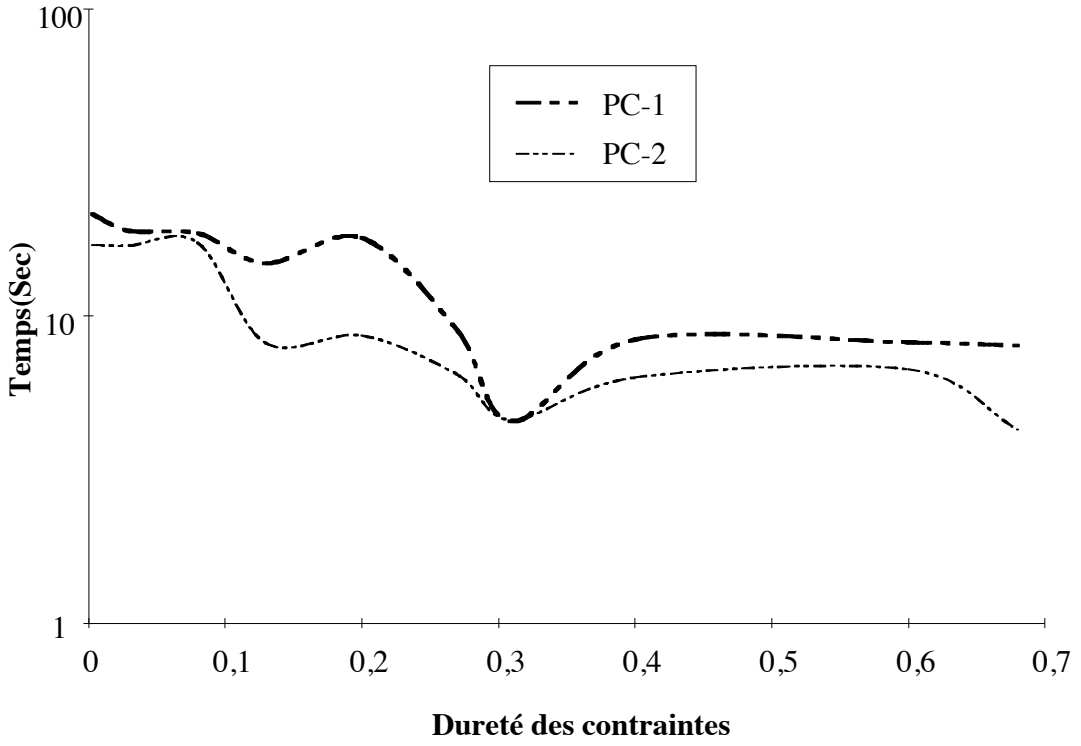


FIG. 10.5: *Performances en temps CPU mis par les algorithmes PC-1 et PC-2.*

L'algorithme PC-1 que nous utilisons est celui proposé par Ladkin et Reinefeld[Ladkin 92a]. Cet algorithme manipule des matrices relationnelles. L'algorithme PC-2 que nous appliquons utilise une liste contenant les arcs auxquels sont attachées les relations à réviser. À chaque fois qu'une relation est modifiée, seuls les arcs en relation avec l'arc contenant cette dernière sont mis dans la liste L pour être révisés (à l'inverse de PC-1 où tous les arcs du graphe doivent être dans ce cas révisés). PC-2 applique donc moins de fois la règle de 3-consistance que PC-1, il est de ce fait plus rapide que PC-1.

10.4.2 Comparaison de PC-1 et PC-2 dans l'étape de recherche de solutions

Nous avons présenté dans la deuxième partie de notre rapport (sous section 7.1) deux méthodes permettant d'assurer la deuxième phase de notre algorithme général. La première méthode utilise uniquement la propagation numérique. La deuxième méthode, quant à elle, utilise une recherche de solutions à deux niveaux : symbolique puis numérique. En appliquant un algorithme de consistance de chemins nous déterminons d'abord une solution symbolique. Nous cherchons ensuite une solution numérique pour cette solution symbolique.
Afin de déterminer lequel des deux algorithmes PC-1 et PC-2 est le plus efficace pour la recherche de solutions symboliques, nous avons effectué des tests dans lesquels nous comparons les algorithmes PC-1 et PC-2 utilisés dans la deuxième phase de notre algorithme général. Les

tests sont effectués sur des problèmes de contraintes temporelles générés aléatoirement et vérifiant la consistance de chemins (rendus chemin-consistant après application d'un algorithme de consistance de chemins). La figure 10.6 présente les temps CPU moyens mis pour déterminer la première solution symbolique en utilisant l'un des deux algorithmes de consistance de chemins.

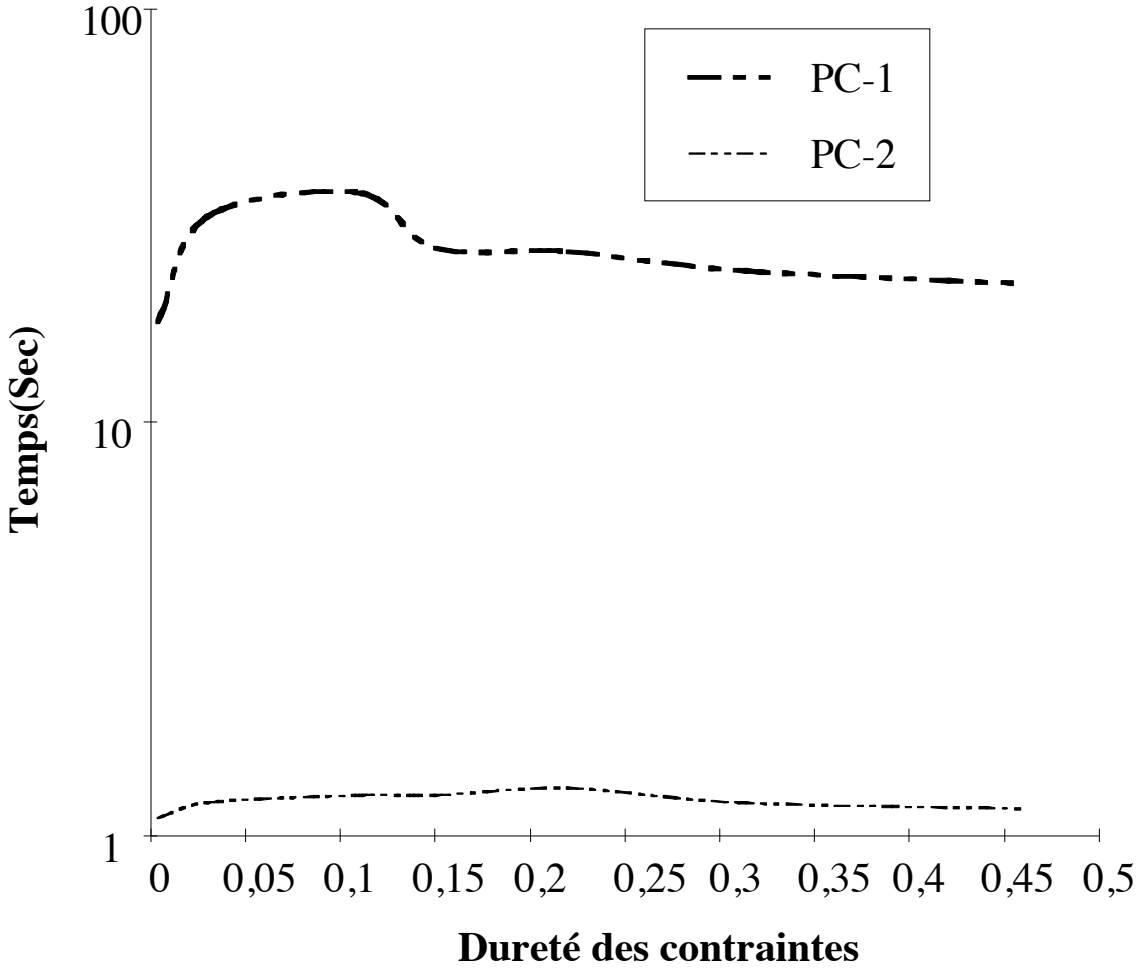


FIG. 10.6: *Performances en temps CPU mis par PC-1 et PC-2 durant l'étape de recherche de solutions symboliques.*

Nous remarquons, d'après les résultats des tests, que PC-2 est beaucoup plus rapide que PC-1. Ceci s'explique par le fait que nous avons rendu l'algorithme PC-2 incrémental ce qui permet, à chaque fois qu'une primitive de Allen est fixée sur un arc du graphe durant la recherche de solutions symboliques, de réviser non pas tous les arcs du graphe, comme cela est le cas de PC-1, mais uniquement ceux qui sont en relation avec l'arc modifié.

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

Les résultats des tests de la section précédente nous permettent d'opter pour l'algorithme PC-2 afin d'assurer la consistance de chemins dans notre algorithme général. Rappelons que PC-2 est appliqué 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 propagation numérique. Nous comparons, de ce fait, 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 10.7 présente les temps CPU moyens mis par chacune des deux méthodes pour obtenir la première solution numérique de problèmes générés aléatoirement.

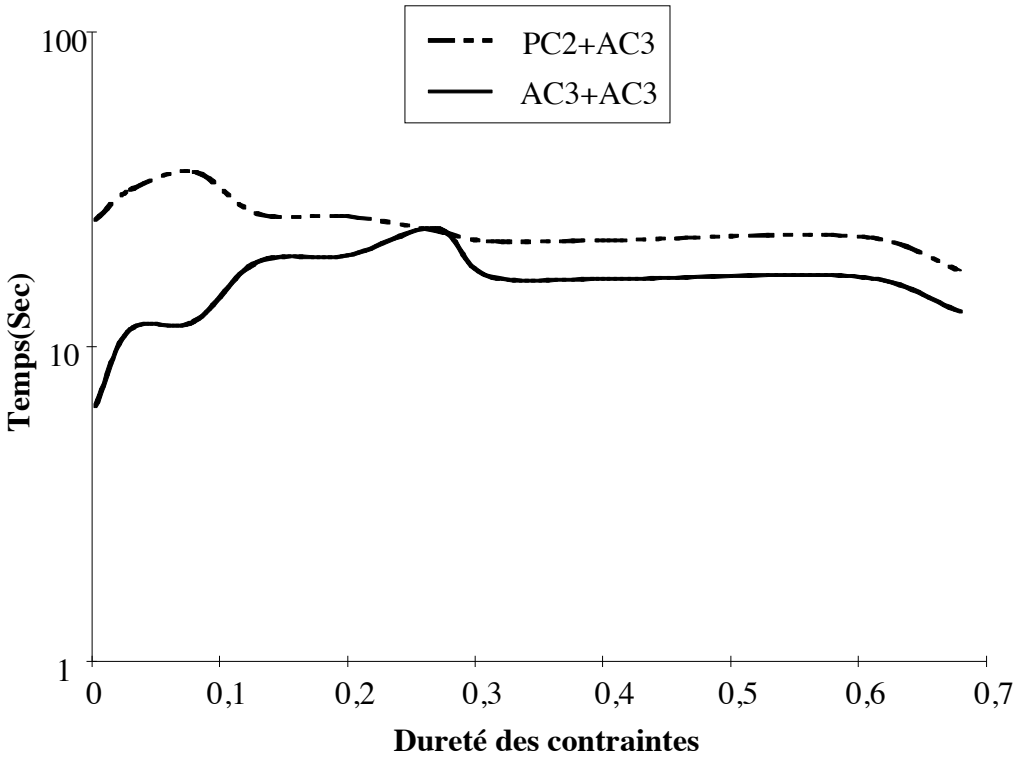


FIG. 10.7: *Performances en temps CPU mis par chacune des deux méthodes.*

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.

De ce fait, la deuxième méthode que nous avons proposé pour assurer la deuxième phase de notre algorithme général, et qui utilise une recherche de solutions à deux niveaux, est très coûteuse en temps et moins performante que la première méthode qui utilise uniquement une recherche de solutions numériques. En effet, le temps mis pour déterminer une solution symbolique nécessaire par la suite pour déterminer la solution numérique correspondante, par la méthode à deux niveaux, est supérieur à celui mis pour déterminer une solution numérique au problème en utilisant la première méthode.

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.

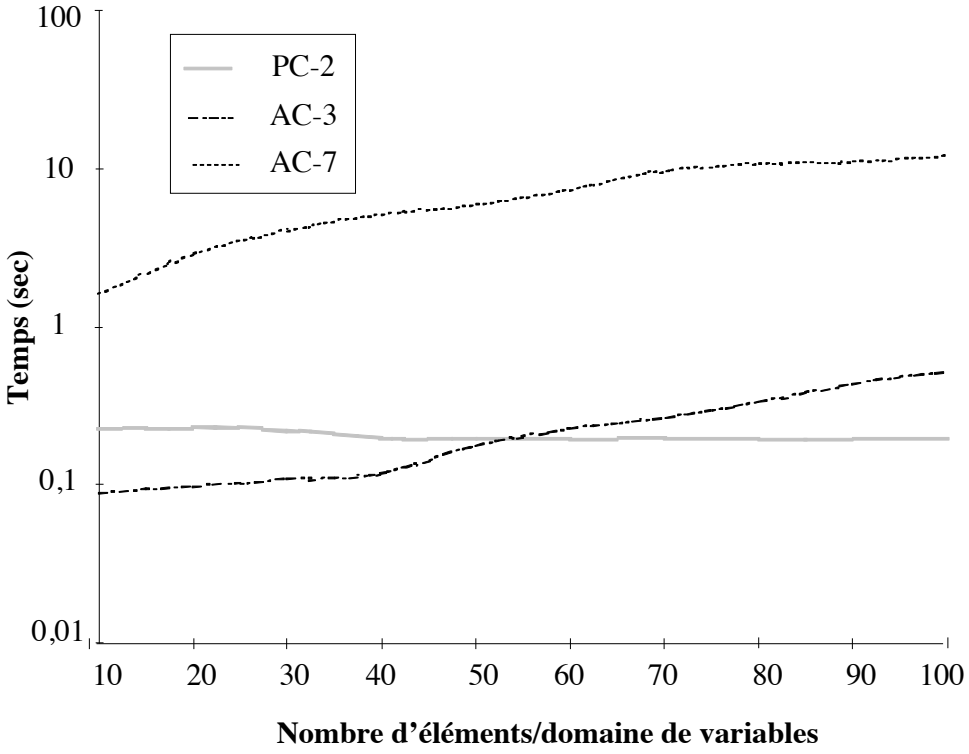


FIG. 10.8: *Performances en temps CPU mis pour détecter l'inconsistance de problèmes de contraintes.*

La figure 10.8 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.

10.6 Comparaison des stratégies de recherche de solutions

Les tests de comparaison effectués dans les sections précédentes nous ont permis d'opter pour les algorithmes AC-7 et AC-3 pour assurer la consistance locale dans notre algorithme général de la manière suivante :

– AC-7 assure la consistance d'arcs dans l'étape de pre-traitement, et

– AC-3 assure la consistance d'arcs dans l'étape 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, en plus du choix des algorithmes de consistance locales, nous devons déterminer, parmi les stratégies de recherche de solutions que nous avons citées en première partie de notre rapport, la plus rapide en temps d'exécution.
Rappelons que la différence entre ces stratégies 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.

Les différentes stratégies de recherche de solutions que nous allons présenter et comparer diffèrent dans leur degré d'utilisation des algorithmes de consistance d'arcs, AC-3 dans notre cas. Dans ce qui suit, nous présentons les 4 stratégies que nous allons comparer et donnons à chaque fois la version de l'algorithme AC-3 correspondante.

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. Nous avons, dans ce cas, implanté l'algorithme AC-3 appliqué, dans la phase effective de recherche de solutions, de la manière suivante :

Début
Soit le graphe $G = (U, E)$, i le nœud qui vient d'être instancié
  $Q \leftarrow \{(k,i) \mid (k,i) \in U \ \wedge \ k \neq i\}$
    (liste contenant tous les arcs à reviser)
  Tant-Que $Q \neq Nil$ **Faire**
    $Q \leftarrow Q - \{(k,i)\}$
    Si $REVISE(k,i)$ **Alors**
      Si $Domaine(k) = Nil$ **Alors** retourner Faux
      Sinon $Q \leftarrow Q \sqcup \{(j,k) \mid (j,k) \in U \wedge j \neq k\}$
      Fin-Si
    Fin-Si
  Fin-Tant-Que
  Retourner Vrai
Fin

La liste Q est initialisée à tous les nœuds du graphe qui sont en relation avec le nœud courant (nœud qui vient d'être instancié).

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 2 de l'algorithme AC-3 présenté ci-dessus 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, présenté ci-dessus, 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 10.9 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

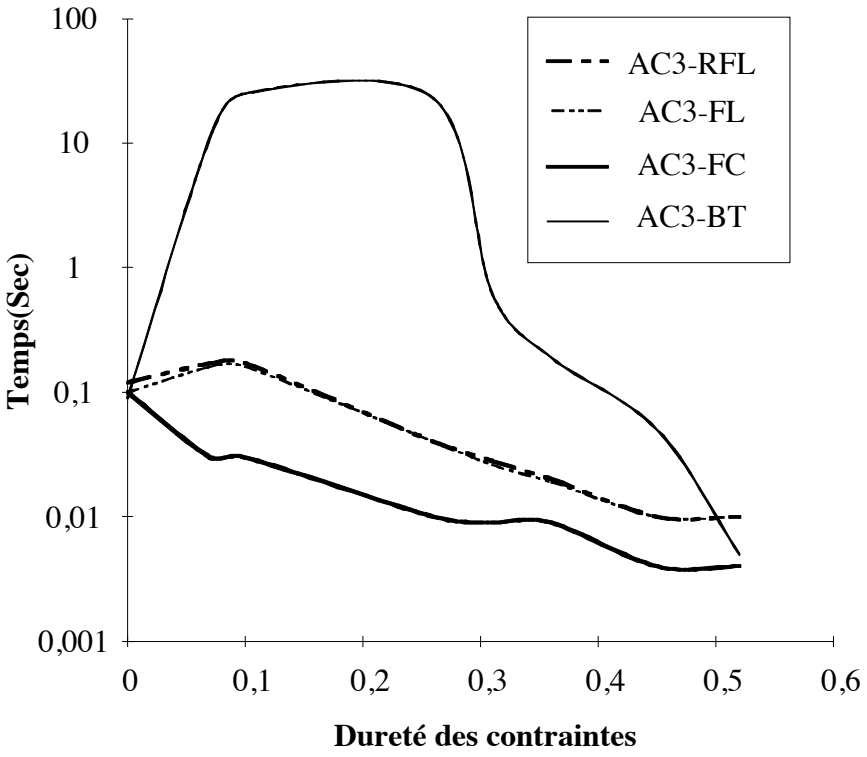


FIG. 10.9: *Performances en temps CPU mis par chacune des 4 stratégies*.

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.

Chapitre 11

Évaluation de notre outil de raisonnement

L'étude comparative du chapitre précédent nous a permis d'établir un outil efficace permettant de résoudre des problèmes de contraintes temporelles dans des délais de temps très intéressants. Cet outil de raisonnement est basé sur notre algorithme général que nous définissons, à partir des résultats de cette étude, comme suit :

- nous utilisons dans l'étape de filtrage l'algorithme AC-7,
- nous appliquons l'algorithme AC-3 durant l'étape de recherche de solutions,
- la stratégie de recherche de solutions que nous utilisons est le "*Forward Cheking*", et
- dans le cas où la propagation symbolique est utilisée, elle sera assurée par l'algorithme de consistance de chemins PC-2.

Afin de déterminer les performances, en temps, de notre outil à résoudre des problèmes de contraintes temporelles symboliques, numériques ainsi que les problèmes d'ordonnancement, nous avons effectué des tests expérimentaux d'évaluation sur des problèmes de contraintes temporelles, de nature différente, générés aléatoirement.
Tous les tests que nous présentons dans ce chapitre sont réalisés sur une station SUN SPARC 20 ayant 64 Mo de mémoire vive.

11.1 Propagation numérique

Nous présentons dans cette section les performances, en temps, de notre outil de raisonnement à traiter les informations 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 dans le chapitre précédent.
Chaque test, correspondant à un nombre de variables donné du problème, est effectué sur 100 exemples de problèmes consistants générés aléatoirement.

La figure 11.1 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.

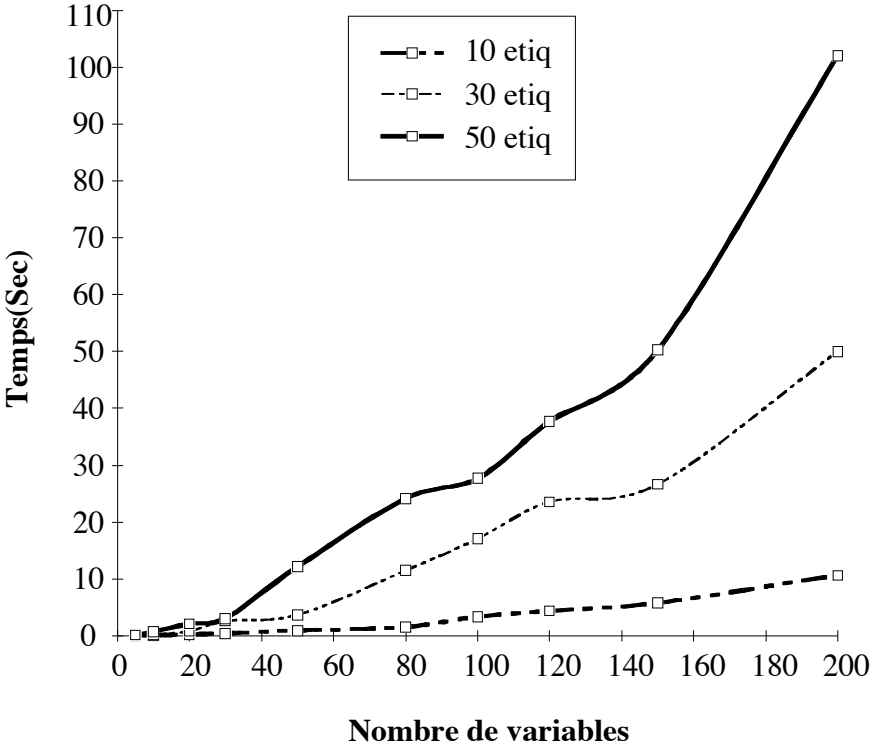


FIG. 11.1: *Temps CPU moyen pour déterminer la première solution numérique.*

Les courbes de la figure 11.1 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.

11.2 Propagation symbolique

Les tests du chapitre précédent nous ont permis de conclure que la propagation symbolique n'est souvent pas utile pour le traitement des informations numériques. Cependant, dans des applications réelles souvent l'information numérique n'est pas disponible. Nous sommes alors amenés à compléter ce manque d'information numérique par des contraintes qualitatives et à traiter par conséquent les relations symboliques.

La propagation symbolique est assurée par l'algorithme de consistance de chemins PC-2. Afin de traiter les informations symboliques dans des délais de temps acceptables, nous avons

apportés des améliorations à cet algorithme dont, notamment, la réduction du nombre de calcul de composition de relations qualitatives. Ce calcul est, comme nous l'avons précisé dans le troisième chapitre de la deuxième partie, nécessaire pour appliquer la règle de 3-consistance nécessaire à l'algorithme PC-2 pour assurer la consistance de chemins.

Pour déterminer les performances, en temps, de notre algorithme général à traiter les informations symboliques, nous avons effectué des tests de la même façon que ceux présentés dans la section précédente sauf que nous déterminons cette fois-ci le temps mis pour déterminer la première solution symbolique de problèmes de contraintes consistants générés aléatoirement. Chaque test, correspondant à un nombre de variables donné, est effectué sur 100 exemples de problèmes de contraintes symboliques consistants générés aléatoirement. Nous ne générons, en effet, que les contraintes qualitatives.

La figure 11.2 présente les temps CPU moyens mis pour déterminer, respectivement, la consistance de chemins et la première solution symbolique aux problèmes de contraintes générés aléatoirement.

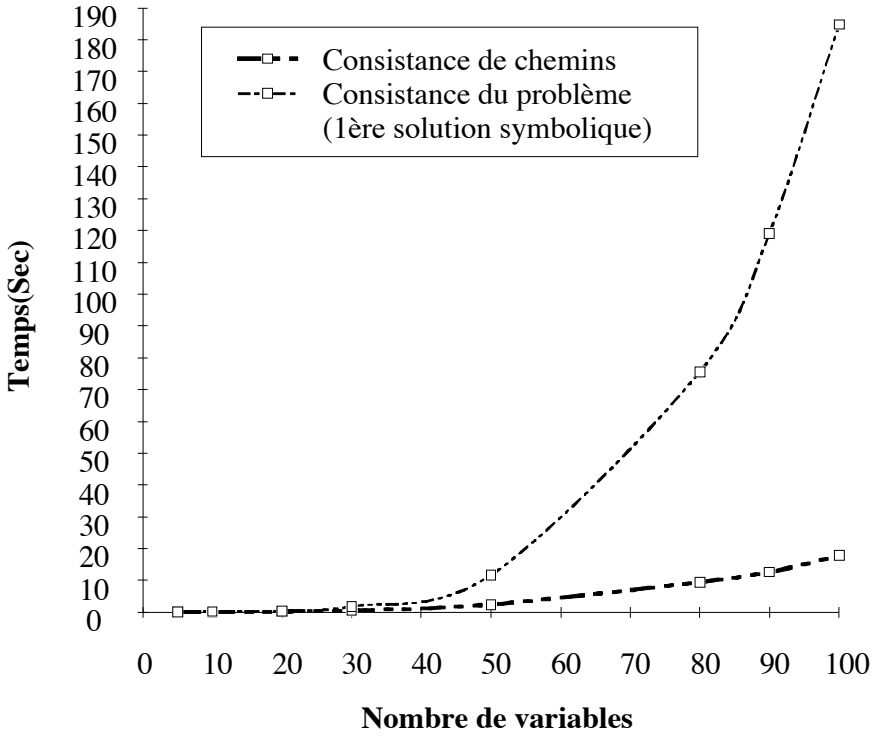


FIG. 11.2: *Performances en temps CPU pour la propagation symbolique.*

D'après la figure 11.2, nous remarquons qu'à l'inverse de la recherche de solutions numériques, la première phase consomme ici beaucoup moins de temps que la phase de recherche de solutions, surtout dans des problèmes de grande taille (ayant un grand nombre de variables). Ceci est du au fait que PC-2 est appelé plus de fois dans la phase de recherche de solutions symboliques que AC-3 dans la phase de recherche de solutions numériques. En effet, la solution symbolique est construite en fixant à chaque fois une primitive de Allen sur un arc donné du

graphe, un appel à l'algorithme PC-2 permettra alors de propager les changements sur tout le graphe de contraintes symboliques. L'appel à l'algorithme AC-3 durant l'étape de recherche de solutions numériques se fait à chaque assignation d'une variable (un nœud donné du graphe) donnée par une valeur de son domaine. De ce fait, il y a beaucoup plus d'appel à PC-2 qu'a AC-3 (nous traitons des graphes complets, le nombre d'arcs est égal dans ce cas à $n*(n-1)/2$ où n est le nombre de nœuds du graphe). Notons qu'il n'y a pas d'appel à l'algorithme PC-2, respectivement AC-3, lorsque la relation disjonctive de l'arc choisi, respectivement la variable choisie, contient une seule primitive de Allen, respectivement une valeur de domaine de la variable.

11.3 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. Le but étant de trouver un ordre qui minimise la date de fin d'exécution de la dernière tâche[Mouhoub 96c][Mouhoub 96a].
Nous représentons un problème d'ordonnancement, dans notre modèle, de la manière suivante : Chaque tâche est représentée par un événement ayant pour contrainte quantitative une fenêtre temporelle traduisant les informations temporelles numériques de la tâche, à savoir, sa durée d'exécution, sa date de début au plus tôt et sa date de fin au plus tard. Les contraintes qualitatives reliant les différents événements sont des relations disjonctives de la forme : $P \vee P^{\smile} \vee M \vee M^{\smile}$ et exprimant le fait que deux tâches ne peuvent se chevaucher (étant donné qu'elles sont effectuées par une machine mono-processeur).
Des exemples de représentations de problèmes d'ordonnancement de tâches non préemptives sont présentés dans le deuxième chapitre de la partie précédente (cf exemples 1, 2 et 3).
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* `Horizon` *la date de fin de disponibilité de la ressource (date avant laquelle toutes les tâches doivent être exécutées) et* `Duree_max` *la durée maximale d'exécution de chaque tâche du problème à générer. Pour chaque tâche nous générons*

aléatoirement :

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

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 le chapitre 7 une propriété (cf propriété 7.1 de la sous section 7.3) permettant de minimiser le nombre de tests effectué par l'algorithme AC-3 pour assurer la consistance d'arcs[10]. 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 7.1 que nous avons appliquée sur AC-3, ce dernier algorithme devient plus performant, en temps CPU, que AC-7.

Nous présentons dans la figure 11.3 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 11.3, que AC-3 auquel nous avons appliqué la propriété 7.1 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.

10. 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

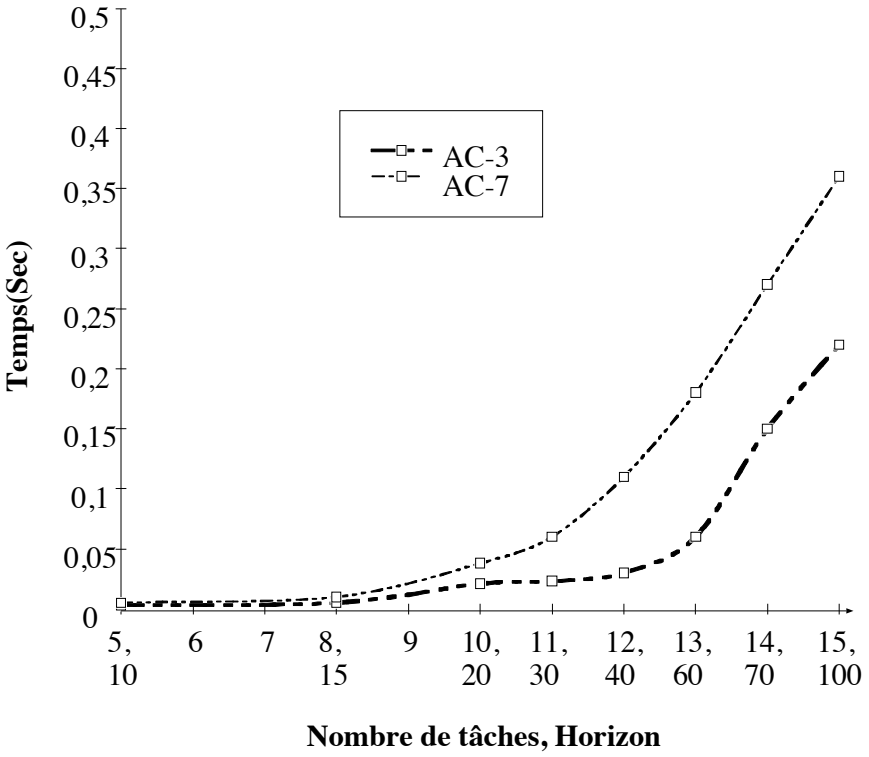


FIG. 11.3: *Performances en temps CPU pour assurer la consistance d'arcs*.

Chapitre 12

Conclusion

Nous avons proposé un algorithme général permettant de résoudre des problèmes de contraintes temporelles symboliques et numériques. Cet algorithme est divisé en deux phases : une phase de filtrage et une phase de recherche de solutions. Afin d'améliorer les performances de cet algorithme, nous avons utilisé des algorithmes de consistance locale dans les deux phases de ce dernier. Ces algorithmes de consistance locale sont de deux types : des algorithmes de consistance d'arcs permettant de traiter les contraintes numériques et des algorithmes de consistance de chemins permettant de traiter les contraintes symboliques. Nous avons opté pour les algorithmes AC-3, AC-4 et AC-7 pour assurer la consistance d'arcs, et les algorithmes PC-1 et PC-2 pour assurer la consistance de chemins.

Pour déterminer lequel de ces algorithmes de consistance d'arcs ou de chemins est le mieux adapté dans l'une ou l'autre des deux phases de l'algorithme général, nous avons effectué des tests de comparaison de ces algorithmes.
Concernant la propagation numérique, nous remarquons, d'après les tests, que AC-7 est le meilleur algorithme permettant d'assurer la consistance d'arcs dans un graphe de contraintes temporelles. En effet, grâce à la propriété de bidirectionalité, cet algorithme effectue beaucoup moins de tests que les algorithmes AC-3 et AC-4. Cependant, dans l'étape de recherche de solutions où la consistance d'arcs est appliquée à chaque assignation de variables[11], non pas sur tout le graphe mais uniquement sur les arcs en relation avec la variable assignée, nous remarquons que AC-3, utilisé d'une manière incrémentale, est l'algorithme le plus rapide dans ce cas. En effet, cet algorithme de révision d'arcs utilise simplement une liste qu'il initialise à chaque fois à tous les arcs à réviser, ce qui n'est pas le cas des algorithmes de maintien de support AC-4 et AC-7 qui doivent sauvegarder, à chaque fois qu'ils sont appelés dans la deuxième phase, toutes les structures de données contenant les informations sur les valeurs de variables ce qui est très coûteux en temps.
Nous choisissons donc l'algorithme AC-7 pour l'étape de pre-traitement et AC-3 pour l'étape de recherche de solutions.
Concernant la consistance de chemins, nous remarquons que PC-2 est plus performant que PC-1 dans les deux phases de l'algorithme général. Ceci est du au fait de l'utilisation pour PC-2 d'une liste contenant les arcs à réviser. De ce fait, à chaque fois qu'une relation est modifiée,

11. Notons que dans le cas où le domaine de la variable à assigner contient une seule valeur, l'algorithme de consistance d'arcs n'est pas appliqué

seuls les arcs adjacents à l'arc auquel est attachée cette dernière sont révisés et non pas tous les arcs du graphe. Nous optons donc pour PC-2 afin d'assurer la propagation symbolique.

Dans l'étape de recherche effective de solutions numériques, nous avons dans un premier temps opté pour la stratégie *"Real Full Look ahead"*. L'algorithme de consistance d'arcs est appliqué dans ce cas, à chaque assignation d'une variable donnée, sur tout le graphe de contraintes. Cette stratégie n'est cependant pas la meilleure. En effet, nous avons effectué des tests de comparaison des stratégies que nous avons citées dans la première partie du rapport pour déterminer la plus efficace d'entre elles en temps d'exécution.
Comme nous l'avons mentionné, ces stratégies diffèrent dans leur degré d'utilisation des algorithmes de consistance d'arcs. La stratégie du "*Backtracking*" effectue uniquement des tests entre le nœud instancié et les nœuds passés. Ceci offre l'avantage de ne pas perdre beaucoup de temps dans les tests de consistance, cependant la recherche de solutions nécessite beaucoup de retours arrière ce qui affecte les performances d'une telle stratégie. D'un autre coté, les stratégies "*Real Full Look ahead*" et "*Full Look ahead*" sont celles qui enlèvent le plus d'inconsistances ce qui minimise le nombre de retours arrière pour déterminer une solution au problème, les performances en temps de ces deux stratégies sont, par contre, affectées par le temps perdu par les tests de consistance. La stratégie "*Forward Checking*" qui représente une variante de la stratégie "*Full Look ahead*" et qui assure uniquement une partie des tests de cette dernière (les tests sont effectués entre le nœud instancié et les nœuds futurs uniquement dans le cas du "*Forward Checking*" alors qu'ils sont effectués aussi entre les nœuds futurs entre eux dans le cas du "*Full Look ahead*") présente les meilleurs résultats en temps d'exécution. C'est la stratégie que nous choisirons pour la recherche de solutions numériques.

Afin de déterminer si la propagation symbolique assurée par les algorithmes de consistance de chemins est utile pour la recherche de solutions numériques, nous avons effectués des tests qui ont permis de constater que le temps perdu par la consistance de chemins pour réduire le nombre de relations de Allen par contrainte qualitative, utilisées par la suite pour la consistance d'arcs, n'est pas compensé par le temps global de recherche de solutions. La consistance de chemins est néanmoins utile pour déterminer l'inconsistance de problèmes de contraintes.

L'étude comparative expérimentale nous a ainsi conduit à définir un algorithme de recherche de solutions numériques utilisant l'algorithme AC-7 dans l'étape de filtrage et AC-3 dans l'étape de recherche de solutions. AC-3 est appliqué dans ce cas selon la stratégie du "*Forward Checking*". La propagation symbolique n'est utilisée que si l'information numérique n'est pas disponible.

Les performances de cet algorithme sont présentées, à travers des tests expérimentaux, dans le chapitre 11. Nous avons également traité dans ce chapitre un cas particulier de problèmes de contraintes temporelles. Ce sont les problèmes d'ordonnancement de tâches. Le résultat de cette étude est le suivant.
Bien que AC-7 soit le meilleur algorithme permettant d'assurer la consistance d'arcs dans le cas général, il n'en est pas de même dans le cas d'ordonnancement de tâches. En effet, AC-3, utilisant la propriété 7.1 que nous avons défini précédemment (cf sous section 7.3), présente de meilleurs résultats que AC-7. Rappelons que cette propriété permet de minimiser le nombre

de tests de consistance entre deux variables données dans le cas où la primitive de Allen $P^{\smile}$ appartient à la relation qualitative entre ces deux variables. Dans le cas des problèmes d'ordonnancement (où la primitive $P^{\smile}$ est présente dans chaque relation qualitative) cette propriété est applicable sur tout le graphe de contraintes et permet de réduire considérablement le nombre de tests de consistance d'arcs lorsqu'elle est utilisée par l'algorithme AC-3 (nous avons précisé dans la sous section 7.3 que la propriété 7.1 ne réduit pas le nombre de tests de consistance d'arcs lorsqu'elle est appliquée à l'algorithme AC-7).

Conclusion et perspectives

Ayant comme objectif la représentation et l'interprétation de l'information temporelle sous ses deux formes, symboliques et numériques, nous avons mené dans un premier temps une étude de synthèse dans le domaine du raisonnement temporel. Cette étude, que nous avons présenté dans la première partie de notre thèse, nous a permis de définir un modèle de représentation des informations temporelles qualitatives et quantitatives. Ce modèle, que nous avons présenté dans la seconde partie de notre rapport de thèse, généralise l'algèbre d'intervalles de Allen afin d'intégrer les contraintes métriques.

Notre choix s'est porté sur la logique réifiée et plus précisément l'approche de Allen pour représenter l'aspect symbolique du temps. Cette approche offre en effet une richesse d'expressivité des informations qualitatives bien que les problèmes traitant des systèmes fondés sur cette dernière soient NP-Complets.
L'objet temporel de base que nous considérons est l'événement. Nous représentons ce dernier par le couple (ϕ, I) où ϕ est une assertion logique atemporelle et I l'intervalle de temps durant lequel ϕ est vraie.
Les informations qualitatives relient ou situent deux événements temporels donnés entre eux et sont traduites par une disjonction de relations définies par Allen entre ces événements.
Les informations quantitatives situent un événement donné dans ou par rapport à un référentiel temporel et sont représentées par des domaines de variation, appelés fenêtres temporelles, permettant de contraindre numériquement les différents événements.

Après avoir défini un modèle de représentation des informations temporelles et atteint en conséquence la première partie de notre objectif, nous nous sommes orientés sur la seconde : être capable d'exploiter ces informations et de tirer des conclusions du modèle, en répondant aux besoins suivants :

1. vérifier la consistance d'un problème de contraintes temporelles donné représenté par le modèle que nous proposons,

2. déterminer un scénario cohérent dans le cas où le problème de contraintes est consistant, et

3. offrir une exécution rapide et un temps de réponse acceptable permettant d'exploiter directement les différents résultats.

Pour ce faire, nous avons opté pour les techniques de propagation de contraintes afin de définir un outil de raisonnement à notre modèle permettant de répondre aux besoins que nous venons de citer.
Tout problème de contraintes temporelles peut en effet être codé de manière naturelle par un graphe temporel dans lequel les nœuds représentent les différents événements et les relations qualitatives entre événements sont attachées aux arcs. À chaque nœud, représentant un événement donné, est reliée une fenêtre temporelle contraignant numériquement ce dernier. Étant donné notre choix d'une représentation discrète du temps, chaque domaine de variation contient un ensemble fini d'éléments. Ceci nous permet de ramener tout problème de contraintes temporelles en un problème d'étiquetage pouvant être résolu efficacement par des techniques de satisfaction de contraintes. Ceci justifie notre choix de ces techniques que nous avons étudiées

et présentées dans la première partie de notre rapport afin de déterminer celles qui permettent de résoudre efficacement les problèmes que nous traitons.
Notre modèle, extension de l'algèbre de Allen, offre une expressivité puissante des problèmes temporels. En revanche, avec une telle représentation, nous nous attaquons aux problèmes les plus difficiles. En effet, le problème de résolution des réseaux de contraintes qualitatives exprimées par des disjonctions de relations de Allen (appelés réseaux IA) est NP-Difficile.
En tirant profit de l'étude des techniques de propagation de contraintes, nous avons défini un algorithme général de résolution de problèmes de satisfaction de contraintes temporelles. Afin de pallier les problèmes de complexité que nous venons d'évoquer, cet algorithme a été défini 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 avons proposé.

Les algorithmes de consistance locale sont appliqués dans les deux phases de l'algorithme général de la manière suivante :

- Nous utilisons les algorithmes de consistance d'arcs pour élaguer certaines valeurs inconsistantes (les valeurs qui ne vérifient pas la consistance d'arcs) appartenant aux domaines des variables de notre réseau de contraintes et assurer en conséquence la propagation numérique. La fonction d'élagage utilise la traduction en équations arithmétiques des relations qualitatives (disjonctions de primitives de Allen).
  Parmi les algorithmes de consistance d'arcs que nous avons étudiés et présentés dans la première partie de notre rapport, nous avons choisi l'algorithme de révision d'arcs AC-3 et les algorithmes de maintien de support AC-4 et AC-7.
- Nous utilisons des algorithmes de consistance de chemins qui se basent sur la règle de transitivité de Allen pour réduire les relations qualitatives en supprimant certaines primitives de Allen inconsistantes (ne vérifiant pas la consistance de chemins) et assurent par conséquent la propagation symbolique.

Étant donné que nous nous sommes intéressés à une classe non polynômiale de problèmes de satisfaction de contraintes et que nous nous sommes préoccupés de la complexité en moyenne en temps, nous avons étudié les contraintes temporelles que nous manipulons, afin de définir des propriétés que nous avons utilisées pour améliorer les performances des algorithmes de consistance d'arcs et de chemins que nous avons choisi. Ces propriétés sont les suivantes :

1. Les valeurs de variables étant ordonnées dans leurs domaines, nous avons défini une propriété permettant de minimiser le nombre de tests effectué par les algorithmes de consistance d'arcs. Nous avons modifié les algorithmes AC-3, AC-4 et AC-7 afin qu'ils puissent supporter cette propriété.

2. Les algorithmes de consistance de chemins utilisent la règle de transitivité de Allen pour réduire les relations qualitatives. L'application de la règle de transitivité nécessite l'accès à la table de transitivité de Allen. Cet accès étant très coûteux, nous avons étudié les relations qualitatives dans le but de minimiser le nombre d'accès à cette table. Cette étude nous a permis de déterminer des cas où le calcul de la transitivité peut être effectué sans un tel accès. Nous avons donc modifié les algorithmes PC-1 et PC-2 afin de supporter ces techniques ce qui a permis d'améliorer les performances de ces deux algorithmes.

3. En pratique, nous utilisons une représentation des relations qualitatives par des vecteurs de bits. Nous avons, de ce fait, défini un calcul relationnel correspondant à cette représentation. Cette représentation offre les deux avantages suivants :

   (a) Un gain en espace mémoire lors du stockage de la matrice relationnelle représentant le graphe symbolique. Ce gain est d'autant plus important lors de la recherche de solutions symboliques. Durant cette recherche en effet la matrice relationnelle, représentant un état donné du graphe, est sauvegardée à chaque fois que l'on fixe une primitive de Allen sur un arc donné du graphe. Une représentation par des vecteurs de bits permet dans ce cas de réduire considérablement l'espace mémoire nécessaire pour stocker les différents états du graphe.

   (b) Un gain en temps de calcul des différentes relations induites par transitivité lors de l'application de la règle de 3-consistance de Allen.

Nous avons également modifié les algorithmes de consistance d'arcs et de chemins afin de les appliquer d'une manière incrémentale dans la phase de recherche de solutions numériques et symboliques. Les modifications que nous avons apportées aux algorithmes de consistance locale que nous avons choisis sont les suivantes :

1. pour les algorithmes de consistance d'arcs, les listes contenant les arcs à réviser sont initialisées, à chaque assignation de variable durant l'étape de recherche de solutions numériques, non pas à tous les arcs du graphe mais à uniquement ceux en relation avec la variable instanciée. Ceci permet d'éviter de tester inutilement des arcs déjà rendus consistants,

2. pour les algorithmes de consistance de chemins, les listes contenant les arcs à réviser sont initialisées, à chaque fois que l'on fixe une primitive de Allen sur un arc donné durant la recherche de solutions symboliques, non pas à tous les arcs du graphe mais uniquement à ceux adjacents à l'arc qui vient d'être modifié.

Chaque algorithme de consistance d'arcs ou de chemins que nous avons choisi présente des avantages et des inconvénients. Afin de déterminer lequel de ces algorithmes est le plus performant dans l'une ou l'autre des deux phases de recherche de solutions, nous avons effectué une étude expérimentale permettant de comparer les performances en temps de ces algorithmes. Nous avons également comparé les stratégies de recherche de solutions, que nous avons présentées dans la première partie de notre rapport, afin de déterminer la meilleur d'entre elles.
Les résultats de cette étude sont les suivants :

1. Grâce à la propriété de bidirectionalité utilisée par l'algorithme AC-7, ce dernier est le meilleur algorithme permettant d'assurer la consistance d'arcs sur un graphe de

contraintes temporelles. C'est donc l'algorithme que nous avons choisi pour assurer la consistance d'arcs dans la phase de pre-traitement de notre algorithme général. Cependant ce n'est pas le meilleur algorithme permettant d'assurer la consistance d'arcs durant la phase de recherche de solutions. En effet, pour utiliser AC-7 d'une manière incrémentale, nous devons sauvegarder à chaque étape de la recherche de solutions (à chaque assignation de variable) toutes les structures de données contenant les informations sur toutes les valeurs de variables. Une restauration de ces structures de données est faite à chaque retour arrière. Ces opérations de sauvegarde et restauration des structures de données sont très coûteuses en temps et affectent considérablement les performances de l'algorithme AC-7. Ceci n'est pas le cas de l'algorithme AC-3 qui utilise simplement une liste qu'il initialise à chaque fois qu'il est appelé à tous les arcs à modifier. AC-3 est donc le meilleur algorithme de consistance d'arcs permettant d'assurer la deuxième phase de notre algorithme général. Notons que AC-4 souffre des mêmes inconvénients de l'algorithme AC-7.

2. L'algorithme PC-2 est meilleur que l'algorithme PC-1 pour assurer la consistance de chemins dans les deux phases de l'algorithme général. Ceci est du à l'utilisation, pour l'algorithme PC-2, d'une liste contenant les arcs qui doivent être révisés. De ce fait, à chaque fois qu'une relation donnée est modifiée, seuls les arcs adjacents à l'arc auquel est attachée la relation modifiée sont révisés et non pas tous les arcs du graphe comme c'est le cas de l'algorithme PC-1. Nous optons donc pour l'algorithme PC-2 afin d'assurer la consistance de chemins dans les deux phases de l'algorithme général.

3. Bien que la consistance de chemins est coûteuse pour déterminer la solution numérique d'un problème de contraintes, elle est efficace pour déterminer l'inconsistance de problèmes de contraintes temporelles et indispensable dans le cas où l'information numérique n'est pas disponible.

4. Bien que la stratégie "*Real Full Look ahead*" soit, parmi les stratégies que nous avons étudiées, celle qui enlève le plus d'inconsistances durant l'étape de recherche de solutions, le temps consommé par cette dernière pour supprimer ces inconsistances n'est pas compensé par le temps de recherche de solutions. La stratégie "*Forward Checking*" qui assure une partie des tests de consistance de la stratégie "*Real Full Look ahead*" présente, parmi les stratégies étudiées, les meilleurs résultats en temps.

Ces résultats nous ont ainsi permis d'améliorer les performances de notre algorithme général de traitement des problèmes de contraintes temporelles.

Nous avons également traité les problèmes d'ordonnancement de tâches qui constituent un cas particulier des problèmes de satisfaction de contraintes temporelles. Nous avons alors constaté que AC-3 est plus performant que AC-7, dans ce cas, pour assurer la consistance d'arcs. En effet, nous avons défini une propriété, sous section 7.3, permettant de minimiser le nombre de tests de consistance entre deux variables données dans le cas où la primitive de Allen $P^{\smile}$ appartient à la relation qualitative entre ces deux variables. Dans le cas des problèmes d'ordonnancement, cette propriété est applicable sur tout le graphe de contraintes et permet de réduire considérablement le nombre de tests de consistance d'arcs lorsqu'elle est utilisée

par l'algorithme AC-3. Rappelons que cette propriété s'applique difficilement à l'algorithme AC-7 et ne permet pas dans le cas de ce dernier algorithme de réduire le nombre de tests de consistance, cf sous section 7.3.

Nous avons effectué des tests de performance de notre outil de raisonnement sur des problèmes de contraintes de différentes tailles (nombre de variables et nombre de valeurs dans chaque domaine de variables). Grâce aux améliorations que nous avons apportées aux algorithmes que nous utilisons, nous constatons que les résultats sont satisfaisants pour des problèmes de moins de 200 variables. Les techniques que nous utilisons ne suffisent par contre pas dans le cas de problèmes de très grande taille pour faire face au problème de complexité. Le recours à d'autres techniques s'impose dans ce cas. Une idée serait d'utiliser des algorithmes approximatifs capables de produire une solution proche de l'optimum avec une complexité polynômiale. Ces algorithmes auront donc la qualité d'être "Anytime" c'est à dire qu'ils fournissent les résultats demandés en temps voulu.
Une nouvelle technique d'approximation fondée sur une approche connexionniste a été proposée au sein de l'équipe RFIA-CRIN & INRIA Lorraine [Gallone 96]. Les résultats de tests de cette technique pour des problèmes d'ordonnancement de tâches sont intéressants dès lors que la taille du problème est importante.

De nombreuses applications industrielles nécessitent un environnement dynamique. Un concepteur, par exemple, doit faire des restrictions (ajouter des contraintes) pour compléter la définition de son problème mais doit aussi retirer des contraintes quand il n'y a plus de solutions. Les algorithmes de consistance locale que nous utilisons peuvent facilement être rendues incrémentaux pour l'ajout de contraintes. Cependant lorsqu'il s'agit de retirer une contrainte il faut déterminer l'ensemble de valeurs qui ont été retirées à cause de cette contrainte. Les algorithmes de consistance locale que nous utilisons ne peuvent le faire. Nous devons donc utiliser des algorithmes de consistance locales dynamiques [Debruyne 95] qui consistent à conserver une justification pour chaque valeur retirée afin de pouvoir déterminer les valeurs à remettre lors d'une relaxation de contraintes (retrait de contraintes).

Bibliographie

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

[Allen 84] J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[Bacchus 91] F. Bacchus, J. Tenenberg et J. A. koomen. A non-reified temporal logic. *Artificial Intelligence*, 52:87–108, 1991.

[Barrouil 84] C. Barrouil. Contrôle de la cohérence d'un plan par la méthode du simplexe. *4ème Congrès Reconnaissance des Formes et Intelligence Artificielle*, pages 531–542, Paris, France, 1984.

[Bell 84] D. Bell et A. Tate. Using temporal constraints to restrict search in a planner. *Technical report, AiAi-TR-5, Université de Edinbourg*, 1984.

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

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

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

[Davis 81] A. L. Davis et A. Rosenfeld. Cooperating processes for low-level vision: A survey. *Artificial Intelligence*, 17:245–265, 1981.

[Dean 87] T. L Dean et D. V. McDermott. Temporal database management. *Artificial Intelligence*, 32:1–55, 1987.

[Debruyne 95] R. Debruyne. Les algorithmes d'arc-consistance dans les csp dynamiques. *Revue d'Intelligence Artificielle*, 9:239–267, 1995.

[Dechter 89] R. Dechter et I. Meiri. Experimental evaluation of preprocessing techniques in constraint-satisfaction problems. *IJCAI'89*, pages 290–296, Menlo Park, Calif, USA, 1989.

[Dechter 91] R. Dechter, I. Meiri et J. Peal. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[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.

[Estman 72] C. Estman. Preliminary report on a system for general space planning. *Communications of the ACM*, 15:76–87, 1972.

[Gallone 96] J. M. Gallone. Résolution de problèmes sous contrainte de ressources à l'aide de réseaux neuromimétiques : Application à l'ordonnancement. *Thèse de Doctorat de l'université H.P. Nancy I*, 1996.

[Gashing 78] J. Gashing. Experimental Case Studies of Backtrack versus Waltz-Type versus New Algorithms for Satisfying Assignment Problems. *Second Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 449–454, Toronto, Canada, 1978. Canadian Information Processing Society.

[Gashing 79] J. Gashing. Performance measurement and analysis of certain search algorithms. *Ph.D. diss, Dept. of Computer Science, Carnegie Mellon Univ.*, 1979.

[Gaspin 95] C. Gaspin, C. Bessière, A. Moisan et T. Shiex. Satisfaction de contraintes en biologie moléculaire. *Revue d'intelligence artificielle*, 9(3):355–381, 1995.

[Ghallab 89] M. Ghallab. Relations temporelles symboliques : Représentation et algorithmes. *Revue d'Intelligence Artificielle*, 3(3):67–115, 1989.

[Halpern 86] J. Y. Halpern et Y. Shoham. A propositional modal logic of time intervals. *In: Symp on Logic in computer science*, pages 279–292, 1986.

[Han 88] C.C. Han et C.H. Lee. Comments on Mohr and Henderson's Path Consistency Algorithm. *Artificial Intelligence*, 36:125–130, 1988.

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

[Haugh 87] B. A. Haugh. Non-standard semantics for the method of temporal arguments. *IJCAI'87*, pages 449–454, Milan, Italie, 1987.

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

[Koomen 87] A. Koomen. The timelogic temporal reasoning system in common lisp. *Technical Report TR231, Université de Rochester*, pages 241–246, 1987.

[Ladkin 92a] P.B. Ladkin et A. Reinefeld. Effective solution of qualitative interval constraint problems. *Artificial Intelligence*, 57:105–124, 1992.

[Ladkin 92b] P.B. Ladkin et A. Reinefeld. A symbolic approach in interval constraint problems. *Lecture Notes in Computer Science*, no 737, 1992.

[Lhomme 93] O. Lhomme. Consistency techniques for Numeric CSPs. *IJCAI'93*, pages 232–238, Chambery, France, 1993.

[Ligozat 90] G. Ligozat. Weak Representations of Interval Algebras. *AAAI90*, pages 715–720, Boston, MA, 1990.

[Ligozat 91] G. Ligozat. On Generalized Interval Calculi. *AAAI91*, pages 234–240, 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.

[Malik 83] J. Malik et T. O. Binford. Reasoning in time and space. *IJCAI'83*, pages 342–345, Karlsruhe, Allemagne, 1983.

[McDermott 82] D. V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[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.

[Mohr 88] R. Mohr et G. Masini. Good old discrete relaxation. *Proc. 8th European Conference on Artificial Intelligence*, pages 651–656, Munich, Allemagne, 1988.

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

[Mouhoub 96a] M. Mouhoub, F. Charpillet et J.P. Haton. Comparison and improvement of constraint propagation techniques for Interval-based temporal reasoning. *Proc of the 15th Workshop on Planning and Scheduling*, pages 264–277, Liverpool, 1996.

[Mouhoub 96b] 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.

[Mouhoub 96c] M. Mouhoub, F. Charpillet et J.P. Haton. Constraint Propagation Techniques for Scheduling Problems. *Proc of the ECAI Workshop on Intelligent Scheduling of Production Processes*, Budapest, Hongrie, 1996.

[Nebel 95] B. Nebel et H.J. Burckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's Interval Algebra. *J. ACM*, 42:43–66, 1995.

[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.

[Nudel 90] B. Nudel. Some applications of the constraint satisfaction problem. *Technical Report, CSC-90-008, Computer Science Dept, Wayne State Univ.*, 1990.

[Pape 87] C. Le Pape et S. Smith. Management of Temporal Constraints for Factory Scheduling. *Temporal Aspects in Information Systems Conference*, pages 165–176, Sophia Antipolis, France, Mai 1987.

[Reichgelt 89] H. Reichgelt. A comparison of first-order and modal logics of time. *In Logic-based Knowledge Representation. The MIT Press*, pages 143–176, 1989.

[Rit 86] J. F. Rit. Propagating temporal constraints for scheduling. *AAAI-86*, pages 383–388, Philadelphia, PA, 1986. American Association for Artificial Intelligence.

[Rit 88] J. F. Rit. Modélisation et propagation de contraintes temporelles pour la planification. *Thèse de Doctorat à l'INPG*, 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.

[Shoham 86] Y. Shoham. Chronological ignorance time, non monotonicity, necessity and causal theory. *AAAI-86*, pages 389–393, Philadelphia, PA, 1986.

[Shoham 87] Y. Shoham. Temporal Logics in AI : Semantical and Ontological considerations. *Artificial Intelligence*, 33:89–104, 1987.

[Shoham 88] Y. Shoham. Chronological Ignorance: Experiments in Non monotonic Temporal Reasoning. *Artificial Intelligence*, 36:279–331, 1988.

[Tolba 91] H. Tolba, F. Charpillet et J.P. Haton. Representing and propagating constraints in temporal reasoning. *AI communications*, 4:145–151, 1991.

[Tolba 92] H. Tolba. Contribution à l'étude du raisonnement temporel: Intégration des informations qualitatives et quantitatives et propagation de contraintes. *Thèse de Doctorat de l'université H. P. Nancy I*, 1992.

[Tsang 94] E. Tsang. *Foundation of Constraint Satisfaction*. Academic Press, 1994.

[van Beek 90a] P. van Beek. Reasoning about qualitative temporal information. *AAAI-90*, pages 297–326, Boston, MA, 1990.

[van Beek 90b] P. van Beek et R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

[van Beek 92] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.

[van Beek 96] P. van Beek et D. W. Manchak. The design and experimental analysis of algorithms for temporal reasoning. *Journal of Artificial Intelligence Research*, 4:1–18, 1996.

[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.

[Vere 83] S. A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE, Trans. Pattern Analysis and Machine Intelligence*, 5(3):246–267, Mai 1983.

[Vila 94] L. Vila. A survey on temporal reasoning in artificial intelligence. *AICOM*, 7(1):4–27, 1994.

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

[Wallace 92] R. J. Wallace et E. Freuder. Ordering heuristics for arc consistency algorithms. *Proc. Ninth Canad Conf on AI*, pages 163–169, Vancouver, Canada, 1992.

[Wallace 93] R. J. Wallace. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. *IJCAI'93*, pages 239–245, Chambery, France, 1993.

[Waltz 75] D. Waltz. Understanding line drawings of scenes with shadows. *the Psychology of Computer Vision*, pages 19–91, 1975.

[Warren 74] D. Warren. Warplan: a system for generating plans. *Memo 76, Université d'Edinbourg*, 1974.