Le temps en planification et en ordonnancement. Vers une gestion complète et efficace de contraintes hétérogènes et entachées d'incertitude

Thierry Vidal

Année 1995

Thèse

présentée au
Laboratoire d'Analyse et
d'Architecture des Systèmes du CNRS

en vue de l'obtention du
Doctorat de l'Université Paul Sabatier de Toulouse

Spécialité: *Intelligence Artificielle*

par

Thierry VIDAL
Ingénieur ENSEEIHT

Le Temps en Planification et en Ordonnancement

Vers une Gestion Complète et Efficace de Contraintes Hétérogènes et Entachées d'Incertitude

Soutenue le *13 septembre 1995* devant le jury composé de MM.

Georges **GIRALT** *(LAAS-CNRS)*. . . . . Président

Malik **GHALLAB** *(LAAS-CNRS)* . . . . . Directeur de Thèse

Jean-Paul **HATON** *(CRIN, Univ. de Nancy)* . . . . . Rapporteurs
Christian **PELLEGRINI** *(CUI, Univ. de Genève)*

Rachid **ALAMI** *(LAAS-CNRS)*. . . . . Examinateurs
Michel **CAYROL** *(UPS, Toulouse)*
Jacques **ERSCHLER** *(INSA, Toulouse)*

Rapport LAAS n° 95445

Cette thèse a été préparée au
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS
7, avenue du Colonel Roche, 31 077 Toulouse Cedex.

- Mais tu vas pleurer ! dit le petit prince.

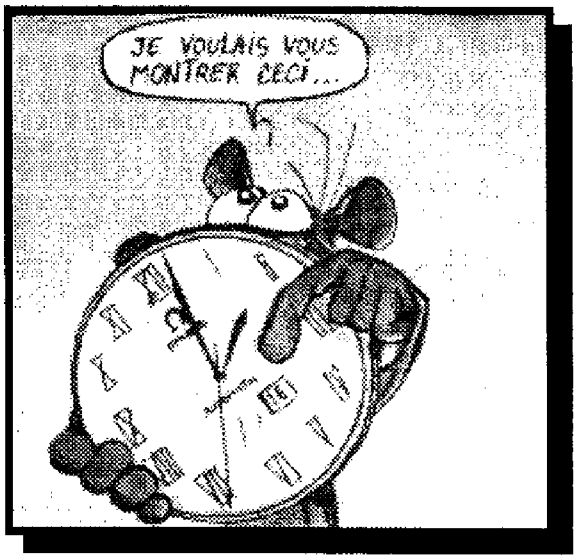
- Bien sûr, dit le renard.

- Alors, tu n'y gagnes rien !

- J'y gagne, dit le renard, à cause de la couleur du blé.

A Bruno, dont le sourire est parfois venu me redonner courage, dans ces soirées trop longues où j'oubliais de rentrer chez moi ...

Préambule



A l'heure du bilan, comment ne pas ressentir le besoin de rendre hommage à ceux qui, de près ou de loin, scientifiquement, administrativement ou humainement, ont accompagné le déroulement de cette thèse (enfin) achevée, et sont de fait pour quelque chose dans l'aboutissement du mémoire que vous venez d'entrouvrir ?

Merci donc à MM. Alain Costes et Georges Giralt pour m'avoir accueilli respectivement au sein du LAAS-CNRS et du groupe Robotique et Intelligence Artificielle. Que Georges Giralt soit également remercié pour avoir bien voulu endosser le rôle de Président de Jury lors de la soutenance.

Ensuite, je tiens à assurer MM. Christian Pellegrini et Jean-Paul Haton de ma profonde gratitude pour avoir accepté sans sourciller de se pencher sur ce mémoire afin de l'enrichir de leurs remarques et de leurs commentaires, malgré les contraintes temporelles *pour le moins serrées qui leur furent imposées. Qu'ils sachent également combien j'ai apprécié leur participation active, qu'elle soit réelle ou virtuelle, au jury de soutenance. MM. Jacques Erschler et Michel Cayrol m'ont également fait le plaisir de s'intéresser à ce travail et d'assumer avec esprit et à-propos la charge de jurés.*

Cet ouvrage n'aurait jamais vu le jour sans Malik Ghallab, qui plus qu'un directeur de thèse, a été pour moi une constante source d'inspiration sur le plan scientifique, toujours disponible et sans cesse à l'écoute. Mon seul tort fut sans doute de ne pas assez en profiter ... Sa confiance et son estime, quoique discrètes, ont fini par avoir raison de mes doutes, et ont contribué à développer et affermir une vocation d'abord hésitante, mais désormais totale et passionnée, pour la recherche scientifique.

Rachid Alami, également membre actif du jury, et Félix Ingrand, en me proposant de développer une application spécifique en complément à mes recherches, m'ont offert un cadre d'étude des plus attrayants, et l'ont accompagné par leurs remarques et suggestions constantes. Qu'ils sachent que je leur en sais gré.

Que serait un groupe de recherche sans le dévouement d'une secrétaire aussi indispensable que discrète ? Que Jacky, mais aussi Nicole, reçoivent toute ma sympathie, pour leur disponibilité, leur compétence et leur gentillesse à toute épreuve. Qualités partagés par l'ensemble des personnels administratifs du laboratoire, notamment le service documentation à qui revenait la lourde tâche d'effectuer les tirages du présent document.

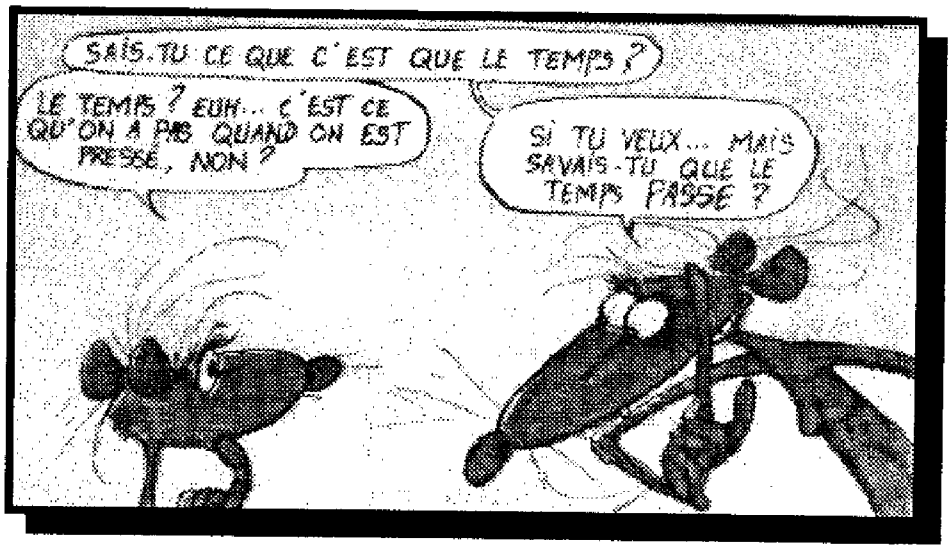
Je ne suis pas prêt d'oublier les longues discussions scientifiques, parfois tardives, toujours passionnées, avec ceux qui furent plus que des collègues ou des collaborateurs. Merci donc à Amine, Hervé, Philippe, merci aussi à Hélène, pour s'être creusés les méninges et avoir noirci en ma compagnie un nombre incalculable de tableaux blancs.

Ces années furent l'apprentissage d'une double compétence: recherche et enseignement, qui pour moi restent indissociables. Mes pensées vont donc également vers ceux avec qui j'ai travaillé dans les équipes d'enseignement de l'université, ainsi que ceux qui ont occupé avec moi les bancs des stages de formation (plus ou moins) pédagogique.

Et puis, comment ne pas mentionner cette richesse essentielle que les années de thèse nous distillent quotidiennement, même si nous n'en prenons pas toujours pleinement conscience, et qui tient dans la simple appartenance à une communauté, une "tribu" devrais-je dire, de stagiaires et de doctorants, régie par des coutumes ancestrales basées sur la chaleur humaine, l'ambiance nonchalante et décontractée, et la solidarité dans les coups durs ? J'adresse donc mes salutations fraternelles à tous les fidèles des pauses café, des petits restos, des barbecues, des apéros, des pots, et surtout des saucissons de la Montagne Noire partagés sur un coin de bureau à des heures indues; aux acharnés des pétanques, du billard, des sorties voile, des sorties montagne: Jean-Philippe, Mourad, Fawzi, Benoît, Milko, Fifi, Patricia, Vincent, Domi, Marijo, Patrick, Philipov, Flo, Xophe, Michel, Christophe, Hanna, Maher, Sieph, Alain, Sep, Bricout... auxquels s'ajoutent ceux que j'ai déjà cités, et ceux que j'ai forcément oubliés ... Parmi tous ceux-là, certains resteront plus que de sympathiques "codoctorants": de vrais amis. A quoi bon les citer en particulier: ils se seront reconnus ... Et pour finir, bienvenu aux nouveaux: nul doute qu'ils sauront perpétuer la tradition.

Je terminerai par un clin d'oeil plein de tendresse à mes parents, pour leur confiance et leur soutien indéfectibles, et puis tout simplement pour m'avoir donné toutes ces choses essentielles qui sont "invisibles pour les yeux". Aurais-je oublié quelqu'un ? Bien sûr. Mais elle sait combien ce que je lui dois ne saurait s'écrire ...

Toulouse, le 26 septembre 1995.



Introduction Générale

L'**automatisation** des tâches procède de la volonté séculaire de déléguer à une machine un ensemble d'activités pénibles ou dangereuses pour l'homme, ou plus prosaïquement d'utiliser la machine là où elle se révèle plus efficace, et donc plus rentable. Dans certains domaines tels que la robotique spatiale, ou l'intervention sur sites dangereux pour l'homme, cette volonté s'accompagne d'une double exigence: une **autonomie** quasi-complète, et une **efficacité** optimale. En d'autres termes, l'homme ne pouvant intervenir directement, le robot doit être capable, quelle que soit l'urgence de la situation, d'agir ou de réagir seul en garantissant à la fois le **succès** de la mission et sa propre **sécurité**.

Dès lors, il devient indispensable de doter ces machines d'une véritable "intelligence" embarquée, leur permettant de planifier l'ensemble des actions à mener pour réaliser la mission qui leur est confiée. L'Intelligence Artificielle, discipline scientifique à part entière dont l'objet essentiel est de modéliser la connaissance, semblait particulièrement bien armée pour s'attaquer à ces problèmes. C'est ainsi que s'est progressivement développé le domaine de la **planification de tâches**, dans lequel s'inscrit le système **IxTeT** développé dans le groupe "Robotique et Intelligence Artificielle". Pour pouvoir jouer pleinement son rôle, IxTeT devait se doter de capacités de raisonnement sur l'**action** et sur le **changement**. Qui dit raisonnement sur le changement dit raisonnement sur le **temps**. C'est donc par là qu'il a fallu commencer.

En se basant sur la logique réifiée, IxTeT a pu séparer d'un côté des mécanismes de raisonnement logique, et de l'autre des mécanismes de raisonnement purement temporel. En ce qui concerne ces derniers, il a choisi de s'appuyer sur un module indépendant de **gestion de contraintes**. Un phénomène temporel peut en effet être représenté, si l'on fait abstraction de la sémantique du problème abordé, à un ensemble d'entités élémentaire, lesquelles peuvent être

soit des instants, soit des intervalles, reliées par des contraintes. Ces contraintes peuvent être de deux types.

- Nous devons en premier lieu manipuler des contraintes **symboliques** exprimant de simples notions de précédence ou de non ubiquité entre entités temporelles. Par exemple, dans le cadre d'une application de robotique, on pourra exprimer le fait que *le robot Adam doit rejoindre sa base avant la nuit.*
- Par ailleurs, nous aimerions pouvoir manipuler des contraintes **numériques**, exprimant des notions de durée ou de date, comme par exemple *la nuit tombe à 21h* ou encore *il faut entre 20 et 30 minutes à Adam pour regagner la base*

De tels modules temporels s'appuient de ce fait le plus souvent sur des modélisations issues des Problèmes de Satisfaction de Contraintes (**CSPs**). La première version d'IxTeT [Mounir90] ne prenait en compte que les contraintes symboliques à l'intérieur d'un graphe d'instants. Il se devait d'être enrichi par une prise en compte des contraintes numériques telles que les durées des tâches ou les dates d'événements attendus, contraintes par nature **imprécises** dans nos applications. C'est ici qu'intervient le travail de recherche dont le présent mémoire se veut la relation fidèle et détaillée.

Pour bien saisir les spécifications de notre module temporel, il est nécessaire de poser clairement les interactions entre celui-ci et le planificateur. Elles se situent à deux niveaux.

- Le planificateur doit d'une part effectuer constamment des **requêtes** temporelles simples, c'est-à-dire qu'il a besoin de vérifier si tel instant est avant tel autre, ou de déterminer à quelle date est susceptible d'avoir lieu tel événement.
- Il a également besoin d'ajouter de manière incrémentale de nouvelles contraintes temporelles, tout en vérifiant la cohérence d'un tel ajout vis-à-vis des contraintes déjà présentes dans le graphe. On parle alors de **mise-à-jour** avec **maintien de cohérence**.

On peut montrer qu'en termes de **performances**, fournir rapidement des réponses aux requêtes nécessite des mises-à-jour coûteuses, et qu'inversement des mises-à-jour efficaces se contentant de vérifier la cohérence sans effectuer de propagation complète nécessitent en retour la mise en place d'algorithmes de requêtes coûteux. Au niveau symbolique, il a été possible de trouver un bon **compromis** entre les deux. Il n'en est pas de même au niveau numérique. Afin de privilégier les requêtes, dont la fréquence est plus élevée, il s'avère en fait nécessaire de maintenir le **réseau complet "minimal"**, ce qui nécessite l'utilisation d'un algorithme relativement coûteux de propagation de contraintes.

Les différences de complexité entre mises-à-jour symboliques et numériques nous ont conduit à opter pour un modèle temporel hétérogène autorisant l'utilisation d'algorithmes spéciali-

sés pour chaque type de contrainte, ce qui nécessite des mécanismes d'**interaction** efficaces et complets, notamment en termes de détection des précédences induites par les contraintes numériques. Il n'en reste pas moins que la complexité de l'algorithme de propagation numérique reste inacceptable en pratique.

Par ailleurs, il s'avère nécessaire de redéfinir le problème temporel de la planification en termes d'**incertitudes**. Dire qu'un Problème de Satisfaction de Contraintes est cohérent signifie en effet implicitement que l'on peux construire une solution en fixant librement les valeurs effectives de chaque contrainte, c'est-à-dire dans notre cas qu'il est possible de décider de la durée réelle des tâches du robot, ou des dates des événements attendus, en cours d'exécution. Dans notre cas, nous ne disposons que d'un contrôle partiel: pour certaines contraintes numériques, qualifiées de **contingentes**, la valeur (durée ou date) réelle est aléatoire, c'est-à-dire qu'elle sera observée pendant l'exécution du plan. Ceci témoigne donc d'un degré d'incertitude pesant sur les données d'entrée, qui remet totalement en cause la validité du paradigme CSP sur lequel nous nous basons.

Autant de problèmes qui vont être résolus tour à tour. Le mémoire va s'organiser de la manière suivante. Nous allons d'abord proposer au lecteur, dans le chapitre 1, une petite promenade au sein du raisonnement temporel. Après quoi nous l'inviterons au chapitre 2 à une visite guidée du domaine de la planification, le menant à la découverte du système IxTeT et de ses principales caractéristiques et spécifications en matière de gestion temporelle.

Partant de là, le chapitre 3 développera l'argument selon lequel les applications de planification témoignent d'une faible proportion d'informations numériques. Cette constatation nous conduira à confiner la propagation numérique à l'intérieur d'un sous-graphe. Cette méthode du **Sous-Graphe Numérique** sera détaillée tout au long de ce chapitre.

La suite du programme passe par la prise en compte des contraintes contingentes. De fait, nous remplaçons la notion classique de cohérence par le concept de "**contrôlabilité**" du graphe temporel, qui permet de s'assurer que l'on pourra mener à bien l'exécution du plan, quelles que soient les valeurs prises par les contraintes contingentes. La vérification de contrôlabilité est effectuée par l'intermédiaire d'un modèle temporel dual, appelé **Graphe de Décision**, sans aucune réduction des performances globales du système. Le Graphe de Décision fournit également les informations nécessaires à un contrôle optimal de l'exécution du plan. Tout ceci fera l'objet du chapitre 4.

La présentation des méthodes originales de gestion des contraintes temporelles dans le planificateur IxTeT s'arrêtera là. Nous nous permettrons cependant avant de conclure une petite escapade en direction d'un domaine voisin, à savoir l'**ordonnancement** et l'**allocation de ressources**. Le chapitre 5 présentera en effet une **application spécifique** d'ordonnancement de

tâches de manutention dans un environnement multi-robots, l'objectif étant de distribuer équitablement les tâches à l'ensemble des robots, de manière à satisfaire au mieux les contraintes temporelles de l'application. L'incertitude pesant sur les contraintes numériques nous posera là aussi des problèmes intéressants, nous obligeant à affecter les tâches à court-terme, en parallèle avec l'exécution, selon un principe "**d'horizon glissant**". L'exigence d'efficacité qui en découle est satisfaite, malgré la taille du graphe temporel, grâce à une technique de **décomposition** du graphe dépendant de l'application.

Il ne nous restera plus alors qu'à livrer dans le chapitre 6 quelques réflexions sur le travail réalisé, sous la forme de **prospectives** pouvant être envisagées à court et à moyen-terme. C'est sur une conclusion générale succincte que nous refermerons alors ce mémoire, qui nous l'espérons aura éclairé le lecteur peu familier du domaine du **raisonnement temporel basé sur les contraintes**, mais aussi suscité l'intérêt du "connaisseur" avide de suggestions et d'idées plus ou moins innovantes, propres à enrichir, pourquoi pas, sa propre réflexion.

Chapitre 1. Modéliser le Temps en Intelligence Artificielle

1. Le Temps, Référence Universelle dans un Monde en Mouvement

L'Intelligence Artificielle s'est progressivement affirmée comme une discipline scientifique à part entière vers la fin des années 60. Elle est née de la volonté d'automatiser des tâches qui jusque-là étaient réservées à l'homme, du fait qu'elles exigeaient une certaine capacité à raisonner, aussi bien sur l'environnement que sur la tâche elle-même. Pour se faire, l'Intelligence Artificielle doit s'attacher à **modéliser** les objets sur lesquels le raisonnement doit porter, lesquels varient d'un domaine d'application à l'autre. Cette notion de modèle suggère à la fois la **représentation** des connaissances manipulées et la définition de processus de **raisonnement** portant sur celles-ci.

Un grand nombre de domaines d'application ont en commun le concept d'**activité**, que l'on peut définir comme un phénomène qui évolue dans le temps, induisant des changements dans l'environnement. Cette caractéristique apparaît dans des domaines aussi divers que le diagnostic ou la planification, ou encore la supervision, ou le langage naturel. Parler de **changement** suppose implicitement que l'on se situe dans un monde dynamique. La référence au temps devient alors incontournable. Au contraire, raisonner sur un monde statique rendrait totalement superflue la prise en compte du temps.

Modéliser le changement passe généralement par la définition d'états successifs du monde, dans lesquels la dimension temporelle pourra apparaître de manière **implicite ou explicite**. Dans le premier cas, on s'attache à définir une logique globale du changement, alors que l'autre alternative aboutit à une gestion séparée des entités temporelles au sein de modules spécifiques, indépendants de la sémantique du problème abordé. Ce premier chapitre n'a pas pour but de faire un tour d'horizon complet des différents modes de traitement du temps, mais simplement de mettre en perspective les principaux courants de pensée, en focalisant progressivement sur les aspects correspondant à nos propres choix. Nous invitons le lecteur désirant une immersion plus détaillée dans le monde du raisonnement temporel à se plonger dans l'article très complet de Ll. Vila [Vila94a].

2. Une Logique qui Prenne en Compte l'Aspect Temporel

Les premiers travaux visant à prendre en compte le temps ont pris très naturellement comme point de départ les logiques classiques, en cherchant à y intégrer la notion de qualifica-

tion temporelle, c'est-à-dire le fait que les propositions manipulées étaient vraies à certains "moments". Ces tentatives de "théorisation" du temps se sont vite heurtées aux propriétés et à la structure qui lui sont propres, nécessitant notamment de vérifier la cohérence de l'ensemble des informations temporelles. Il est de fait apparu, comme nous allons le voir, que le temps n'était décidément pas un paramètre comme les autres.

2.1. Le Statut Particulier du Paramètre Temps

2.1.1. Les Logiques Classiques

La qualification temporelle peut d'abord être vue comme un paramètre supplémentaire des propositions considérées. Nous prendrons comme exemple le fait qu'un certain Albert part en vacances le 1er juillet, ce qui peut simplement s'exprimer par

départ-vacances (Albert, 1-7-95)

L'avantage d'une approche aussi simple est le cantonnement strict à la **logique du premier ordre**, et donc la possibilité d'utiliser le pouvoir calculatoire des résolveurs de théorèmes désormais bien maîtrisés par la communauté IA. Par contre, l'approche paraît manquer de souplesse dès lors que l'on veut représenter des propriétés liées à la structure même du temps, comme la relation d'ordre existant implicitement entre diverses qualifications temporelles. Comment traiter par exemple la loi générale selon laquelle les causes doivent précéder les effets ? Comment également vérifier la cohérence temporelle de l'ensemble des propositions, c'est-à-dire leur compatibilité avec la relation d'ordre évoquée ? Comment encore représenter des informations dont la qualification temporelle est indéfinie, et uniquement connue relativement à d'autres propositions, comme "*Elsa part en vacances après Albert*" ? Ou encore des durées, du style "*Albert part en vacances pendant deux semaines*" ? Comment enfin répondre à des requêtes où le temps entre en jeu, comme "*Où sera Elsa le 14 juillet ?*" ? Autant de capacités, en termes de représentation et de raisonnement, qui sont indispensables dès lors que l'on aborde des applications d'envergure. Des améliorations ont été apportées en typant les paramètres. Un type particulier donné aux paramètres temporels permet de circonvenir à certaines limitations, mais il apparaît néanmoins très délicat de définir les axiomes liés à la relation d'ordre temporel.

Signalons que cette expressivité limitée se retrouve dans l'approche classique en matière de représentation de l'évolution, appelée le "**Calcul de Situations**", dont nous reparlerons dans le prochain chapitre lorsque nous présenterons le domaine de la planification. Le Calcul de Situations permet de représenter le changement sous la forme d'un ensemble d'**états** du monde,

reliés les uns aux autres par une relation implicite d'ordre temporel. On se contente alors de spécifier pour chaque état l'ensemble des propositions qui sont vraies et celles qui sont fausses.

2.1.2. Les Logiques Modales

Dans une deuxième approche, on va ajouter des opérateurs modaux pour dire par exemple qu'une proposition "sera vrai à un moment donné dans le futur", ou qu'elle a "toujours été vraie". L'avantage est que cela permet de combiner cette représentation avec d'autres représentations de type modal, par exemple pour appréhender les notions de possibilité ou de croyance. Elle s'inscrit du même coup dans le cadre général du raisonnement sur les "mondes possibles".

Par contre, on peut d'abord remarquer que les démonstrateurs utilisés dans les logiques modales sont beaucoup moins efficaces que dans le cadre de la logique du premier ordre. Mais la principale critique est encore une fois la limitation du pouvoir expressif. Il s'agit en effet d'une approche relativiste du temps, où la notion d'antériorité trouve assez naturellement sa place. Les approches modales font de ce fait preuve d'une bonne adéquation par exemple au langage naturel. Elles sont par contre a priori moins adaptées à la manipulation de références temporelles absolues, et en particulier des informations de type date d'un événement ou durée d'une assertion. Signalons cependant à ce sujet les progrès réalisés dans le domaine des modalités indexées.

2.1.3. Les Logiques Réifiées

Les considérations précédentes donnent du temps l'image d'un paramètre en fait très original, et qu'il conviendrait de traiter spécifiquement. Pour cela, de nombreux chercheurs ont été amenés à vouloir en quelque sorte isoler le temps, l'extirper des propositions logiques dans lesquelles il restait prisonnier. La qualification temporelle accédant à un statut autonome, il devient nécessaire de rétablir d'une manière ou d'une autre le lien la rattachant à la proposition logique. Cette "**réification**" passe par la définition de "meta-propositions", dans lesquelles le temps d'une part, et l'assertion atemporelle de l'autre, deviennent de simples termes. Ces meta-propositions sont appelées prédicats temporels, le plus simple et le plus immédiat étant le prédicat HOLD, qui exprime l'assertion selon laquelle un fait P est vrai durant une portion de temps donnée I, ce qui s'écrit

HOLD $(P, I) \Leftrightarrow P$ est vrai durant I.

Par rapport à l'exemple évoqué plus haut, nous pouvons par exemple exprimer la chose suivante:

HOLD (vacances(Albert), [1-7-95, 15-8-95]),

ce qui n'est qu'un exemple, la qualification temporelle étant ici représentée par l'intermédiaire d'un intervalle temporel dont les bornes s'expriment sous la forme "jour-mois-an". Ceci constitue un choix de type ontologique parmi d'autres. Nous étudierons ces considérations plus spécifiquement dans la section suivante.

Les premiers travaux de réification du temps remontent au début des années 80. Drew McDermott [McDermott82] tout d'abord définit une théorie du temps sous la forme d'un ensemble de prédicats temporels dans lesquels la qualification temporelle repose sur des instants, alors que James Allen [Allen84] base sa théorie sur l'intervalle. Ces choix les amènent à distinguer divers types de prédicats temporels, témoignant de sémantiques diverses, que nous allons voir rapidement dans la section suivante. Ces sémantiques ont cependant été critiquées pour leur manque de rigueur formelle et leur dépendance trop étroite vis-à-vis des choix ontologiques. C'est pourquoi Yoav Shoham est souvent considéré comme l'artisan majeur de la réification du temps [Shoham87], car bien que ses travaux soient postérieurs, il est le premier à avoir défini une sémantique claire du temps, basée sur un seul prédicat de réification, et dans laquelle la notion d'instant et d'intervalle pouvaient être indifféremment supportées.

Nous n'entrerons pas dans de plus amples considérations, lesquelles se situeraient hors du contexte de ce mémoire. Signalons simplement que des travaux postérieurs ont à leur tour critiqué le travail de Shoham, proposant à la place une approche plus rigoureuse d'un point de vue théorique, et plus efficace aussi car profitant des avantages de la logique du premier ordre. Il s'agit de partir des logiques classiques typées, en désignant les paramètres temporels comme étant du type *tokens*, sur lesquels des opérations de quantification peuvent être facilement appliquées, autorisant l'écriture de manière immédiate des règles générales de description des phénomènes temporels, tel que *"les causes précèdent les effets"*. Nous renvoyons le lecteur au modèle développé par Vila et Reichgelt [Vila93a], qui, quoiqu'issu des approches classiques, est relié au concept de logique réifiée. L'exemple précédent s'écrirait ici

HOLD (vacances(Albert, token$_i$))

où *token$_i$* peut être remplacé par une valeur particulière, par exemple [1-7-95, 15-8-95], ou bien désigner une variable, permettant d'exprimer par exemple le fait que chaque fois qu'Albert est en vacances, il prend dix kilos ...

Nous avons pu constater le lien étroit existant entre la notion de causalité et la prise en compte du temps. Nous allons voir cela plus en détail dans ce qui suit, et constater qu'il peut être avantageux de relâcher ce lien, isolant les entités temporelles de leur cadre sémantique.

2.2. Temps et Causalité: Amis ou Ennemis ?

2.2.1. Définir une Logique Temporelle

Les travaux précédemment cités [McDermott82, Allen84, Shoham87] ont été initialement motivés par la volonté de définir de véritables théories du temps, permettant de représenter avant tout le changement, c'est-à-dire en fait les relations de cause à effet entre divers phénomènes temporels. De telles théories permettent de s'attaquer aux problèmes de prédiction (comme en planification) aussi bien que d'explication (en diagnostic par exemple). Il s'agit aussi de décrire des propriétés telles que

- la **contradiction** inhérente entre deux propositions, ce qui nécessite que leurs diverses qualifications temporelles soient distinctes,
- la **persistance** d'un fait, c'est-à-dire que le fait est censé perdurer tant que rien ne vient le contredire.

A partir de là, il devient nécessaire de mettre en place des capacités de **raisonnement non-monotone**. En d'autres termes, on veut pouvoir ajouter aussi bien que retirer des propositions temporellement qualifiées dans notre base de connaissances, construisant par là-même une "chaîne de causalité" décrivant le processus dynamique abordé. Par exemple, considérons un fait A causant l'apparition d'un effet B, lequel persiste jusqu'à l'apparition à l'instant i d'un phénomène C. B et C étant contradictoires, B devient donc faux à l'instant i (ce qui constitue ce que [Hertzberg91] appelle une limitation de la persistance). Si maintenant je désire retirer C, je dois alors **rétablir** le fait que B reste vrai au-delà de l'instant i, ce qui est loin d'être trivial. Nous y reviendrons à la fin de ce chapitre, dans le cadre des TMMs.

Le Calcul d'Evénement

Le "**Calcul d'Evénements**" [Kowalski86] propose un formalisme bien adapté à la représentation de la causalité. En effet, en opposition au Calcul de Situations, il considère au lieu de l'état l'événement comme objet principal du raisonnement. C'est-à-dire qu'au lieu de considérer le changement sous un angle global, on cherche plutôt à représenter des changements locaux. Ceci permet d'éviter le problème de **rémanence** (plus connu sous le terme anglophone de "frame problem"), essentiellement lié au Calcul de Situations, c'est-à-dire la nécessité de représenter a priori, non seulement ce qui a changé, mais aussi TOUT ce qui est resté vrai lors d'un changement d'état.

Un événement représente donc la position temporelle **instantanée** à laquelle une proposition donnée change d'état. Des prédicats permettent alors de (1) définir la sémantique de chaque événement, et (2) relier chaque événement aux propositions qu'il affecte, comme par exemple *commence(E,P)* [resp. *termine(E,P)*] pour dire que l'événement E rend vraie [resp.

fausse] la proposition *P*. Ces prédicats font de suite penser à ceux de la logique réifiée. Cela étant, cette technique se place très à l'écart des méthodes de réification, la dimension atemporelle et la dimension temporelle restant ici étroitement imbriquées dans une représentation globale unifiée.

L'avantage principal est que le concept de persistance est ici naturel, et qu'il est possible de représenter la contradiction entre propositions ou les liens de causalités par l'intermédiaire de nouveaux prédicats (voir les extensions proposées par [Borillo90]). L'événement permet par contre de représenter essentiellement des positions temporelles relatives. Il est néanmoins possible d'incorporer dans le modèle des références temporelles absolues, pour exprimer des dates ou des durées, en complétant les axiomes du Calcul d'Evénement [Bernard91, Vila94b].

Les approches basées sur la logique réifiée

Dans les approches déjà citées [McDermott82, Allen84], les divers auteurs ont également cherché à attaquer les problèmes de contradiction, de persistance et de causalité, dans une vision globale du temps et du changement. Des prédicats temporels ont été proposés, tels que

ECAUSE ($event_1$, $time_1$, $event_2$, $time_2$) [Allen84],

pour définir la relation de cause à effet entre deux "événements"[1]. Une manière assez sympathique et intuitive de voir les choses est de qualifier de telles approches de représentations temporelles "**haut-niveau**", comme dans [Rit88]. Bien que celles-ci posent clairement la relation existant entre le temps et la causalité, la complexité de la tâche à mener a conduit leurs auteurs à un même constat: avant de mettre en place une théorie globale du temps, il convient de proposer des mécanismes plus élémentaires pour maintenir un ensemble d'informations temporelles, à l'intérieur de modèles excluant toute dimension sémantique, ce que nous allons voir au paragraphe suivant. [Rit88] parle alors de représentations temporelles "**bas-niveau**".

2.2.2. Une Gestion Séparée des Connaissances Temporelles

L'approche "bas-niveau" consiste à profiter de la réification pour découpler totalement le temps et la dimension logique. On peut voir l'ensemble des propositions temporelles comme une table à deux dimensions, où le temps figure en abscisse, et l'ensemble des propositions en ordonnée. Pour chaque proposition *P*, nous disposons donc dans cette table de l'ensemble des qualifications temporelles *I* telles que le couple <*P*, *I*> est une **occurrence** temporelle de *P*. Il est alors possible de faire abstraction de la dimension sémantique (c'est-à-dire des propositions logiques) et de gérer l'ensemble des qualifications *I* dans un module séparé de **gestion de con-**

1. Notons que pour James Allen, un événement est l'occurrence d'une proposition temporelle, c'est-à-dire qu'il est défini sur un intervalle, alors que dans la suite un événement sera par définition instantané, comme nous allons le voir dans la section suivante

traintes temporelles. Celui-ci est utilisé comme sous-système du système global de résolution de problème (planificateur ou système de diagnostic, par exemple).

Du même coup, nous assistons à un "partage du travail", le système global prenant en charge tout ce qui relève de la déduction logique, y compris les relations de causalité, le gestionnaire de contraintes temporelles n'étant plus chargé que des deux tâches fondamentales que sont

- vérifier la **cohérence** de la base temporelle, soit a posteriori, soit à chaque ajout,
- et répondre à des **requêtes** émanant du système global, concernant par exemple la position relative courante entre deux primitives temporelles.

Il n'est pas nécessaire de faire durer plus longtemps le suspense: il s'agit là de l'approche que nous avons choisi d'adopter dans le planificateur IxTeT, dont nous allons poser les bases théoriques dans le prochain chapitre. Signalons simplement rapidement que la dénomination IxTeT signifie à l'origine "Indexed Time Table", en référence à la notion de table à deux dimensions qui vient d'être évoquée.

La ligne de démarcation entre dimensions temporelles et sémantiques n'est cependant pas si clairement définie qu'il y paraît. En effet, concernant les problèmes de contradiction et de persistance, nous verrons dans le chapitre 2 que le choix reste ouvert entre un traitement au niveau logique ou au niveau temporel. Nous exposerons dans ce chapitre uniquement les fonctionnalités de base d'un module temporel. Néanmoins, avant de parler plus en détail de ces gestionnaires de contraintes temporelles, il convient de poser clairement ce qui n'a pour le moment été évoqué que de manière informelle, à savoir le choix des primitives et des relations temporelles à manipuler, qui conditionnent fortement aussi bien l'expressivité du module temporel que les capacités de raisonnement qu'il convient de lui adjoindre.

3. La Dimension Ontologique: Eléments de Base

Ontologie signifie "théorie métaphysique de l'être en tant qu'être". Initialement utilisé pour qualifier la philosophie d'Aristote, ce terme se rattache plus prosaïquement dans les sciences modernes à la description des entités physiques manipulées et à leurs propriétés intrinsèques. Définir l'ontologie d'un système consiste à définir les divers concepts manipulés, ainsi que leurs propriétés, leur classification, et les capacités de raisonnement que l'on souhaite pouvoir leur appliquer. Dans le cas d'un système temporel, l'ontologie est au service de la description d'une structure qui soit en conformité avec les propriétés du temps. Celles-ci peuvent être postulées différemment, selon le cadre d'application dans lequel on se situe. On peut en effet voir

le temps comme une structure

- dotée d'un ordre *partiel* ou *total*,
- *bornée* ou *infinie*,
- *discrète* ou *continue* (on dit aussi *dense*),

Ces choix dépendent le plus souvent du domaine d'application. La plupart des systèmes actuels considèrent les primitives temporelles comme étant **partiellement ordonnées**, autorisant du même coup une représentation du parallélisme entre phénomènes temporels. Pour une tâche de prédiction, le temps est considéré comme étant **borné à gauche** (c'est-à-dire que l'on se donne une origine des temps), et a priori infini à droite. Dans une tâche d'explication, cela pourra être le contraire. Néanmoins, il est toujours possible pour une application donnée de se fixer des bornes arbitraires si cela s'avère nécessaire (par exemple, fixer la naissance de Jésus Christ comme origine des temps n'est pas très contraignant et permet de représenter du même coup des dates absolues sous la forme "jour-mois-année"). Pour ce qui est du choix entre temps discret et continu, nous y reviendrons plus loin.

3.1. L'Entité: Vers une Sémantique des Phénomènes Temporels

La notion d'entité est une notion ontologique, dont il convient de ce fait de toucher quelques mots. Nous ne nous y attarderons pas cependant car elle se situe à un niveau sémantique, et donc hors du cadre des représentations bas-niveau qui nous intéressent plus particulièrement.

Il s'agit en fait de spécifier la sémantique de l'affirmation "la proposition P est vraie sur l'unité de temps I". Drew McDermott [McDermott82] distingue

- un **fait**: une proposition vraie de manière **homogène** sur un intervalle, et donc identiquement vraie sur tout sous-intervalle, comme par exemple "*Albert est en vacances tout le mois de Juillet*" (il est donc en vacances par exemple le 14 juillet),
- un **événement**: quelque chose qui "s'accomplit" sur toute la durée de l'intervalle, et ne peut donc être subdivisé, comme par exemple "*Albert a fait 1000 kms pendant le mois de Juillet*",

Ces définitions, bien qu'intuitivement correctes, ont été abondamment critiquées sur le plan formel. James Allen [Allen84] propose quant à lui de distinguer trois types d'assertions. Aux **propriétés** et **événements** (similaires aux notions de McDermott), il ajoute un troisième type d'assertion, le **processus**, censé être vrai sur "au moins un nombre important de sous-intervalles", comme par exemple "*Albert est parti en vacances durant l'été*", qui suppose qu'il y ait eu éventuellement plusieurs périodes de congé distinctes. Ces définitions s'avérant dans certains exemples ambiguës, Yoav Shoham [Shoham87] propose d'étendre à un ensemble, qu'il quali-

fie d'exhaustif, de 6 entités distinctes, lesquelles ont été à leur tour remises en question dans des travaux postérieurs (voir pour plus de détails [Vila94a]) ...

3.2. La Primitive Temporelle, Brique Elémentaire de la Représentation

Nous avons parlé jusqu'à présent de manière très informelle d'intervalles et d'instants pour qualifier temporellement une proposition. Le choix de l'un ou de l'autre est cependant au centre de la spécification des systèmes temporels, qu'ils soient haut-niveau ou bas-niveau, puisque ce n'est qu'à partir d'une entité de base clairement définie qu'il est possible de construire une structure temporelle.

3.2.1. L'Intervalle: une Représentation Naturelle de l' Activité

Dans une première vision des choses, il apparaît qu'une proposition est généralement vraie sur une portion de temps, parfois dénommée période. Ceci correspond bien à la notion d'activité, qu'il s'agisse de processus dans le domaine de la supervision, ou de tâches dans les domaines de la planification et de l'ordonnancement.

C'est ce qui a motivé le choix d'une ontologie basée sur l'intervalle dans les travaux de J.Allen et de ses successeurs. Chaque proposition étant liée à un intervalle, la base temporelle sera constituée d'un ensemble d'intervalles dont il convient de définir le positionnement temporel relatif, comme nous allons le voir un peu plus loin.

3.2.2. L'Instant: une Représentation Naturelle du Changement

Par contre, le changement est par nature instantané, ce pourquoi McDermott a préféré s'en référer à l'instant comme brique de base de sa théorie. Un instant se définit alors comme le lieu d'un changement d'état d'une proposition. Cette approche est en fait la suite logique de la philosophie adoptée dans le Calcul de Situations, où la dynamique s'exprime sous la forme d'une succession d'états instantanés, mais est aussi conforme à l'esprit du Calcul d'Evénements, puisqu'un événement, comme nous l'avons vu, est par nature également instantané. Cela n'a rien d'étonnant, puisque ces deux techniques relèvent toutes deux d'une volonté de modéliser le changement.

Cela étant, activité et changement sont étroitement liés, et les deux points de vue, instant ou intervalle, sont a priori tous deux défendables. L'opposition atavique qui s'est développée

entre ces deux écoles a porté sur deux aspects. D'abord, le compromis entre efficacité et expressivité en termes de relations temporelles, conduisant à des études de complétude des algorithmes. Nous verrons cela un peu plus loin. Mais un premier élément de comparaison peut être exposé dès à présent, car il concerne l'expressivité de la primitive en tant que telle.

3.2.3. Elément de Comparaison: le Problème de l'Instant Divisé

Dans [Shoham87], on peut voir déjà que si l'approche par intervalles est considérée comme effectivement la plus naturelle, il serait également souhaitable de pouvoir représenter un phénomène instantané. On peut se demander pourquoi Allen refuse d'intégrer l'instant dans son formalisme en tant que simple extrémité d'un intervalle. En fait, c'est pour des raisons de sémantique, liées à ce que l'on appelle le "Problème de l'Instant Divisé" (PID) [vanBenthem83]. En deux mots, si P est vraie sur l'intervalle I et fausse sur I' tels que I' succède immédiatement à I, alors qu'en est-il de P à l'instant séparant exactement I et I' (appelé instant divisé) ? Cela revient à se demander si l'extrémité d'un intervalle appartient ou non à cet intervalle. Trois choix distincts peuvent être adoptées:

1. soit on répond OUI, et alors, au niveau de l'instant divisé, P est à la fois vraie et fausse, ce qui est incohérent,
2. soit on répond avec Allen NON, et alors cela traduit une indétermination,
3. soit on choisit de voir les intervalles comme étant semi-bornés, à droite ou à gauche, ce qui rétablit la cohérence mais peut être considéré comme artificiel [Vila93b].

On peut aussi éliminer purement et simplement le problème en **discrétisant** le temps. Dire que le temps est discret suppose que l'on peut déterminer un couple d'instants tels qu'il n'y ait aucun autre instant entre les deux. Dans le cas des intervalles, on pourra alors définir un intervalle comme étant un ensemble fini d'instants. On peut alors considérer que l'instant fictif situé à la rencontre de deux intervalles échappe à la discrétisation, et n'existe donc tout simplement pas. Le temps discret peut néanmoins apparaître comme moins expressif que le temps continu. Dans [Barber93] (nous reviendrons sur ces travaux à la fin du chapitre 3), on peut trouver une discussion intéressante sur le fait que la discrétisation peut être ajustée à tout moment, simplement en modifiant le niveau de **granularité**. On retrouve alors le pouvoir expressif du temps continu. Signalons au passage que le fait d'utiliser un système informatique disposant d'une horloge amène à approximer de facto le temps continu par une représentation discrète du temps, avec une granularité extrêmement fine (voir [Laruelle94] sur ce sujet), ce qui peut servir d'argument pour déconsidérer le PID.

Cela étant, de notre point de vue, se poser la question de la valeur de vérité d'une proposition lors d'un changement de celle-ci n'est surtout pas porteur de la moindre signification. Cela

revient à méconnaître la signification même du changement. C'est ce qui fait dire à van Bethem que l'"on a un problème de PID seulement si on tient à en avoir un" ...

Dans notre système IxTeT, deux approches distinctes ont été adoptées. En premier lieu, dans le domaine de la supervision, [Dousson94] s'est orienté vers la dernière des trois alternatives suggérées plus haut, à savoir considérer au niveau de représentation le plus élevé des intervalles fermés à gauche et ouverts à droite, suivant en cela l'approche de E.Tsang [Tsang87a]. Dans le cadre de la planification, c'est la deuxième alternative qui a été suivie. Afin de lever l'indétermination qui en découle, un principe similaire à [Vila93b] a été adopté: considérer au niveau le plus haut **à la fois** l'instant et l'intervalle, le premier rendant compte d'une modification d'une valeur de vérité, le second au contraire étant dépositaire d'une assertion établie durant un laps de temps. Cela suppose donc que l'extrémité d'un intervalle n'est pas inclue dans celui-ci, puisqu'elle s'exprime sous la forme d'un instant, lieu d'un changement. Nous y reviendrons au chapitre 2.

♣♣♣

Par ailleurs, on a souvent argué du fait que seul l'instant était à même de représenter un événement instantané. Or, après réflexion, un événement ponctuel représente un changement, donc le passage de vrai à faux d'une proposition *P*, ce qui peut s'exprimer sous la forme de deux intervalles se succédant l'un à l'autre.

En conclusion, manipuler aussi bien l'instant que l'intervalle au niveau le plus haut permet une appréhension simple et correcte des concepts à la fois de changement et d'activité. Cela n'influe cependant pas sur la représentation bas-niveau, qui peut a priori reposer aussi bien sur les intervalles que sur les instants, un intervalle pouvant s'exprimer sous la forme d'une paire d'instants, et un instant pouvant être vu comme l'extrémité d'un intervalle. Il reste à les comparer sous l'angle des relations temporelles, ce qui sera fait dans la prochaine section.

3.3. Les Relations Temporelles

Après avoir vu le problème du choix des primitives temporelles, il nous reste à étudier les divers modes de représentation des relations temporelles entre ces primitives. Nous avons choisi de consacrer une section particulière à ces aspects ontologiques primordiaux, en les replaçant dans le cadre plus général du raisonnement basé sur les contraintes. Signalons simplement qu'il va nous falloir considérer deux types de contraintes: les contraintes **symboliques** (positionnement relatif) et les contraintes **numériques** (dates et durées), ces dernières constituant la préoccupation essentielle de ce mémoire.

4. Les Gestionnaires de Relations Temporelles

Notre structure temporelle va donc reposer sur un ensemble discret de primitives (instants ou intervalles), reliées par des relations. Par exemple, le fait qu'Albert reste quelques jours chez sa belle-mère pendant les vacances peut s'exprimer, d'un point de vue temporel, sous la forme de deux intervalles I_1 et I_2 tels que *HOLD(vacances(Albert),*I_1*)* et *HOLD(chez(Albert,belle-mère),*I_2*)*. Prendre en compte de telles informations dans une base temporelle nécessite alors de pouvoir exprimer la relation $I_2 \subseteq I_1$.

Cet exemple définit ce que nous allons appeler une contrainte symbolique. Ce type de relation permet d'exprimer le positionnement relatif entre deux entités, c'est-à-dire les notions de précédence ou de recouvrement temporel. Nous allons voir que chercher à caractériser ces relations symboliques nous amène à définir des structures algébriques. Par ailleurs, il sera nécessaire également de manipuler des informations numériques, telles que durées ou dates, pour lesquelles la structure algébrique pourra être avantageusement présentée sous la forme d'un **CSP** (**Problème de Satisfaction de Contraintes**). En fait, l'approche par contraintes n'est qu'un cadre formel particulier dans lequel s'expriment des notions algébriques. Nous avons décidé de nous placer dès le départ dans cette perspective, et c'est pourquoi nous allons commencer par poser les bases théoriques de cette technologie. Signalons à titre de remarque que le terme de relations, utilisé dans les premiers travaux sur le sujet, et le terme de contraintes, utilisé plus spécifiquement dans le cadre des CSPs, sont équivalents. Nous utiliserons dans la suite aussi bien l'un que l'autre.

4.1. Introduction aux Problèmes de Satisfaction de Contraintes

Signalons tout d'abord qu'un état de l'art assez complet sur les CSPs peut être consulté dans la thèse de C.Bessières [Bessières92]. Un problème de satisfaction de contraintes est la donnée d'un ensemble de variables dont les valeurs sont restreintes par des contraintes. Nous allons nous intéresser exclusivement aux CSPs **binaires**, c'est-à-dire ne manipulant que des contraintes entre deux variables, et dans un premiers temps, au contexte classique dans lequel les contraintes sont **discrètes**. Le CSP à considérer est un triplet $\mathcal{G}$ = {**X**, **D**, **C**}, où

- **X**=$\{x_1, ..., x_n\}$ est un ensemble de variables,
- **D**=$D_1 \times ... \times D_n$ représente les domaines de ces variables (ensembles de valeurs pouvant leur être affectées a priori),
- **C**=$\{c_{i,j} / 1 \leq i,j \leq n\}$ est un ensemble de contraintes d'entrée.

Une **contrainte** $c_{i,j}$ exprimera la contrainte entre les variables x_i et x_j, restreignant du même coup les valeurs pouvant être prises simultanément par ces deux variables. Dans le cas discret, elle s'exprimera généralement sous la forme d'un ensemble de couples de valeurs permises $c_{i,j} = \{ (x_{i1}, x_{j1}), ..., (x_{ip}, x_{jp})\} \subseteq D_i \times D_j$.

Remarquons au passage qu'un tel réseau peut toujours être rendu complet (c'est-à-dire que l'on peut expliciter toutes les contraintes), en considérant que l'absence de contrainte entre deux variables x_i et x_j peut être représenté par une contrainte autorisant explicitement tous les couples de $D_i \times D_j$. On définit ensuite une solution d'un CSP comme étant une instanciation de toutes les variables satisfaisant l'ensemble des contraintes, c'est-à-dire:

- Une **solution** est un ensemble $\delta=\{x_1 \in D_1, ..., x_n \in D_n\} / \forall c_{i,j} \in \mathbf{C}, (x_i, x_j) \in c_{i,j}$.

Par ailleurs, un CSP binaire a l'avantage de pouvoir s'exprimer sous la forme d'un réseau où les noeuds sont les variables et les arcs sont les contraintes. Plusieurs problèmes courants peuvent être résolus dans un CSP:

1. La vérification de la **cohérence globale**, c'est-à-dire l'**existence** d'au moins une solution.
2. La détermination d'une ou de l'ensemble des solutions.
3. L'obtention du **réseau minimal** (ou "problème d'étiquetage minimal"): le réseau minimal est le réseau complet où les domaines des variables [resp. les contraintes] sont restreints aux seules valeurs [resp. seuls couples de valeurs] appartenant à au moins une solution. Les contraintes obtenues sont nommées "contraintes minimales".

Remarque : Les contraintes minimales, qui jouent un rôle central dans notre approche, comme nous le verrons au chapitre 2, représentent ce qui doit être nécessairement vérifié pour pouvoir construire une solution. C'est pourquoi on parle également de contraintes "nécessaires".

E.Freuder a défini précisément les notions de **cohérence** d'un réseau de contraintes [Freuder82]. Nous les rappelons ci-après succinctement:

- un réseau est ***k*-cohérent** ssi étant donné une instanciation de k-1 variables, satisfaisant l'ensemble des contraintes qui pèsent sur elles (on parle de solution partielle), et une k-ième variable v_k, il est possible de choisir une valeur dans le domaine de v_k qui satisfasse l'ensemble des contraintes pesant sur l'ensemble des k variables ainsi constitué.
- un réseau est **fortement *k*-cohérent** ssi il est j-cohérent $\forall\, j \leq k$.
- un réseau est **globalement cohérent** (ou décomposable [Dechter91]) ssi il est fortement n-cohérent, où n est la dimension du réseau.

En particulier, la 2-cohérence et la 3-cohérence dans un réseau complet s'expriment ainsi:

- Un CSP est 2-cohérent ssi
$\forall(i,j), i,j=1..n, i\neq j, \exists (x_i, x_j) \in D_i \times D_j / (x_i, x_j) \in c_{i,j}$.

- Un CSP est 3-cohérent ssi
$\forall(i,j,k), i,j,k=1..n, i\neq j\neq k, \exists (x_i, x_j, x_k) \in D_i \times D_j \times D_k /$
$(x_i, x_j) \in c_{i,j}, (x_j, x_k) \in c_{j,k}, (x_i, x_k) \in c_{i,k}$

D'un point de vue algorithmique, la recherche d'une solution passe généralement par des techniques d'**énumération avec retours-arrière**. C'est-à-dire que l'on choisit successivement une valeur pour chaque variable. Dès que l'on tombe sur une contrainte qui ne peux plus être satisfaite, on doit reconsidérer les choix effectués jusque-là. C'est ce que l'on appelle un retour-arrière. Plusieurs techniques de retour-arrière peuvent être appliquées, du retour-arrière chronologique aux approches "intelligentes" ("back-jumping", "forward-checking" [Bessieres92]), visant à revenir à la source de l'incohérence.

De ce fait, dans le cas le plus général, la recherche d'une solution est un problème NP-complet. Les résultats de Freuder permettent de limiter la combinatoire en appliquant des algorithmes de filtrages polynômiaux pour réduire l'espace de recherche [Mackworth77&85]. Un algorithme de 2-cohérence (ou **cohérence d'arcs**) permet d'éliminer des domaines les valeurs qui ne sont pas 2-cohérentes, en analysant tous les couples de variables, ce qui nécessite un algorithme en $O(n^2)$. La 3-cohérence (ou **cohérence de chemin**) peut de la même manière être établie par un algorithme cubique. Dans les deux cas, une incohérence est détectée dès qu'une contrainte, ou un domaine d'une variable, devient vide. Nous verrons ces algorithmes plus en détail dans le chapitre 2 dans le cadre des contraintes numériques.

L'avantage de disposer d'un réseau décomposable apparaît alors immédiatement:

1. La 2-cohérence ou la 3-cohérence implique la cohérence globale, c'est-à-dire que l'algorithme de filtrage détecte une incohérence ssi il n'existe pas de solution. En d'autres termes, ces algorithmes sont complets.

2. La propagation par 3-cohérence fournit le réseau minimal. Il est alors possible de déterminer une solution par énumération sans retour-arrière [Freuder82]. Ceci va nous être très utile dans la suite, essentiellement au niveau numérique. Par ailleurs, dans le réseau minimal, nous disposons directement des contraintes minimales, qui peuvent alors être requises en temps constant (cf chapitre 2).

4.2. Gestion de Relations Symboliques

Concernant les relations symboliques, une description plus formelle pourra être trouvée dans le travail d'Amine Mounir-Alaoui [Mounir90], qui a développé ces aspects dans le cadre du projet IxTeT.

Il convient généralement de commencer par définir l'ensemble des **relations élémentaires** entre deux primitives. En d'autres termes, supposons que I_1 et I_2 sont deux primitives "instanciées", c'est-à-dire dont on a fixé la position temporelle (un point ou un segment) sur la demi-droite $\Re^+$; la position relative temporelle entre I_1 et I_2 définit alors une relation élémentaire. Les relations que l'on peut ainsi spécifier sont mutuellement exclusives par définition. Elles définissent une **base** $\mathbf{B}=\{r_1, \ldots, r_n\}$.

Cela étant, les relations temporelles sont généralement partiellement indéfinies, comme on le verra au chapitre 2 dans le cadre de la planification. Cela signifie que les primitives ne sont pas instanciées a priori, et que nous disposons pour chaque couple de primitives de plusieurs relations possibles. Une relation symbolique s'exprime alors par une **disjonction** de relations élémentaires, que nous appellerons **relation composée**. Si l'on peut de cette manière exprimer toutes les relations possibles, on dit que la base est complète.

Ceci s'apparente à un CSP, dont les variables sont les primitives. Les domaines de ces variables sont des ensemble de points (si la primitive est l'instant) ou de segments (si la primitive est l'intervalle). Si nous considérons le temps continu, il s'agira respectivement de la demi-droite $\Re^+$ ou de l'ensemble des parties convexes de $\Re^+$. Si le temps est discret, on devra alors considérer les ensembles d'entiers ou de rationnels. Quant aux contraintes, il s'agit des relations composées. Trouver une solution consiste alors à choisir une relation élémentaire dans chaque relation composée, ce qui du même coup "fixe" la position des primitives. Nous allons poser tout cela de manière un peu plus formelle dans les deux cas de figure évoqués (instant et intervalle). A partir de quoi nous pourrons définir les différents types d'opérations sur ces relations, et la structure induite.

4.2.1. L'Algèbre Complète d'Intervalle

Nous considérons désormais $\mathbf{U}=\{I_1, I_2 \ldots\}$ comme étant un ensemble discret fini d'intervalles. Les choix ontologiques d'Allen l'ont amené à mettre en avant les 13 relations élémentaires entre intervalles. Nous avons représenté dans la figure ci-après uniquement les 7 relations directes.

Il obtient donc une base complète **B**= {b, m, o, s, f, d, e, b', m', o', s', f', d'} [Allen83], à partir de laquelle 2^{13} relations composées sont possibles. Nous noterons **R**={ρ_1, ρ_2, ...} l'ensemble de ces relations. Il reste alors à définir les deux opérations suivantes:

- l'**intersection** $\rho_1 \cap \rho_2$: par exemple, si d'une part on sait que I_1 [b $\vee$ m] I_2, et d'autre part I_1 [m $\vee$ o] I_2, on doit pouvoir en déduire que la seule relation élémentaire possible entre les deux est b. Allen fournit la table complète d'intersection pour tout couple de relations composées.
- la **composition** $\rho_1 \oplus \rho_2$: il s'agit en fait d'une opération permettant de calculer la relation transitive issue de deux relations: par exemple, si I_1 b I_2 et I_2 m I_3, alors I_1 b I_3. Là aussi, on peut donner la table complète pour la composition.

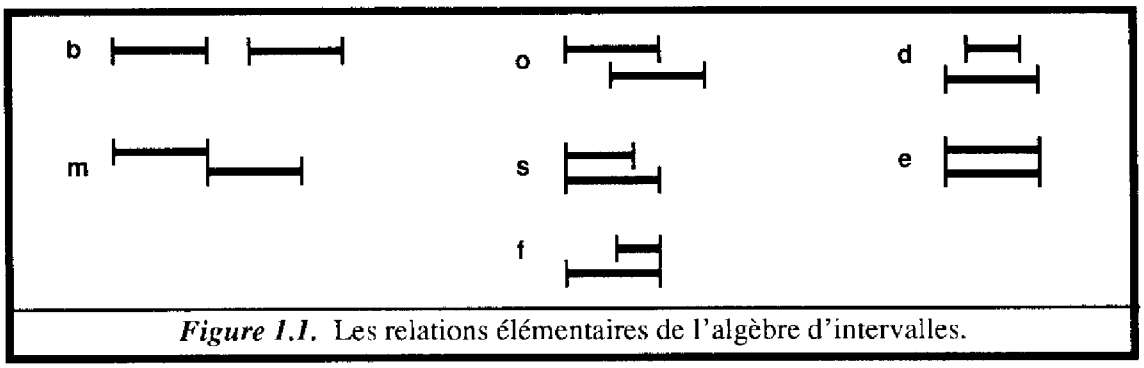


Figure 1.1. Les relations élémentaires de l'algèbre d'intervalles.

Ces deux opérations permettent de définir une structure d'algèbre sur l'ensemble **R** (voir par exemple [Tsang87a] pour une description formelle de diverses axiomatiques). Elles sont à la base de l'algorithmique qui va pouvoir être mise en place. Rappelons les exigences de base à ce niveau, qui sont (1) le maintien de la cohérence de la base à chaque ajout d'une nouvelle relation, et (2) l'explicitation de relations induites, qui permettront de répondre facilement à des requêtes sur la base. Allen propose pour se faire un algorithme polynômial qui détermine la fermeture transitive de l'ensemble des relations. Une incohérence est détectée dès lors qu'une relation est réduite à la disjonction vide, c'est-à-dire qu'aucune relation élémentaire n'est compatible avec le reste de la base. Remarquons que les opérations d'intersection et de composition sont également à la base des algorithmes de filtrage des CSPs. En fait, l'algorithme d'Allen peut être vu comme un algorithme de cohérence de chemin. Sa complexité est d'ailleurs en $O(n^3)$.

Cependant, M.Vilain et H.Kautz [Vilain86] ont démontré que cet algorithme n'était pas complet, c'est-à-dire qu'il pouvait ne pas détecter certaines incohérences, et que la vérification de la cohérence de l'ensemble des relations était un problème NP-complet.

4.2.2. Les Algèbres d'Instant et l'Algèbre d'Intervalle Réduite

C'est pourquoi Vilain et Kautz ont proposé une algèbre d'instants, où cette fois-ci l'ensemble **U** est un ensemble discret fini d'instants. La base est **B**={**<**, **=**, **>**} (c'est-à-dire *avant*, *égal* et *après*), à partir de laquelle on peut définir les 8 relations composées de l'ensemble **R**={**<**, **=**, **>**, $\leq$, $\geq$, **?**, $\neq$, **ø**}, où **?** exprime l'ignorance (c'est-à-dire la disjonction [**<** $\vee$ **=** $\vee$ **>**]), et **ø** traduit l'incohérence (c'est-à-dire la relation vide). Les opérations d'intersection et de composition s'obtiennent facilement de la même manière que pour les relations entre intervalles, et conduisent également à la définition d'une algèbre.

Dans [Vilain86], on trouve l'affirmation selon laquelle l'algorithme polynômial permettant d'obtenir la fermeture transitive est complet. P.van Beek infirme cette allégation [vanBeek89, Vilain89] en démontrant que cela n'est vrai que dans une sous-algèbre. Il nomme $PA^{\neq}$ l'algèbre de points complète, et PA l'algèbre d'instants **continue**, d'où l'on a exclu la relation d'inégalité $\neq$. Les propriétés d'algèbre demeurent (notamment en ce qui concerne la loi interne $\oplus$), et on vérifie qu'un algorithme de cohérence de chemin du type de celui d'Allen est complet et permet en plus de résoudre le problème d'étiquetage minimal. Cela signifie, dans une perspective CSP, que nous obtenons le **réseau minimal**. Celui-ci est tel que toute relation composée est réduite aux seules relations élémentaires compatibles avec les autres contraintes.

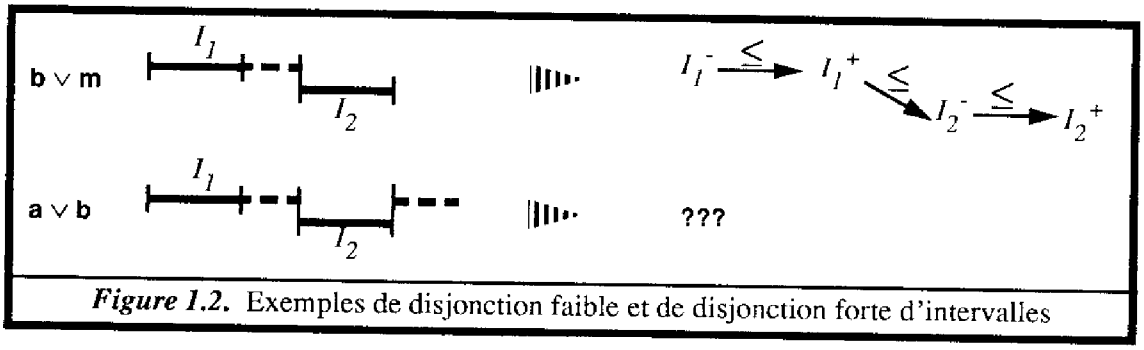


Figure 1.2. Exemples de disjonction faible et de disjonction forte d'intervalles

Van Beek démontre pour cela que PA est strictement équivalente à une algèbre réduite d'intervalle (PIA). Cette restriction consiste à exclure toutes les relations composées induisant une "discontinuité" dans l'ensemble des instanciations possibles des variables concernées [Vilain89]. Par exemple, comme l'illustre la figure ci-dessus, I_1 [a $\vee$ b] I_2 est une relation discontinue (nous parlerons de disjonction "forte"), alors que I_1 [b $\vee$ m] I_2 est une relation continue (nous parlerons de disjonction "faible"). Van Beek constate alors que les disjonctions faibles ont la propriété d'être exprimables sous la forme de conjonctions de relations de l'algèbre d'instants entre les extrémités des intervalles. C'est pour cela que PA est dénommée "algèbre d'instants continue". Si l'on note I_1^- et I_1^+ les instant de début et de fin de I_1, et de même pour I_2, on a par exemple

$$I_1 \text{ [b} \vee \text{m] } I_2 \Leftrightarrow I_1^+ \leq I_2^-$$

D'où Van Beek conclut que les algorithmes de filtrage polynômiaux ne sont complets que pour PA et PIA. Des travaux postérieurs [Nebel94] ont démontré qu'une classe plus large de relations entre intervalles pouvait être traitée en temps polynômial. Il n'en reste pas moins que des disjonctions fortes telles que I_1 [a ∨ b] I_2 sont irréductiblement absentes de telles classes.

Au niveau de l'algèbre d'instants, nous venons donc de voir que seule PA obéit à la propriété de 3-cohérence. Pour traiter le cas de l'algèbre complète d'instants $PA^{\neq}$, van Beek démontre qu'il suffit d'établir la 4-cohérence, et propose un algorithme polynômial qui calcule le réseau minimal en $O(n^4)$.

Signalons par ailleurs que vanBeek s'est intéressé au problème des requêtes dans un réseau temporel tel que ceux que nous venons de voir [vanBeek91]. Comme nous l'avons souligné, les informations temporelles sont partiellement indéterminées. Du coup, si l'on considère une requête du type "*est-ce que I_1 est en recouvrement avec I_2 ?*", on doit distinguer ce qui est **nécessairement** vrai de ce qui est **possiblement** vrai. On peut alors, comme nous le verrons dans le cadre de la planification, forcer par exemple une relation possiblement vraie à être nécessairement vraie en surcontraignant le réseau.

4.2.3. En Résumé

On peut trouver dans [Tsang87a] une synthèse des considérations qui précèdent. Il décrit de manière très complète les équivalences entre algèbres d'intervalle et d'instant. Sa conclusion est la suivante: soit l'on souhaite disposer d'une expressivité la plus riche possible, et l'algèbre d'intervalle est alors la plus appropriée, soit l'on souhaite disposer d'algorithmes polynômiaux complets, et il faut alors se passer des disjonctions fortes. Cela peut être fait en se basant sur l'algèbre d'instants, ou bien de manière équivalente sur l'algèbre réduite d'intervalles.

Signalons simplement pour finir les travaux théoriques de [Ligozat91], qui a étendu ces résultats en définissant une algèbre d'intervalles généralisée, qui permet de représenter des propositions vraie sur un ensemble de n intervalles disjoints (défini comme étant un intervalle généralisé de dimension n), et qui permet la prise en compte d'un instant sous la forme d'un intervalle généralisé de dimension 0.

4.3. Gestion de Contraintes Numériques

Nous allons voir maintenant comment incorporer des contraintes numériques, d'une part dans l'algèbre d'intervalles, et d'autre part dans l'algèbre d'instants.

4.3.1. Valuation Numérique d'Intervalles: un Formalisme Limité

La prise en compte de contraintes numériques dans le formalisme d'Allen se limite aux durées exactes des intervalles. JF.Rit [Rit88] étend ce formalisme en considérant des durées imprécises, sous la forme d'intervalles de durées possibles. Il propose des algorithmes de propagation complets, lesquels ne sont encore une fois polynômiaux que si l'on s'abstient d'exprimer des disjonctions fortes entre intervalles.

J.Dorn a développé des algorithmes efficaces de propagation de contraintes symboliques dans des réseaux d'intervalles fortement séquentiels [Dorn92]. Après quoi il a cherché à incorporer la dimension numérique par l'intermédiaire de dates de début et de durées imprécises pour chaque intervalle [Dorn94], comme dans le système de JF.Rit. Il met en avant un problème important, qui est la propagation des durées en présence de relations de type *during* (relation de base **d** d'Allen). Pour déterminer la contrainte minimale correspondant à la durée d'un intervalle I, il est nécessaire de déterminer tous les ensembles d'intervalles disjoints contenus dans I, et de faire la somme des durées des intervalles de chacun de ces ensembles, ce qui est dans le cas général un processus exponentiel.

Ces travaux se rejoignent sur un point: la limitation de l'expressivité. En effet, il s'agit de manipuler des intervalles qui ont une certaine durée, mais qui sont reliés par des relations purement symboliques. Comment représenter par exemple le fait que le recouvrement de deux intervalles ne doit pas excéder cinq minutes ? En d'autres termes, comment prendre en compte une valuation numérique des relations entre primitives ? Dans [Badaloni94], on se ramène à des précédences entre les instants de débuts et de fin des intervalles pour pouvoir leur associer des valuations numériques. Par exemple, "I_1 b I_2 *of 10 minutes*" sera représenté par la relation entre instants "$I_2^- - I_1^+ \leq 10$". En d'autres termes, on est obligé de se ramener à un formalisme basé sur l'algèbre d'instants.

Bref, lorsqu'on cherche à rajouter des valuations numériques directement dans une structure basée sur les intervalles, on se heurte à deux difficultés:

- l'absence d'un algorithme de propagation des contraintes numériques qui soit à la fois complet et efficace,
- la nécessité de se ramener à des contraintes binaires entre instants, ce qui limite d'autant l'avantage de se baser sur une représentation par intervalles.

4.3.2. Les CSPs Temporels: un Formalisme Riche et Efficace

Dans le cadre de l'algèbre d'instants, R.Dechter, I.Meiri et J.Pearl [Dechter91] ont défini le modèle des TCSPs ("Temporal CSP"), qui est sans nul doute l'approche la plus complète en

terme de valuation numérique d'un graphe d'instants. Ils considèrent un ensemble de contraintes numériques imprécises, données sous la forme d'un ensemble d'inéquations de la forme

$$inf_{ij} \leq X_j - X_i \leq sup_{ij},$$

où X_i représente la date de l'instant i (on confondra par la suite le plus souvent les deux). L'expression ci-dessus exprime donc une durée entre l'instant i et l'instant j. Par exemple, on pourra exprimer le fait qu'*Albert partira en vacances entre 3 et 5 jours après avoir rendu son rapport*. Dans une perspective de réseau d'instants, un ensemble d'inéquations de ce type peut donc s'exprimer sous la forme d'intervalles de durées possibles valuant les arcs (i, j). Formellement, cela nous donne la définition du CSP $\mathcal{G}=\{\mathbf{X}, \mathbf{D}, \mathbf{C}\}$ suivant:

- $\mathbf{X}=\{x_1, \ldots, x_n\}$ est l'ensemble des variables représentant les dates des n instants (ou par abus les n instants eux-mêmes),
- $\mathbf{D}=D_1 \times \ldots \times D_n$ désigne les domaines de ces dates (contraintes unaires),
- $\mathbf{C}=\{C_{i,j} / 1 \leq i,j \leq n\}$ désigne les durées entre instants (contraintes binaires).

Les domaines des variables, ainsi que les contraintes, sont ici **continus** et inclus dans $\Re$: ce sont des intervalles de valeurs possibles. On écrira par exemple $C_{i,j}=[inf_{ij}, sup_{ij}]$, ou $D_k=[inf_k, sup_k]$. En fait, si l'on définit l'instant 0 comme origine des temps, le domaine de la variable X_i (date de l'instant i) est identique à la contrainte $C_{0,i}$ (durée entre l'instant 0 et l'instant i) [Meiri91]. On peut donc homogénéiser la définition ci-dessus, en se ramenant à un simple ensemble de contraintes binaires à domaines continus, et considérer par exemple le problème dual qui consiste à chercher une instanciation des contraintes plutôt que des variables [Ghallab89]. Par ailleurs remarquons qu'ici aussi il est possible de rendre le réseau complet en plaçant entre deux instants non contraints numériquement la contrainte $]-\infty, +\infty[$.

La représentation graphique d'un TCSP est un ensemble de noeuds-instants reliés par des arcs-contraintes. Les définitions classiques de la littérature CSP s'expriment alors de la manière suivante:

1. Une solution sera désormais
   $\delta=\{x_1 \in D_1, \ldots, x_n \in D_n\} / \forall i,j=1..n, i \neq j, x_j - x_i \in c_{i,j}$.
2. Le réseau minimal se définit de la même manière que dans le cas général.
3. Un réseau sera 3-cohérent par exemple ssi
   $\forall (i,j,k), i,j,k=1..n, i \neq j \neq k, \exists x_i, x_j, x_k / (x_j - x_i) \in c_{i,j}, (x_k - x_j) \in c_{j,k}, (x_k - x_i) \in c_{i,k}$.

Dans [Dechter91], on trouve une distinction entre deux types de problèmes. D'un côté, les TCSPs généraux, où une contrainte peut s'exprimer sous la forme d'une disjonctions d'intervalles de durées possibles. Par exemple, on peut vouloir exprimer le fait qu'*Albert partira en vacances dès qu'il aura rendu son rapport, ou bien 3 ou 4 jours après*. D'autre part, on peut se

restreindre aux **STPs** ("**simple temporal problems**"), où l'on ne considère que des contraintes prenant la forme d'intervalles simples.

L'intérêt de cette distinction se situe au niveau algorithmique. En effet, seuls les STPs sont décomposables. Les opérations d'intersection et de composition, auxquelles nous ajoutons l'opération d'inversion, se définissent simplement de la manière suivante. Soient trois contraintes $C_{ij}=[inf_{ij}, sup_{ij}]$, $C_{ik}=[inf_{ik}, sup_{ik}]$ et $C_{kj}=[inf_{kj}, sup_{kj}]$, et soit $C_{ij}'=[inf_{ij}', sup_{ij}']$ une nouvelle contrainte entre i et j. Alors

- Inversion: $-C_{ij} = [-sup_{ij}, -inf_{ij}] = C_{ji}$,
- Intersection: $C_{ij} \cap C_{ij}' = [\max(inf_{ij}, inf_{ij}'), \min(sup_{ij}, sup_{ij}')]$,
- Composition: $C_{ik} \oplus C_{kj} = [inf_{ik}+inf_{kj}, sup_{ik}+sup_{kj}]$.

A partir de là, il est possible de vérifier la cohérence d'un réseau par un simple algorithme de filtrage [Mackworth77]. L'arc-cohérence est a priori suffisante, mais la propagation par cohérence de chemin dispose d'un avantage supplémentaire: elle rend compte du réseau complet minimal, et permet donc de construire une solution sans retour-arrière. Cela n'est vrai que dans les STPs, et est dû à la propriété de distributivité de l'intersection sur la composition. Nous verrons au chapitre 2 en quoi le réseau minimal peut nous être utile. Nous fournirons par la même occasion l'algorithme de cohérence de chemin que nous avons choisi d'utiliser dans notre système, et les raisons précises de ce choix. Nous pouvons simplement ici donner des définitions affinées de la 3-cohérence et du réseau minimal:

(1) Un réseau est 3-cohérent ssi $\forall(i,j,k)$, $i,j,k=1..n$, $i\neq j\neq k$, $C_{ik} \oplus C_{kj} \subseteq C_{ij}$.
(2) Le réseau minimal sera tel que $\forall(i,j,k)$, $i,j,k=1..n$, $i\neq j\neq k$, $C_{ik} \oplus C_{kj} = C_{ij}$.

Il peut être intéressant de signaler dès à présent que les STPs permettent de retrouver les contraintes symboliques, grâce aux correspondances suivantes:

- $C_{ij}=\{0\}$ ssi $i = j$
- $C_{ij}\subset]0, +\infty[$ ssi $i < j$
- $C_{ij}\subset]-\infty, 0[$ ssi $i > j$

A partir de là, on peut construire les correspondances pour les relations $\leq$, $\geq$, $\neq$ et **?**. De même, l'incohérence s'exprime dans le cas des TCSPs par une contrainte $C_{ij}=[inf_{ij}, sup_{ij}]$ telle que $inf_{ij} > sup_{ij}$. En d'autres termes, les contraintes symboliques sont incluses dans les TCSPs, ce qui fait de ce formalisme un **modèle complet** de gestion des contraintes temporelles.

♣♣♣

Pourtant, plusieurs approches ont cherché à gérer à la fois des contraintes symboliques explicites, et des réseaux de contraintes numériques du type des STPs. H.Kautz et P.Ladkin [Kautz91], tout d'abord, utilisent **deux réseaux séparés**, un réseau d'intervalle "à la Allen"

pour le symbolique et un STP pour le numérique. Ils construisent à partir de là des algorithmes de traduction d'un réseau à l'autre. I.Meiri [Meiri91] propose de son côté des **réseaux généralisés**, dont les noeuds sont soit des instants, soit des intervalles, et les contraintes sont soit qualitatives, soit quantitatives. Elle démontre alors que les seuls cas polynômiaux sont ceux qui sont limités à l'algèbre PA (ou l'algèbre d'intervalles équivalente PIA), augmentée d'une structure de type STP, le tout dans un même et unique réseau. Nous reviendrons sur ces systèmes lorsque nous chercherons des éléments de comparaison avec notre approche, à la fin du chapitre 3. Il sera également utile de discuter de l'intérêt qu'il y a à conserver deux types de contraintes, alors même que les STPs peuvent être vus comme une synthèse du symbolique et du numérique. Ceci sera analysé en temps utile.

4.4. Les "Time-Map Managers"

Parallèlement à ces travaux théoriques sur les structures temporelles, D.McDermott et T.Dean ont cherché à développer un module séparé de gestion des contraintes temporelles, qu'ils ont appelé **TMM** ("Time-Map Manager") [Dean87]. Il s'agit d'un module complet destiné à être utilisé dans des applications, essentiellement de planification. Ils se basent sur les travaux théoriques de D.McDermott pour construire une base temporelle où les primitives sont les instants. Ils définissent un "time token" comme étant une assertion temporelle, c'est-à-dire une occurrence d'événement, un événement étant comme pour Allen une proposition vraie entre deux instants.

La particularité de ce système est qu'il rend compte d'un niveau sémantique en gérant explicitement la contradiction entre deux événements par l'intermédiaire de règles de logique. Il incorpore également la prise en compte de la non-monotonie, en utilisant un **ATMS** (Assumption-Based Truth Maintenance System), qui permet de mémoriser en même temps qu'une contrainte le fait qui en est la cause. Ainsi, lorsqu'on retire un fait de la base, il est possible également de retirer toutes les contraintes qu'il a engendré et de revenir donc aisément à un état antérieur. La manipulation de tels modèles de dépendance induit généralement une complexité supplémentaire non négligeable, ne serait-ce qu'en espace.

Le modèle de S.Materne et J.Hertzberg [Hertzberg91] utilise le TMM en lui adjoignant des informations de type date et durée et en étendant les concepts de persistance et de contradiction. Ils considèrent en effet aussi bien la persistance à gauche que la persistance à droite, et deux propositions contradictoires voient leurs persistances respectives "limitées" par un simple ajout d'un instant "terminant" la première proposition, un instant "débutant" la deuxième, et une contrainte de précédence reliant ces deux instants. On obtient donc un système utilisant à la fois le pouvoir expressif de l'algèbre d'instant et des mécanismes d'inférence élémentaires.

4.5. En Résumé

L'approche des TCSPs permet de représenter des contraintes numériques reliant n'importe quel couple d'instants, et fournit des algorithmes de propagation complets, issus directement de la technologie CSP. Elle est en cela beaucoup plus riche que les approches par valuation d'intervalles. Par rapport aux approches de type TMM telles que celle de Hertzberg et Materne, elle offre un cadre théorique plus formel et ne tient pas compte de concepts de sémantique temporelle tels que la persistance et la contradiction. En ce sens, les TCSPs constituent un modèle ayant un champ d'application plus vaste. Nous retrouvons cependant le même type de limitation que dans le cas symbolique: pour disposer d'algorithmes complets polynômiaux, il est nécessaire d'exclure les disjonctions d'intervalles de durées possibles.

5. Conclusion

Dans ce premier chapitre, nous avons fait le tour des diverses approches de modélisation du temps, pour nous focaliser sur celles qui relèvent de la **logique réifiée**. Celles-ci conduisent à la définition de théories du temps permettant de décrire efficacement les concepts de changement et de causalité. A côté de ces représentations haut-niveau, nous nous sommes intéressés à des **représentations bas-niveau**, où la dimension sémantique des phénomènes est découplée de leur dimension temporelle. Nous aboutissons alors à des **gestionnaires de relations temporelles**, dont le rôle est de se mettre au service du système global de résolution de problème, de manière à (1) **vérifier la cohérence** de l'ensemble des contraintes temporelles, et (2) répondre à une **requête temporelle** provenant de ce système global.

Il s'agit alors de choisir qui de l'instant ou de l'intervalle sera la **primitive temporelle** de base. Il s'avère que ce choix est purement ontologique. En effet, dès que l'on souhaite disposer d'algorithmes de propagation de contraintes symboliques **complets** et **polynômiaux**, on doit exclure de notre représentation les "**disjonctions fortes**", et ceci indépendamment de la représentation interne choisie, qui sera laissée à la libre appréciation du concepteur du modèle temporel, en fonction de l'application. On peut simplement noter que l'instant est mieux adapté à la représentation du changement, l'intervalle à celle de l'activité. Au-dessus de cette représentation interne, on adoptera le plus souvent un langage de représentation utilisant l'un et l'autre.

Cela étant, dès que l'on s'intéresse aux **contraintes numériques**, on se rend compte que les approches basées sur des **réseaux d'instants** témoignent d'une représentation plus homogène, et fournissent des algorithmes de propagation complets et efficaces. [Dechter91] fait la synthèse de ce type d'approche, qui, outre sa facilité de mise en oeuvre, inclut implicitement la

prise en compte de contraintes symboliques. On observe de plus une propriété similaire à celle rencontrée dans le cadre des contraintes symboliques: les familles de problèmes polynômiaux se définissent en excluant les cas disjonctifs.

Partant de là, nous allons maintenant voir quels ont été nos choix initiaux en termes de représentation et d'algorithmique temporelle dans le système IxTeT, en situant celui-ci dans le cadre de son domaine d'application privilégié qu'est la planification.

Chapitre II. Planification Temporelle: l'Approche IxTeT

Le travail présenté dans ce mémoire se veut avant tout une contribution au raisonnement temporel. C'est pourquoi nous avons souhaité, dans le chapitre précédent, le situer dans ce cadre, indépendamment de tout domaine d'application. Néanmoins, les choix que nous ferons ne sont pas innocents, et il est indispensable de présenter, quoique succinctement, l'architecture dans laquelle notre gestionnaire de contraintes temporelles est censé s'intégrer. Nous commencerons par un rapide coup d'oeil sur ce qu'il convient de retenir de l'état des connaissances en matière de planification.

1. La Planification de Tâches: Visite Guidée du Domaine

En guise de préambule, rappelons que nous faisons référence ici à la planification de tâches, et non au domaine distinct qu'est la planification de trajectoires ou de mouvements. Signalons également qu'il ne s'agit en aucun cas de faire un état de l'art de la planification (le lecteur souhaitant approfondir sa connaissance du domaine pourra par exemple se reporter à la thèse de P.Régnier [Régnier92]), mais simplement de donner quelques pointeurs sur les aspects qui nous intéressent du point de vue du raisonnement temporel.

Le problème de planification peut se poser d'une manière très générale. On dispose d'un ou de plusieurs agents auxquels on doit fournir un plan de tâches qui leur permettra de réaliser un but prédéfini. Ceci peut s'écrire de la manière suivante: étant donnés

- une **situation initiale**,
- un modèle des **tâches** pouvant être exécutées par l'agent,
- un **but** assigné, sous la forme généralement d'une conjonction de propositions qui doivent devenir vraies,

on souhaite trouver un plan, c'est-à-dire un ensemble de tâches temporellement ordonnées qui permette d'atteindre le but à partir de la situation initiale.

1.1. Les Approches Classiques

La première famille d'approches dérive du Calcul de Situations (cf chapitre 1, § 2.1.1). La situation initiale est l'état du monde à l'instant t_0, et les tâches sont des changements instantanés permettant de passer d'un état à un autre. Une modélisation sous forme purement logique se heurtant au problème de rémanence déjà évoqué, une représentation par opérateurs a été introduite dans STRIPS [Fikes71]. Un opérateur de changement représente une tâche par l'intermédiaire de

- ses **préconditions** (regroupées dans une "delete-list"), qui doivent être vraies avant le déclenchement de la tâche.
- ses **postconditions**, ou **effets** (regroupés dans une "add-list") qui deviennent vraies à l'issue de la tâche.

Toutes les propositions qui ne sont pas explicitement préconditions ou effets persistent implicitement pendant la tâche, ce qui résout le problème de rémanence. Ce formalisme, qui repose sur une hypothèse de **complétude de l'information**, est cependant trop fortement déterministe: il décrit des tâches génériques, sans tenir compte du contexte d'exécution. Dans certaines situations particulières, la tâche aura des effets supplémentaires, qui dépendent non pas de la tâche elle-même, mais de l'état du monde à ce moment-là. Par exemple, une tâche de déplacement d'un robot induit comme effet principal le changement de position de celui-ci. Si une remorque lui est attaché, alors la position de cette remorque devra également avoir changé. Il s'agit là de ce que l'on appelle le problème de **ramification**. Les effets induits tels que celui qui vient d'être décrit sont appelés effets secondaires de la tâche. Il est bien sûr exclu de chercher à écrire une tâche pour chaque situation possible. L'approche la plus convaincante consiste à inférer les effets secondaires à partir d'**axiomes du domaines**. En complément au problème de ramification, signalons ègalement le problème de **qualification**, qui à l'inverse se pose lorsqu'une tâche dispose de préconditions qui sont superflues, car presque toujours vérifiées, sauf dans certaines situations particulières où il conviendrait alors de s'assurer de leur présence.

D'un point de vue algorithmique, la méthode la plus couramment utilisée pour construire un plan, et sur laquelle nous nous baserons, est la suivante: on cherche une tâche qui a pour effet le but désigné. Elle est alors **insérée** dans le plan, et ses préconditions deviennent des sous-buts. Certains sous-buts étant vrais dans la situation initiale, on dit qu'ils sont "**expliqués**". Pour les autres, on relance le processus de recherche d'une tâche résolvant le sous-but. L'algorithme s'arrête lorsqu'il ne reste plus de sous-but non expliqué. La description des opérateurs recourt le plus souvent à des variables quantifiées. Se posent alors les problèmes (1) du **choix de la tâche** à insérer, et (2) du **choix d'instanciation** des variables pour cette tâche. Ces choix sont effectués à l'intérieur d'un **arbre de recherche**. Dès que l'on s'intéresse à des problèmes un tant soit peu réalistes, on se trouve confronté à une explosion combinatoire au niveau de cet arbre. On touche là au problème majeur de la planification: sa complexité intrinsèque. Il est alors nécessaire de développer des heuristiques et des stratégies qui vont permettre de réduire cette complexité.

1.2. Planification Non-Linéaire

Au niveau du pouvoir expressif, il faut distinguer tout d'abord planification linéaire et planification non-linéaire. La planification linéaire ne permet d'obtenir qu'une succession d'états, ce qui ne permet pas de prendre en compte le **parallélisme** de certains phénomènes. La planification non-linéaire pallie à ces inconvénients: dans les successeurs de STRIPS, NOAH par exemple, on cherche à satisfaire des buts conjonctifs, ce qui nécessite de tenir compte des interactions entre les différents buts. Cela oblige à se départir d'une construction linéaire en acceptant le parallélisme entre tâches insérées durant la recherche. NOAH et ses successeurs (comme O-PLAN par exemple) utilisent pour cela des réseaux procéduraux où les tâches instantanées sont reliées entre elles par des contraintes de précédence.

Par contre, la planification non-linéaire pose des problèmes nouveaux en termes de maintien des relations de causalité entre tâches et effets. D.Chapman [Chapman87] a posé les bases théoriques de la planification non-linéaire, dans le système TWEAK, en proposant un **Critère de Vérité**, qui permet de vérifier que l'ajout d'un nouvel opérateur ne vient pas détruire une "explication" déjà établie. Dans le planificateur SNLP [McAllester91], cette volonté s'exprime par l'intermédiaire de la définition de **liens causaux** entre l'opérateur qui "produit" l'effet et celui qui "consomme" celui-ci comme précondition. Ces deux planificateurs proposent également un **algorithme de contrôle global** de l'arbre de recherche reposant sur ce critère.

1.3. Planification Hiérarchique

La première tentative visant à se démarquer des approches classiques et à réduire de manière substantielle la complexité de la tâche de planification coïncide avec ABSTRIPS: une manière d'orienter la recherche est de classer les sous-buts selon leur **criticité**, c'est-à-dire l'effort nécessaire a priori pour parvenir à les expliquer. On planifie d'abord en fonction des sous-buts les plus critiques, après quoi on raffine le plan progressivement en analysant les sous-buts de niveau inférieur.

SIPE [Wilkins88] est un exemple récent de planificateur évolué intégrant à la fois les concepts de hiérarchie et de non-linéarité. Le principe général, qui résume parfaitement ce qui précède, est le suivant: on part d'une représentation de la situation initiale (appelé **plan initial**) dans laquelle on incorpore progressivement des opérateurs sans ordre pré-établi, en fonction de critères hiérarchiques donnés. On construit donc une succession de **plans partiels**, jusqu'à l'obtention du **plan final**. Cette approche permet de développer des stratégies d'**engagement minimal**, c'est-à-dire que l'on conserve le parallélisme le plus longtemps possible, n'ajoutant des contraintes de linéarisation que lorsque cela est vraiment nécessaire.

1.4. Les Planificateurs Temporels

Il n'en reste pas moins que ces travaux témoignent tous de la même lacune en termes d'expressivité: l'instantanéité des opérateurs. Plusieurs planificateurs se sont attachés à circonvenir à cette carence, en introduisant une **représentation explicite du temps**. Ils sont à ce titre qualifiés dans la littérature de planificateurs temporels. Comme nous allons le voir, donner la possibilité aux opérateurs de durer dans le temps enrichit le modèle en termes d'expressivité, mais complexifie également la tâche de planification en multipliant les possibilités de conflits.

Le précurseur dans ce domaine est sans nul doute DEVISER. Son formalisme est essentiellement dérivé de celui des réseaux PERT [Kaufman69], avec une représentation plutôt basée sur les instants. Une tâche doit être déclenchée à l'intérieur d'une fenêtre de dates possibles, et dispose d'une durée fixe. L'un des aspects intéressants de cette approche est la possibilité de disposer dans la situation initiale d'événements attendus dans le futur à des dates connues. Au niveau du but, il est possible également de préciser sa date et la durée pendant laquelle il doit rester vrai. Cette approche se heurte néanmoins à la complexité algorithmique de la propagation des durées en présence de relations entre assertions du type *during*, aspect qui avait déjà été mis en avant au sujet des travaux de J.Dorn (cf § 4.3.1 au chapitre 1).

TIMELOGIC [Allen83] et FORBIN [Miller85] sont les premiers planificateurs à s'inspirer directement des modélisations du temps basées sur l'algèbre d'intervalles pour le premier, et d'instants pour le second. Dans les deux cas, on gère les relations temporelles dans un module séparé, où se pose alors le problème du maintien d'une base temporelle cohérente.

Dans TIMELOGIC, on retrouve toute la puissance expressive des relations d'Allen. Les algorithmes de propagation de relations temporelles sont rendus plus performants grâce à une structure temporelle hiérarchique sous la forme d'intervalles de référence: seules sont explicitées les relations entre intervalles de références, et entre un intervalle de référence et tous ceux qu'il contient. On aboutit ainsi à une décomposition du réseau, permettant des propagations locales, et donc des gains de performances. La critique que l'on peut formuler est double. D'abord, le gain avéré de performance des algorithmes est faible (on obtient une réduction des temps de calculs d'environ 50%). Ensuite, une telle structure hiérarchique des phénomènes temporels dans un plan n'est pas immédiate, et est sujette à variations. J.Koomen propose [Koomen88] de maintenir cette structure de manière dynamique en cours de planification. Il n'en reste pas moins qu'une telle technique ne peut être avantageusement appliquée qu'à des applications où une représentation hiérarchique des propositions temporelles s'impose, ce qui n'est pas véritablement une caractéristique commune des applications les plus couramment abordées.

FORBIN utilise quant à lui le TMM de Dean & McDermott, ce qui lui permet d'utiliser les capacités de raisonnement non-monotone pour gérer les retours-arrières à l'intérieur-même du TMM (voir au chapitre 1). Comme nous l'avons dit, cette gestion est relativement lourde. De plus, FORBIN représente les durées moyennes des tâches, c'est-à-dire que comme DEVISER, il ne tient pas compte de durées imprécises. Par contre, FORBIN est capable de déterminer à partir d'un plan partiellement ordonné une linéarisation qui optimise le temps global du plan. Ceci est fait grâce à un système d'ordonnancement distinct du TMM.

Nous ne pouvions pas ne pas parler de TRIPTIC [Rutten93]. Ce planificateur utilise le TMM de Hertzberg & Materne (voir au chapitre 1). Il a donc à sa disposition la gestion des liens causaux grâce à l'ATMS du TMM, mais il est aussi en mesure de gérer les conflits liés aux propositions contradictoires, grâce à son mécanisme de restriction de la persistance. La richesse de représentation au niveau numérique est également une force supplémentaire. Néanmoins, il reste trop profondément attaché (tout comme DEVISER par exemple) à la philosophie "à la STRIPS", où les préconditions sont rendues fausses juste au début de la tâche et les effets deviennent vrais juste à la fin de celle-ci. Il n'utilise donc pas totalement la richesse de la représentation par instants, contrairement à IxTeT, comme nous allons pouvoir le constater dans la deuxième section de ce chapitre.

Comme nous pouvons le constater, les planificateurs temporels se démarquent des approches issues du Calcul de Situation selon deux axes:

- du point de vue de la **représentation**, ils rendent compte d'une plus grande **richesse expressive** en termes de relations temporelles entre les diverses propositions composant le plan, aussi bien au niveau symbolique que numérique,
- du point de vue **algorithmique**, ils reposent sur une séparation du travail entre le planificateur proprement dit, qui ne gère que le choix et l'insertion de résolvantes de sous-buts, selon les principes rapidement évoqués dans les paragraphes précédents, et un **gestionnaire de contraintes temporelles**, qui maintien la cohérence au niveau strictement temporel. Le lien entre les deux se situe notamment au niveau de la **gestion des conflits**, qui peut être totalement prise en charge au niveau de l'algorithme de planification, comme nous le verrons avec IxTeT, ou partiellement traitée au niveau d'un TMM, comme dans TRIPTIC.

2. Présentation du Système IxTeT

2.1. L'Architecture Décisionnelle du Robot

Nous avons choisi de situer tout d'abord très succinctement l'approche IxTeT dans le cadre de la robotique, domaine dont elle est issue. Le planificateur s'insère dans l'architecture globale d'un **système de décision** pour un **robot autonome**, et constitue un système de raisonnement "haut niveau", dont le but est de fournir au robot un plan de tâches calculé "hors-ligne".

Le plan produit est passé au **module de supervision de tâches** qui est chargé de l'affiner, et éventuellement de modifier la tâche en cours en **réaction** à des événements imprévus. En cas de dysfonctionnement important, le superviseur peut remonter l'information au niveau du planificateur pour lancer une **replanification** partielle de la mission. Ceci traduit le distinguo entre un planificateur postulant l'existence d'un monde prévisible, et un superviseur considérant au contraire l'aspect stochastique de l'environnement, grâce à des capacités de planification réactive de bas-niveau.

Au niveau le plus fin, le **contrôleur d'exécution** distribue les tâches élémentaires selon leur type aux **modules fonctionnels** appropriés: les tâches de vision sont par exemple gérées par le module de perception, interprétation et modélisation de l'environnement, les tâches de déplacement par le module de planification de trajectoires, etc.

2.2. Le Planificateur Temporel

Nous allons maintenant présenter le planificateur IxTeT tel qu'il a été développé par H.Laruelle [Laruelle94], sur la base d'une première étude réalisée par A.Mounir-Alaoui [Mounir90]. Nous ne ferons qu'évoquer superficiellement les améliorations en cours, qui font l'objet de la thèse de P.Laborie [Laborie95a].

2.2.1. Le Choix d'un Planificateur Temporel

Le planificateur IxTeT se place dans le cadre des approches par opérateurs. Les hypothèses de départ sont celles d'un **monde prévisible** et d'un **ensemble d'informations complet**, afin de limiter la combinatoire du problème. L'objectif principal était de proposer un planificateur capable de s'attaquer à des problèmes réalistes, par l'intermédiaire d'une **représentation tem-**

porelle riche. De ce point de vue, IxTeT se situe dans la droite lignée des planificateurs temporels, avec une séparation stricte entre le planificateur proprement dit et un gestionnaire de contraintes temporelles. Par contre, il s'agissait également de profiter pleinement de cette richesse expressive pour s'écarter résolument d'une philosophie "à la STRIPS" par trop réductrice, ce qui a nécessité le développement d'une théorie basée sur un **critère logique de plan solution**. En ce sens, IxTeT apparaît comme l'application des théories formelles de la planification non-linéaire, introduites par D.Chapman, au cadre de la planification temporelle.

2.2.2. Choix en Termes de Représentation

La représentation **haut-niveau**, assurant le lien entre la dimension logique et la dimension temporelle, se réfère à la logique réifiée selon le modèle de Y.Shoham, ce qui conduit à la définition du prédicat universel HOLD pour désigner une assertion temporelle, qui sera toujours considérée comme étant homogène (cf § 3.1 au chapitre 1). Un second prédicat EVENT, défini à partir du HOLD, permet de modéliser un changement instantané. Nous disposons donc d'un langage haut-niveau permettant de prendre en compte à la fois le concept d'activité et celui de changement (cf [Vila93b]). La représentation élémentaire d'une tâche (sans tenir compte de ses effets) sera par exemple la succession d'un EVENT (marquant le début), un HOLD (marquant le déroulement) et un EVENT (marquant la fin).

Au niveau **temporel**, la représentation bas-niveau reposera sur un ensemble discret d'**instants.** L'instant constitue en effet de notre point de vue la manière la plus naturelle de représenter le changement, notamment un événement ponctuel survenant à une date donnée. Ce choix ontologique permet de plus de manipuler plus directement les contraintes temporelles numériques, comme nous l'avons vu au chapitre précédent.

Au niveau **logique**, le choix s'est porté sur une représentation des propositions sous la forme d'attributs multivalués, c'est-à-dire qu'un attribut prendra une valeur dans un domaine donné. Par exemple, pour exprimer la position du robot, au lieu de *position(robot, pièce1)*, on écrira *position(robot) = pièce1*. Ceci permet de résoudre a priori un grand nombre de conflits, en interdisant implicitement des contradictions issues des propriétés du monde dans lequel on se situe, telles par exemple que la non ubiquité, dans l'exemple donné. Remarquons également que l'on retrouve un formalisme type CSP, les attributs représentant des variables auxquelles sont associées des domaines de valeurs. La représentation et la gestion de contraintes d'instanciation sont du même coup facilitées. Par ailleurs, une typologie des attributs a été définie par rapport à leur portée temporelle. Nous distinguons:

- Les attributs **rigides**: leur valeur n'est pas modifiée dans le temps, comme par exemple la *connexité* entre deux pièces contiguës,

- Les attributs **contingents**: leur valeur varie dans le temps hors de toute influence de l'agent pour lequel on planifie, comme par exemple la *lumière-extérieure*, qui passe de *jour* à *nuit*,
- Les attributs **contrôlables**: leur valeur est modifiée par l'agent au cours du temps, comme par exemple la *position* du robot.

A partir de là, on peut représenter un opérateur (ou une **tâche**, selon la terminologie propre à IxTeT) comme un intervalle temporel relié à ses préconditions et effets par des **contraintes symboliques** quelconques, autorisant ainsi une plus grande expressivité au niveau temporel. On pourra par exemple exprimer le fait qu'un effet devienne vrai après un certain laps de temps à l'issu de la tâche. La figure ci-dessous illustre l'exemple de la tâche *Saisir*, pour laquelle nous avons représenté les assertions par des traits horizontaux et les événements (c'est-à-dire les changements de valeur d'un attribut) par un tiret vertical. L'absence de tiret vertical à l'extrémité d'une assertion exprime la persistance de celle-ci, que ce soit à gauche ou à droite. Dans le cas de la tâche *Saisir*, l'agent (le robot) doit être au même endroit que l'objet à saisir durant toute la tâche. L'objet change de position à un instant donné à l'intérieur de la tâche. Au même instant, la main du robot n'est plus libre. Remarquons que cette tâche est générique dans le sens où l'objet, ainsi que sa position initiale (qui est aussi celle du robot) ne sont pas instanciés a priori. Des **axiomes de cohérences** peuvent être liés à cette tâche, comme par exemple le fait que la taille de l'objet doit être compatible avec la capacité de la main du robot.

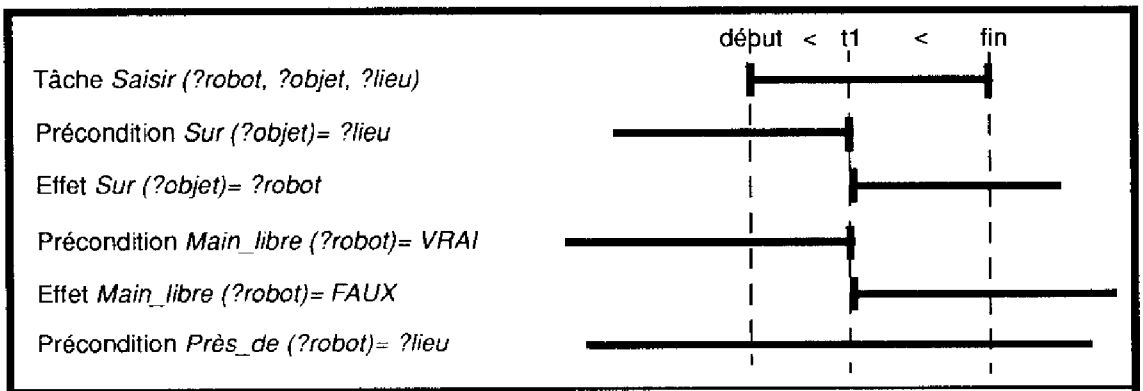


Figure 2.1. Représentation temporelle d'un opérateur *Saisir*.

De plus, nous pouvons facilement représenter dans la situation initiale (appelée plan initial) des événements attendus à des dates données, ou des buts également datés explicitement.

Une description hiérarchique des tâches a été adoptée, sous la forme de **macro-opérateurs**, eux-mêmes composés de plusieurs sous-tâches. Par exemple, un macro-opérateur *Déplacer-objet* sera composé des trois tâches élémentaires *Saisir*, *Aller-à* et *Poser*. Ces macro-opérateurs constituent donc des "morceaux de plan" pré-établis, que l'on pourra insérer directement, faci-

litant du même coup le travail du planificateur. On autorise la présence d'incohérences dans la description de ces macro-tâches, qui seront résolues en fonction du contexte. Ceci permet de disposer de macro-opérateurs génériques pouvant s'adapter à diverses situations. Il est à noter que ces tâches sont précompilées hors-lignes, c'est-à-dire qu'elles sont transformées en un graphe d'instants qui pourra être directement inséré dans le graphe global représentant le plan en cours.

En résumé, retenons qu'une base IxTeT représentant un **plan partiel** se composera d'un **ensemble d'assertions et d'événements**, d'une **table de variables** à instancier, et d'un **graphe d'instants**.

2.2.3. Stratégies et Fonctionalités du Système de Planification

A partir de là, H.Laruelle a cherché à définir un critère de plan solution, prenant en cela pour modèles les travaux menés dans les planificateurs non-temporels TWEAK ou SNLP. Il définit les concepts de conflit possible et conflit nécessaire. Un **conflit possible** correspond soit à deux assertions possiblement contradictoires susceptibles de se recouvrir, ou à un événement susceptible de venir "casser" une assertion (on parle alors de **menace**), soit à un conflit d'utilisation d'une ressource non partageable. Il y a deux manières distinctes de résoudre un conflit:

- **instancier** les deux prédicats de manière à ce qu'ils ne soient pas contradictoires,
- **ajouter des contraintes temporelles** forçant le non-recouvrement des occurrences, un peu à la manière du principe de "promotion/demotion" de TWEAK [Chapman87].

Une instanciation ou une contrainte temporelle résolvant un conflit sera appelée **résolvante**. On parlera de **conflit nécessaire** si et seulement si il n'existe pas de résolvante compatible avec le plan courant. La présence d'un conflit nécessaire implique l'absence de solution, et nécessite donc un retour-arrière sur l'un des choix précédemment effectués. L'obtention d'une **solution** se caractérise alors comme l'absence de conflits possibles et de sous-buts non expliqués. C'est sur ces simples caractérisations qu'il est alors possible de bâtir un algorithme de planification complet et correct. Le principe général repose sur un arbre de recherche globale, grâce auquel on va s'attacher à expliquer les sous-buts, tout en résolvant les conflits possibles, selon une stratégie globale de **moindre engagement**. Expliquer un sous-but se fait par

- simple établissement d'un lien causal, par **ajout d'une précédence**, si le plan contient déjà une proposition pouvant expliquer le sous-but (on parle d'"établissement naturel"),
- **ajout d'un opérateur** ayant pour effet le sous-but.

L'originalité de l'approche repose dans la notion unifiée de résolvante, aussi bien pour les sous-buts que pour les conflits. On aboutit à une architecture à trois composantes, pour laquelle

nous reprenons le schéma figurant dans la thèse d'H.Laruelle, quelque peu mis à jour, qui permet de bien visualiser l'ensemble de la méthode, laquelle va être rapidement décrite ci-après.

1. le **module d'analyse** gère à la fois la recherche des résolvantes des sous-buts en cours (étape de faisabilité), et l'identification des conflits induits et la recherche des résolvantes existant pour ces conflits (étape de satisfiabilité);

2. le **module de contrôle de la recherche** va effectuer une résolution globale selon trois niveaux:
   - choix d'une opération (sous-but ou conflit),
   - choix de l'opérande, c'est-à-dire le sous-but ou le conflit particulier,
   - choix de la résolvante dans l'arbre de recherche globale;

3. le **module de gestion du plan partiel** met à jour l'ensemble des prédicats, la table des variables et le graphe d'instants.

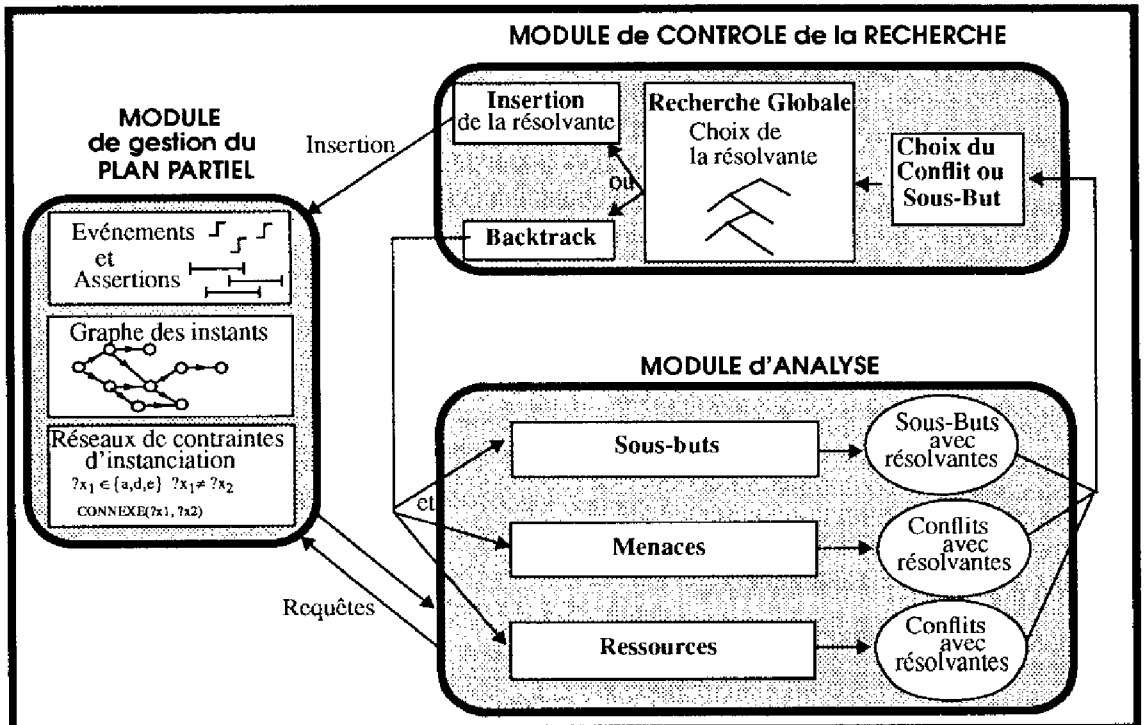


Figure 2.2. Architecture globale du planificateur.

Le module de contrôle est l'élément central, sur lequel est concentré tout l'effort de recherche, par l'intermédiaire d'heuristiques globales appropriées. Si aucune résolvante compatible avec le plan courant n'est trouvée, on effectue un **retour-arrière** sur le module d'analyse, sinon on insère la résolvante au niveau du module de gestion du plan partiel.

Il est un point essentiel qui doit être exposé dès maintenant. La stratégie globale du planificateur IxTeT consiste à remonter toute la complexité de la tâche au module de contrôle, sur lequel sont concentrées un ensemble d'heuristiques, et de se doter par contre de modules associés très performants. C'est pourquoi le module de gestion des contraintes temporelles, que nous allons maintenant détailler, ne prend pas en compte les disjonctions fortes du type "*la proposition P ne doit pas être en recouvrement avec la proposition Q, c'est-à-dire que P [a ∨ b] Q*" (cf § 4.2.3 au chapitre 1). Ce type d'information est traité sous la forme d'un branchement supplémentaire de l'arbre de recherche globale: on effectue un choix d'ordonnancement a priori, sur lequel on prut effectuer un retour-arrière en cas d'échec. Le gestionnaire de contraintes temporelles sera de ce fait limité aux cas polynômiaux.

Pour terminer, signalons que ce planificateur a depuis été enrichi, grâce aux travaux de Ph.Laborie. D'une part, il a intégré dans le planificateur développé par H.Laruelle un module de **gestion des ressources**, qui a été adjoint au module d'analyse (comme l'illustre la figure précédente), permettant de prendre également en compte au niveau du contrôle de la recherche les conflits de ressources (évoqués au début de ce paragraphe), essentiellement dans le cadre des ressources partageables consommables [Laborie94]. D'autre part, nous n'avons parlé dans ce qui précède que d'une hiérarchie de représentation des opérateurs. ce qui ne classe pas IxTeT pour autant dans la catégorie des planificateurs hiérarchiques héritiers de ABSTRIPS et SIPE par exemple. Cette lacune a été comblée par la prise en compte de **niveaux de contrôles hiérarchisés** améliorant sensiblement les performances de la recherche globale [Garcia95].

2.2.4. Un Gestionnaire de Contraintes Temporelles, Pour Quoi Faire ?

Les paragraphes précédents ne se veulent rien d'autre qu'une rapide évocation du contexte d'application de notre gestionnaire de contraintes temporelles, sur lequel nous allons nous concentrer. Ce contexte clairement identifié va nous permettre de mettre en lumière les différentes fonctionnalités dont nous allons avoir besoin. En particulier, si l'on considère ce qui a été dit au paragraphe précédent, on identifie clairement les trois tâches essentielles du module de gestion des contraintes temporelles:

1. **Insertion**: dans le cas où la résolvante est une simple contrainte temporelle, elle est directement placée dans le graphe; dans le cas où il s'agit d'un opérateur, celui-ci, préalablement précompilé, se ramène à un ensemble d'instants et de contraintes qui doivent également être incorporées dans le graphe.

2. **Maintien de cohérence**: une vérification de la cohérence de l'ensemble des contraintes temporelles doit être effectuée à chaque ajout, de manière à rejeter au plus tôt un ajout incohérent, et effectuer ainsi un retour-arrière sur le module d'analyse.

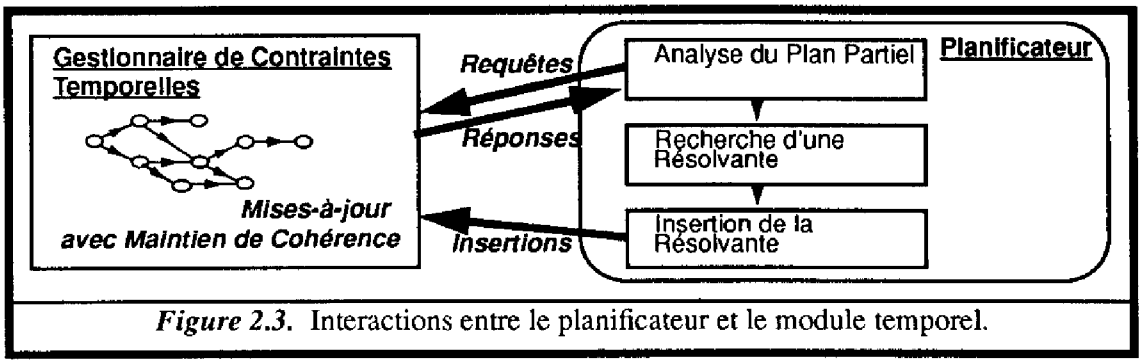


Figure 2.3. Interactions entre le planificateur et le module temporel.

3. **Réponse à une requête**: le module d'analyse cherchant les différentes résolvantes possibles a besoin d'effectuer fréquemment des requêtes simples en direction du graphe temporel, pour connaître le positionnement relatif de divers instants, ou encore des dates ou des durées particulières. Ceci lui permet de voir par exemple s'il n'existe pas déjà un établissement "naturel" d'un sous-but.

Ceci est résumé dans le schéma simplifié ci-dessus. Les deux premiers points conduisent à définir ce que l'on souhaite en termes de **mise-à-jour** du graphe, c'est-à-dire de propagation **incrémentale** (à chaque insertion) des contraintes temporelles. Le troisième point quant à lui doit être précisé. Nous avons dit plus haut qu'il était essentiel que le module d'analyse soit en mesure de détecter les conflits nécessaires. Pour cela, il doit être en mesure de disposer des contraintes minimales en réponse à ses requêtes. Prenons un exemple pour nous en convaincre. Donnons-nous deux assertions HOLD(P_1, $[d_1, f_1]$) et HOLD(P_2, $[d_2, f_2]$). Si nous constatons par exemple que nécessairement $d_1 \leq d_2 \leq f_1$, alors nous en déduirons qu'il y a recouvrement nécessaire entre ces deux occurrences de P_1 et P_2, c'est-à-dire que l'on identifiera un conflit nécessaire. Pour cela, il nous faut bien évidemment pouvoir disposer des contraintes minimales entre d_1 et d_2 et entre d_2 et f_1. Nous parlerons de **requête complète** lorsque la requête, comme nous le souhaitons, retournera la contrainte minimale.

Cette exigence en termes de requêtes complètes est étroitement liée à l'algorithme de propagation que l'on souhaite utiliser. Par exemple, nous avons vu qu'une propagation par 3-cohérence rendait compte du réseau minimal, et permettait donc de disposer de requêtes complètes en temps constant. Une propagation par 2-cohérence sera moins coûteuse, mais ne fournit pas le graphe complet. Elle obligera donc en contrepartie à développer des algorithmes de requêtes plus complexes. Il va s'agir en fait de trouver un **compromis satisfaisant entre mises-à-jour et requêtes**, ce qui sera abordé en détail au paragraphe 2.4.1.

De fait, dans le cadre du planificateur IxTeT, il nous a paru important de signaler dès à présent la proportion de requêtes provenant du planificateur. Appelons *ri* le rapport du nombre de requêtes par rapport au nombre d'opérations d'insertions effectuées par le planificateur durant la construction d'un plan. Nous avons effectué une série de tests portant sur plusieurs applica-

tions habituellement traitées par IxTeT (allant du monde des blocs à des applications réelles telles qu'un chantier de construction impliquant deux robots, ou une mission de prélèvements d'échantillons sur site planétaire), en mesurant *ri*. Nous obtenons en moyenne un rapport de 25 pour 1, pour des plans produits de taille réduite (environ une centaine d'instants). Nous pouvons nous attendre à ce que ce rapport augmente avec la taille *n* du graphe. En effet, le nombre d'insertion est de l'ordre de *n*, alors que le nombre de requêtes augmente de manière quadratique vis-à-vis de *n*. Ceci est à relier à l'augmentation du nombre de résolvantes possibles pour un sous-but ou un conflit donné, lorsque *n* augmente.

En conclusion, nous devons retenir que les requêtes représentent l'opération critique, et doivent donc être privilégiées par rapport aux mises-à-jour, pour ce qui est de leur efficacité.

2.3. Une Gestion Efficace des Contraintes Symboliques

Signalons tout d'abord qu'une description plus détaillée des aspects abordés dans les paragraphes qui suivent peut être consultée dans [Mounir90].

2.3.1. L'Algèbre d'Instants Continue Revisitée

Nous avons souhaité pour des raisons de complexité nous limiter à l'algèbre d'instants réduite PA. En effet, en planification, il est rarement utile d'imposer que deux instants soient disjoints. On s'intéresse le plus souvent à de simples relations de précédence. Le fait que ces précédences soient strictes ou non est de peu d'intérêt, et rejoint en fait la polémique dont nous avons parlé au sujet de l'instant divisé. Néanmoins, dans un souci de complétude, nous autorisons une prise en compte spécifique des inégalités $\neq$, dans une structure séparée du réseau de contraintes symboliques. Concentrons-nous tout d'abord sur l'algèbre PA.

A priori, PA=$\{ ?, <, \leq, =, \geq, > \}$. En fait, les relations de stricte précédence $<$ et $>$ impliquent une inégalité $\neq$ entre les instants considérés. A partir du moment où l'on désire considérer à part ces inégalités, il n'est pas utile de représenter explicitement ces relations strictes. Par ailleurs, les instants étant des primitives purement temporelles, découplées de leur sémantique propre, deux instants égaux seront immédiatement **agrégés** en un seul objet dans notre réseau. Celui-ci aura donc besoin de représenter uniquement les relations $\{ ?, \leq, \geq \}$ de manière explicite. Nous définissons donc deux structures, qui sont

1. Le **réseau symbolique** $\mathcal{S} = \{ \mathbf{V}, \mathbf{R_s} \}$, avec:
   - $\mathbf{V}$ = l'ensemble des instants,
   - $\mathbf{R_s} = \{ [i\ r\ j] / (i,j) \in \mathrm{V}^2$, et $r \in \{ ?, \leq, \geq \} \}$.

2. L'**ensemble des inégalités** $I = \{\mathbf{V_i}, \mathbf{R_i}\}$, avec:
   - $\mathbf{V_i} \subset \mathbf{V}$
   - $\mathbf{R_i} = \{\ [i \neq j] / (i,j) \in \mathbf{V_i}^2\}$.

Remarquons au passage que I est un ensemble vis-à-vis duquel seules des opérations d'ajout et des requêtes seront effectuées. Aucune propagation n'aura lieu d'être déclenchée dans I. Rappelons en effet que la relation $\neq$ n'est pas transitive: $(i \neq j \wedge j \neq k)$ n'implique pas $i \neq k$. La prise en compte des inégalités va par contre influer, comme nous allons le voir, sur les mécanismes de propagation dans $\mathcal{S}$.

La représentation est complète en entrée dans le sens où $i = j$ signifiera tout simplement que i et j sont une seule et même variable de **V**, et $i < j$ par exemple pourra s'exprimer par $i \leq j$ dans $\mathcal{S}$ et $i \neq j$ dans I. En résumé, nous allons nous focaliser sur un réseau symbolique simplifié, dans lequel des algorithmes efficaces de propagation pourront être développés, prenant en compte le cas des inégalités uniquement lorsque cela est nécessaire. Il restera à prouver que les algorithmes obtenus sont complets.

Notons pour finir que $\mathcal{S}$ et I, quoique virtuellement séparés, sont en fait définis à l'intérieur d'un même réseau d'instants réel.

2.3.2. L'Arbre de Recouvrement Maximal Indexé

Nous pouvons aller encore un peu plus loin, en remarquant que $(i \geq j) \Leftrightarrow (j \leq i)$, c'est-à-dire que l'on peut se restreindre, en termes de représentation minimale, aux seules relations **?** et $\leq$. En d'autres termes, deux instants sont soit non contraints, soit reliés par une relation de précédence, qui est par nature une relation d'ordre. Nous nous sommes donc ramenés à la représentation d'une relation d'ordre partiel sur un ensemble **V**. Ceci peut être représenté sous la forme habituelle d'un graphe orienté. Ce graphe a la particularité d'être sans circuits et de disposer d'un noeud initial 0, qui représente l'origine des temps (ce qui permet de représenter, comme nous le verrons plus loin, des dates absolues sous la forme de contraintes numériques entre 0 et un instant i). Un arc orienté entre deux noeuds-instants représentera donc une précédence, l'absence d'arc signifiera l'ignorance **?** (c'est-à-dire l'absence de contrainte). La figure suivante représente un tel réseau.

En ce qui concerne le problème central du compromis entre requêtes et mises-à-jour, remarquons que si l'on se contente du graphe d'entrée, sans rendre explicites toutes les contraintes symboliques induites, il va falloir pour répondre à une requête symbolique (*"quelle est la relation entre i et j ?"*) effectuer des opérations de parcours de graphe. Les résultats de la théorie des graphes nous permettent d'extraire un **arbre de recouvrement maximal** de ce graphe, et

de séparer du même coup les relations qui font partie d'une "branche" de l'arbre et celles qui relient deux branches distinctes. Nous qualifierons ces dernière de relations "résiduelles". La structure globale se définit alors comme un ensemble d'instants i disposant des champs suivants:

- le **père** de i dans l'arbre,
- l'ensemble des **fils** de i dans l'arbre,
- l'ensemble des successeurs de i par des **relations résiduelles**,
- un **index**,
- un **rang**.

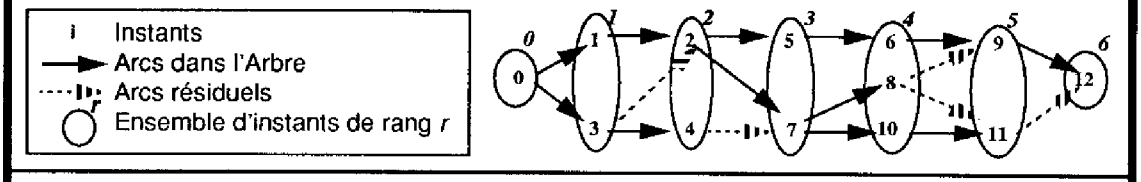


Figure 2.4. Le réseau symbolique $\mathcal{S}$.

L'index et le rang vont permettre d'obtenir des requêtes très efficace. L'indexation est une méthode de numérotation des instants en fonction de la branche à laquelle ils appartiennent. En comparant directement les index (en temps quasiment constant), on peut donc savoir si deux instants sont sur une même branche, ce qui établit du même coup une précédence entre eux. Pour ce qui est du rang, il se définit comme la longueur du plus court chemin de 0 à i dans l'arbre. La structure de rang est représentée dans la figure ci-dessus. Elle rend compte de propriétés nous permettant d'effectuer un premier test de la manière suivante:

- Si $rang(i) = rang(j)$ alors i **?** j
- Si $rang(i) < rang(j)$ alors $(i$ **?** $j \vee i \leq j)$

Cette caractérisation du rang nous sera utile dans le cadre des algorithmes présentés dans le prochain chapitre. Au niveau symbolique, l'observation des rangs nous autorise à diriger la recherche. Par exemple, si $rang(i) < rang(j)$, nous allons chercher à établir que $i \leq j$. L'index quant à lui permet une recherche immédiate dans l'arbre. Si i et j ne sont pas sur une même branche, alors et alors seulement nous serons amenés à chercher un chemin de i vers j. Nous obtenons en définitive un algorithme de réponse à une requête symbolique Compare fonctionnant en moyenne en $O(n)$, où n est la taille de **V**. Le recours à cette structure d'arbre de recouvrement maximal indexé nécessite son maintien, c'est-à-dire sa mise-à-jour à chaque ajout, mais aussi à chaque retrait (ce qui témoigne d'un fonctionnement non-monotone). Des algorithmes AddRelation et RemoveRelation ont été développés à cet effet. Ils sont également linéaires par rapport au nombre d'instants. Le lecteur pourra trouver les détails de ces algorithmes dans [Mounir90].

En résumé, plutôt que de chercher le réseau minimal par l'intermédiaire d'un algorithme de cohérence de chemin relativement coûteux, qui permet en retour des requêtes en temps constant, nous avons pu nous ramener dans le cas des contraintes symboliques à un bien meilleur compromis, les **mises-à-jour** tout comme les **requêtes** témoignant d'une **complexité linéaire**. Il reste à étudier l'influence que peuvent avoir les contraintes d'inégalité dans cette méthode.

2.3.3. Agrégation d'Instants et Prise en Compte des Inégalités

[Mounir90] démontre que le fait de ne pas tenir compte des inégalités n'induit aucune incomplétude des algorithmes de propagation (c'est-à-dire qu'il ne risque pas d'y avoir d'incohérence non détectée) à une condition: qu'il n'y ait aucun circuit dans le graphe $\mathcal{S}$. En effet, le seul cas d'incohérence apparaît lorsque

$i \leq j \wedge j \leq i \wedge i \neq j$.

Ces cas ne sont susceptibles d'apparaître que lorsqu'un circuit est détecté dans le graphe $\mathcal{S}$. Rappelons que lorsque deux instants sont égaux, il convient de les agréger. Il en va de même en présence d'un circuit:

si $i_1 \leq i_2 \leq \ldots i_k \leq i_1$,
alors $i_1 = i_2 = \ldots i_k$

Un algorithme récursif de réduction d'un circuit à un instant unique a été mis au point, qui dans le même temps vérifie qu'il n'y a pas d'inégalité dans le circuit, par simple interrogation directe de la structure I:

$\forall\, i_c, i_c$, appartenant au circuit, est-ce que $(i_c \neq i_c) \in \mathbf{R_i}$?

Ce simple algorithme d'agrégation Collapse permet de détecter toute incohérence pouvant être induite par une inégalité. Cet algorithme, ajouté aux algorithmes de mise-à-jour dans $\mathcal{S}$, conduit en pire cas à retrouver la complexité de l'algorithme de van Beek en $O(n^4)$ pour l'algèbre complète d'instants $PA^{\neq}$. En fait, si l'on tient compte du fait qu'il est très rare de voir apparaître un circuit dans un réseau symbolique $\mathcal{S}$, il s'avère en pratique que la complexité supplémentaire induite est tout-à-fait négligeable, alors même qu'en termes d'expressivité et de complétude des algorithmes, notre méthode permet d'appréhender l'algèbre complète $PA^{\neq}$. Notons par contre que cette approche induit une forme d'incomplétude quant aux requêtes: il n'est pas possible de savoir directement si deux instants quelconques sont différents. Par exemple, si $i \leq j$ et $j < k$ (c'est-à-dire $j \leq k$ et $j \neq k$), alors $i < k$ mais une interrogation directe de $\mathcal{S}$ et I ne permet pas de rendre compte de $i \neq k$. Nous aurons simplement $i \leq k$, ce qui est suffisant dans le cadre de nos applications. Retenons donc simplement que **la propagation est complète vis-à-vis de $PA^{\neq}$**, alors que **les requêtes ne sont complète que vis-à-vis de PA.**

2.3.4. Travaux Comparables

Signalons à titre de comparaison le travail de A.Gerevini et L.Schubert [Gerevini93], qui ont redémontré le résultat de P.vanBeek sur l'équivalence entre PA et PIA (cf chapitre 1), dont la preuve était incorrecte. Ils ont surtout construit une structure qui est comparable à la nôtre, extrayant du graphe initial un ensemble de *chaînes*, reliées par des "*meta-contraintes*". Ils utilisent une propriété similaire à notre notion de rang. La différence fondamentale est qu'ils intègrent la relation $\neq$ dans leur modèle, aboutissant à une gestion efficace et complète, tant en termes de requêtes que de maintien de cohérence, de l'algèbre complète d'instants $PA^{\neq}$. Ils sont notamment capables de déterminer la contrainte minimale entre deux instants dans $PA^{\neq}$, en particulier de différencier une précédence stricte ($<$) d'une précédence non stricte ($\leq$). Cette complétude de la représentation les conduit à des complexités légèrement supérieures aux nôtres, mais très nettement inférieures à l'algorithme en $O(n^4)$ de van Beek. Dans les cas où le nombre de relations $\neq$ est très faible, nos deux systèmes témoignent de performances comparables.

On peut également rappeler le travail de J.Dorn, évoqué au chapitre 1, qui dans le cadre de l'algèbre d'intervalles a développé des algorithmes complets et plus efficaces que ceux de Allen dans le cas de réseaux fortement séquentiels [Dorn92]. Il définit une notion de chaînes, et n'explicite pas les relations transitives de stricte précédence entre intervalles, celles-ci pouvant être retrouvées par l'intermédiaire de parcours de ces chaînes. Cette approche est comparable à la nôtre dans le sens où elle cherche à établir un bon compromis entre mises-à-jour et requêtes.

2.4. Vers une Prise en Compte des Contraintes Numériques

Tout ce qui précède nous a permis de situer le cadre du travail présenté dans ce mémoire. Nous partons donc du système IxTeT tel qu'il vient d'être présenté, avec un objectif bien précis: faire en sorte de pouvoir également prendre en compte des **dates et durées imprécises**, de manière à lui donner toute l'expressivité que l'on souhaite trouver dans un planificateur temporel. Par exemple, nous voulons pouvoir spécifier la durée d'une tâche comme étant de (1) *10 à 20 mns*, ou bien dire qu'un événement est attendu (2) *dans environ 5 mns, à une mn près*, ou encore que le but assigné (3) doit être établi *entre 15h et 15h30* et (4) doit persister *pendant au moins 20 mns*. Au vu de la structure symbolique qui était la nôtre, il nous est apparu que le paradigme des STPs constituait l'approche idéale. En effet:

1. La forme des **contraintes binaires** des STPs coïncide avec les notions de dates et de durées imprécises que nous souhaitons représenter. Les exemples ci-dessus donnent les intervalles numériques suivant, en considérant la minute comme l'unité de temps par défaut:

(1) [10, 20]
(2) [4, 6]
(3) [15.00, 15.30]
(4) [20, +∞[

2. Le réseau de type STP s'appuie sur le **même ensemble d'instants** que le réseau $\mathcal{S}$,
3. Les STPs permettent l'utilisation d'algorithmes de mise-à-jour **complets** et **polynômiaux.**

Du coup, nous pouvons définir un nouveau **réseau numérique** répondant aux spécifications des STPs (en confondant les instants avec les dates de ceux-ci, cf chapitre 1, § 4.3.2), et qui s'écrit $\mathcal{G} = \{ \mathbf{V}, \mathbf{R_g} \}$, avec:

- $\mathbf{R_g} = \{ C_{ij}=[inf_{ij}, sup_{ij}] / (i,j) \in V^2 \}$.

C'est-à-dire que nous allons pouvoir construire ce réseau par ajouts incrémentaux de contraintes numériques C_{ij}, qui seront propagées, avec un souci constant de maintien de la cohérence. Se pose alors la question de l'algorithme de propagation à utiliser, c'est-à-dire encore une fois du compromis entre requêtes et mises à jour.

2.4.1. Les Contraintes Minimales: une Exigence Coûteuse

Reprenons ici notre discussion au sujet du compromis entre propagation complète (c'est-à-dire avec maintien de cohérence) et requêtes complètes (c'est-à-dire obtention des contraintes minimales). Une contrainte numérique minimale correspond à l'intervalle de durées possibles restreint aux seules valeurs susceptibles de figurer dans une solution. Nous avons vu qu'au niveau symbolique, il était indispensable de disposer des contraintes minimales. Il en est de même en ce qui concerne les contraintes numériques, et pour au moins une raison essentielle, comme nous allons le voir: les contraintes numériques minimales influent sur les contraintes symboliques minimales.

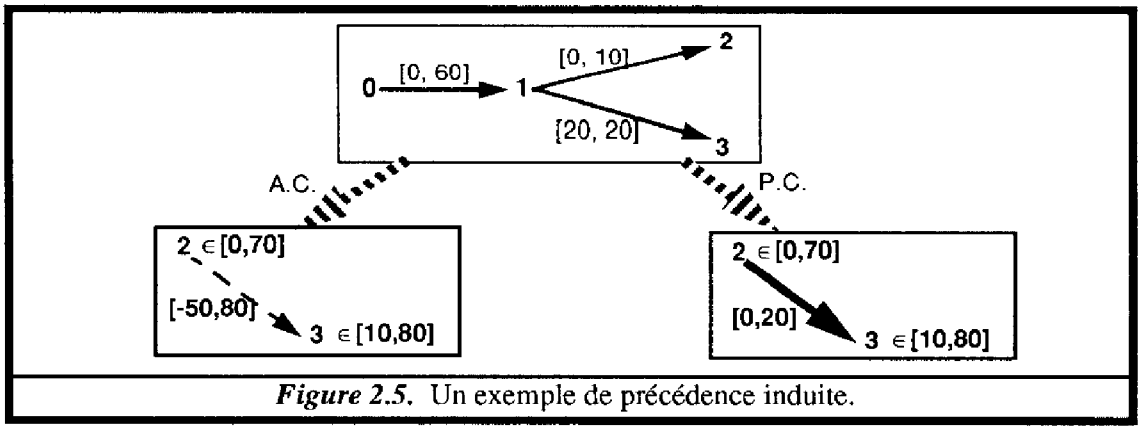


Figure 2.5. Un exemple de précédence induite.

En effet, nous avons vu que les contraintes symboliques pouvaient être retrouvées à l'intérieur du réseau de contraintes numériques. Cette propriété en induit une autre, plus générale: la propagation numérique restreignant les contraintes, elle peut faire apparaître par exemple **de nouvelles précédences.**

Considérons pour nous en convaincre l'exemple de la figure précédente. Nous lui avons appliqué les deux algorithmes de filtrages présentés dans [Mackworth77], par cohérence d'arc d'une part, et par cohérence de chemin d'autre part. Ces deux algorithmes permettent tous deux de vérifier la cohérence d'un STP, du fait de la propriété de décomposabilité qui leur est propre [Dechter91]. Nous voyons que le deuxième algorithme, qui rend compte des contraintes numériques minimales, fait apparaître du même coup une précédence entre les instants 2 et 3, au contraire du premier. En fait, les contraintes numériques données en entrée induisent une précédence nécessaire entre 2 et 3. Cette précédence doit pouvoir être établie dans le cadre d'une requête symbolique.

Définition 2.1 : Une **précédence nécessaire** $i \leq j$ est induite par le réseau numérique ssi le domaine de C_{ij} peut être restreint à $[inf_{ij}, sup_{ij}] \subseteq [0, +\infty[$.

Par la suite, nous emploierons plus simplement le terme de **précédence induite**. En conclusion, pour être certain de récupérer toutes les précédences induites, il est nécessaire de disposer des contraintes minimales dans $\mathcal{G}$. Bien entendu, ce résultat s'applique également au cas d'égalités ou d'inégalités induites. Nous reviendrons sur le problème de l'identification des contraintes symboliques induites dans le cadre de notre méthode dans le chapitre 3.

Le problème se pose en fait dans les mêmes termes que dans le cas strictement symbolique: devons-nous maintenir le réseau minimal, par l'intermédiaire d'un algorithme relativement coûteux, et disposer ainsi des contraintes minimales en temps constant, ou devons-nous essayer de trouver un compromis entre requêtes plus complexes et mises-à-jour moins coûteuses ? Malheureusement, dans le cas numérique, un tel compromis paraît beaucoup plus délicat à établir. En effet, pour établir la contrainte symbolique entre deux instants i et j, il peut donc s'avérer nécessaire de rechercher la contrainte minimale dans $\mathcal{G}$. En l'absence du réseau minimal, cela nécessite de chercher **tous** les chemins numériques possibles de i à j [Barber93], induisant donc une complexité en pire cas en $O(n.e)$, où e est le nombre de contraintes explicites dans le réseau. Ce n'est qu'à ce prix-là que l'on pourrait répondre à notre exigence de requêtes complètes. Deux possibilités nous sont donc offertes, comme cela est illustré dans [Barber93]:

1. Chercher le compromis, ce qui conduit à des algorithmes de mise-à-jour en $O(n)$ pour le cas symbolique et en $O(n.e)$ pour le cas numérique, mais des requêtes aussi bien au niveau symbolique que numérique en $O(n.e)$ en pire cas.

2. Privilégier l'efficacité des requêtes, ce qui nous donne des algorithmes de mise-à-jour en $O(n)$ pour le cas symbolique et en $O(n^2)$ (à chaque étape, voir au paragraphe suivant) pour le cas numérique, mais des requêtes linéaires dans le cas symbolique et en temps constant dans le cas numérique.

Nous reviendrons là-dessus lorsque nous confronterons notre méthode aux autres approches existantes, à la fin du chapitre 3. Rappelons en ce qui nous concerne que la proportion de requêtes par rapport aux mises-à-jour dans un système de planification est relativement importante, et que donc les requêtes doivent être privilégiées. Nous avons donc choisi d'accepter le prix à payer au niveau des mises-à-jour, contre l'assurance de disposer de requêtes complètes efficaces. Nous maintiendrons donc un **réseau numérique minimal** au moyen d'un **algorithme de cohérence de chemin**.

2.4.2. La Volonté de Conserver une Représentation Hétérogène

Le problème est que le réseau minimal ne peut être obtenu en temps polynômial que dans le cas des STPs [Dechter91], ce qui constitue une limitation peu contraignante, d'autant plus que les disjonctions numériques, comme dans le cas des disjonctions fortes symboliques, peuvent être remontées au niveau de l'algorithme de recherche globale du planificateur. Par ailleurs, l'obtention du réseau minimal nécessite un algorithme de propagation par cohérence de chemin. Celui-ci détient une complexité cubique dans le cas d'une **propagation globale** a posteriori (c'est-à-dire après ajout de l'ensemble des contraintes). Pour être précis, on obtient du $O(n^3.a^3)$, où a est la longueur des intervalles numériques, et n la taille du réseau. Le paramètre a étant borné pour une application donnée, on aboutit à du $O(n^3)$. Dans le cas **incrémental**, on est ramené à une complexité en $O(n^2)$ à chaque étape.

PROPAGER (E_{init}) **MAJ** <- E_{init}. *Tant Que* **MAJ** $\neq \varnothing$ *Faire* *Extraire le premier élément de* **MAJ** : (i, j). *Pour* $k = 1$ *à* $n, k \neq i,j$ *Faire* *Si* *MODIFIE* (i, k, j) *Alors* **MAJ** <- **MAJ** $\cup$ (i, k). *Si* *MODIFIE* (k, j, i) *Alors* **MAJ** <- **MAJ** $\cup$ (k, j).	*MODIFIE* (i, j, k) C_{ij} <- $C_{ij} \cap (C_{ik} \circ C_{kj})$. *Si* C_{ij} *modifié Alors* *Si* C_{ij} *tel que min>max* *Alors Sortir (INCOHERENT)* *Sinon retourner (OUI)* *Sinon retourner (NON)*

Figure 2.6. Algorithme PC-2.

Nous rappelons ci-dessus l'algorithme **PC-2** [Mackworth85], qui est la version incrémentale, donc applicable dans notre cas, du filtrage par cohérence de chemin. E_{init} représente

l'ensemble initial des contraintes insérées, qu'il est nécessaire de propager. ***MAJ*** représente l'ensemble des contraintes modifiées en cours de propagation.

N'oublions pas que le réseau symbolique est désormais inutile. Néanmoins, devant la différence sensible de performance des mises-à-jour dans $\mathcal{S}$ et dans $\mathcal{G}$, il paraît plus que souhaitable de **conserver les deux structures**, de manière à profiter d'une gestion symbolique efficace.

Il n'en reste pas moins que la complexité cubique de la propagation numérique constitue une diminution considérable des performances de notre module temporel. Le chapitre 3 va s'employer à trouver une porte de sortie acceptable face à cette impasse. C'est dans ce chapitre que nous allons être amenés à définir formellement l'architecture globale de notre gestionnaire de contraintes temporelles hétérogènes, avec l'ensemble des interactions que cela suppose.

2.4.3. Où Certaines Contraintes ne sont pas Libres ...

Les STPs ont par ailleurs un défaut important en termes d'expressivité. Si l'on reprend les 4 petits exemples numériques donnés au début du paragraphe 2.4, il apparaît que le (3) et le (4) représentent ce que l'on souhaite réaliser au niveau d'un but, qui est par nature contrôlable, c'est-à-dire que l'on peut choisir sa date effective de réalisation, entre 15h et 15h30, ainsi que sa durée effective de persistance. Par contre, (1) est lié à une tâche dont je ne contrôle pas la durée, et (2) se rattache à un événement qui lui aussi arrivera quand il arrivera ... Ces distinctions doivent être reliées à la typologie des attributs présentée au niveau du planificateur (cf § 2.2.2): les cas (3) et (4) représentent des assertions **contrôlables**, les cas (1) et (2) induisant plutôt une notion de **contingence**.

Ceci se traduit dans le STP par des contraintes que nous qualifierons également de contingentes, et d'autres que nous qualifierons de "libres". Le problème est que le paradigme des CSPs suppose implicitement que toutes les contraintes sont libres, c'est-à-dire que lors de la détermination d'une solution, on est libre de fixer la durée ou la date effective pour chaque contrainte. En fait, la notion de contingence induit une prise en compte d'un **niveau d'incertitude** traditionnellement absent du paradigme classique des CSPs.

En conclusion, si l'on veut pouvoir tenir compte de la dichotomie qui vient d'être rapidement suggérée, il va falloir adapter notre modèle pour accepter également ces contraintes qui ne sont pas aussi libres que les autres ... Nous verrons ces aspects en détail au chapitre 4.

3. Conclusion

Nous avons pu voir dans ce chapitre que notre gestionnaire de contraintes temporelles se présentait comme un module séparé, intégré dans une architecture globale de planification. Nous avons souligné la volonté de concentrer toute la combinatoire du problème de planification dans le module de gestion de l'arbre de recherche globale. Cela nous amène à restreindre le module temporel aux seules **représentations polynômiales**, c'est-à-dire l'algèbre d'instants PA au niveau symbolique, et les STPs au niveau numérique, les contraintes d'inégalité $\neq$ étant gérées dans une structure séparée.

Le lien avec le planificateur se retrouve au niveau de deux principaux types d'opérations, qui sont d'une part les **mises-à-jour** (c'est-à-dire insertion et maintien de cohérence), et d'autre part les opérations d'interrogation sous la forme de **requêtes simples**. Une exigence de **complétude** est rattachée à l'un et l'autre de ces deux types d'opérations. Le problème se situe au niveau d'une recherche de **compromis** entre les deux. En effet, plus l'algorithme de propagation choisi sera coûteux, plus les requêtes seront efficaces, et inversement. Au niveau symbolique, il a été possible de mettre en avant un compromis satisfaisant, grâce à une technique complète assurant mises-à-jour et requêtes en temps linéaire. Par contre, au niveau numérique, nous avons été obligés de privilégier l'un ou l'autre. Le choix s'est porté sur un maintien du réseau minimal en $O(n^2)$ à chaque étape, lequel permet en retour d'obtenir des requêtes en temps constant. Celles-ci sont en effet sollicitées avec une fréquence nettement supérieure dans les applications de planification.

Par ailleurs. un des principaux objectifs de notre gestionnaire de contraintes temporelles est de fournir également un bon **compromis entre expressivité et efficacité**. Au niveau de l'efficacité, la nécessité de recourir à un algorithme relativement coûteux de propagation numérique n'est pas pleinement satisfaisant. C'est pourquoi il paraît en premier lieu nécessaire de conserver les trois structures distinctes que sont $\mathcal{S}$, $\mathcal{I}$ et $\mathcal{G}$, pour bénéficier de l'efficacité de la gestion strictement symbolique. Ceci nous conduit à disposer d'une **représentation hétérogène** de contraintes temporelles, reposant néanmoins sur un même réseau d'instants réel. En deuxième lieu, nous allons voir que la spécificité du domaine de la planification nous permet de restreindre le graphe numérique à un sous-graphe, dans lequel la propagation coûteuse pourra être cantonnée. Ceci fait l'objet du chapitre 3.

En ce qui concerne l'expressivité, il s'est avéré nécessaire de distinguer des contraintes libres et des contraintes contingentes, de manière à prendre en compte l'**incertitude temporelle** inhérente aux applications réelles de planification. Cette extension de la représentation sera étudiée en détail au chapitre 4.

Chapitre III. Se Restreindre à un Sous-Graphe Numérique en Planification

1. Un Petit Exemple Illustratif

Histoire de présenter les choses avec un maximum de clarté, nous avons choisi de considérer dès le départ un exemple relativement simple, qui nous permettra d'illustrer au fur et à mesure les concepts abordés.

Imaginons donc un employé de bureau (un chercheur par exemple ...), que nous appellerons Albert (celui-là même dont les vacances et la belle-mère ont été évoquées ici et là au chapitre 1, ce qui n'aura certes pas échappé au lecteur perspicace). C'est la fin de la journée, et Albert est assez pressé car il doit aller chercher ses enfants à l'école et, évidemment, il est déjà légèrement en retard. Il lui faut avant de partir demander à sa secrétaire, qui se trouve avec lui dans son bureau, de lui réserver un vol sur Paris pour la fin de la semaine (il vient juste de se rappeler de ce déplacement, et de réaliser qu'il serait sans doute temps de réserver sa place ...) Nous appellerons cette tâche *Délivrer_Message (Albert, secrétaire)*. Selon le débit de parole qu'il adoptera, cette tâche lui prendra entre 30 secondes et 1 minute. A ce moment, Albert constate à travers la porte ouverte de son bureau qu'une de ses collègues est en train d'appeler l'ascenseur, qui se trouve tout au bout d'un assez long couloir. Il sait que s'il ne profite pas de l'occasion, il va devoir attendre le retour de l'ascenseur, et perdre du même coup un temps précieux. Il décide donc de saisir sa chance ! Il sait d'autre part que la porte de l'ascenseur est susceptible de s'ouvrir dans les 10 secondes qui suivent, suivant l'étage d'où provient l'ascenseur, et qu'il dispose ensuite de 30 secondes précisément avant que la porte ne se soit refermée. Nous avons donc là deux événements que nous appellerons *Ouverture (ascenseur)* et *Fermeture (ascenseur)*. Enfin, il sait également que, suivant l'allure à laquelle il se déplacera, il lui faudra entre 20 et 40 secondes pour atteindre l'ascenseur. Nous appellerons cette tâche *Aller_A (Albert, bureau, ascenseur)*.

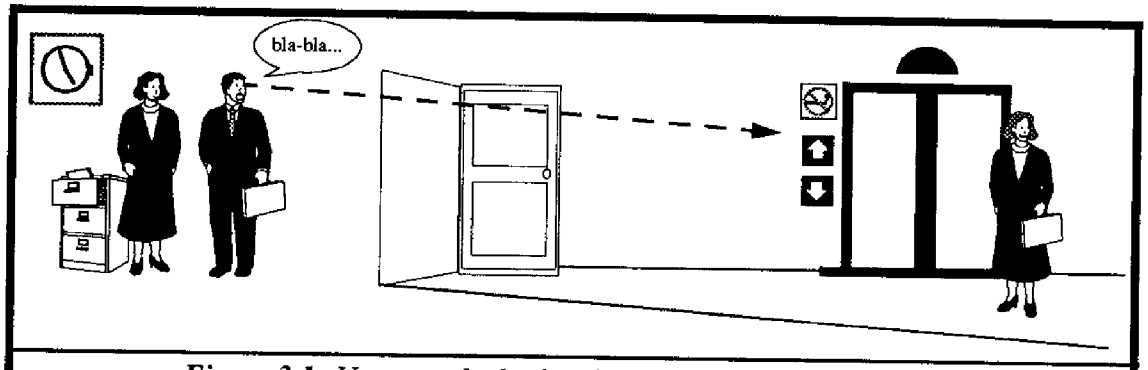


Figure 3.1. Un exemple de situation temporellement contrainte.

Maintenant, si l'on considère cet exemple sous l'angle de la planification, on conviendra aisément que le problème peut être décrit par l'intermédiaire d'une situation initiale où les

positions des trois protagonistes sont connues, ainsi que les événements datés *Ouverture (ascenseur)* et *Fermeture (ascenseur)*, et d'un ensemble de deux buts à résoudre qui sont *Message-Délivré* et *Dans (Albert)=ascenseur*. On dispose également des tâches *Délivrer_Message (Albert, secrétaire)* et *Aller_A (Albert, bureau, ascenseur)*, qui vont pouvoir être insérées l'une après l'autre dans le plan initial pour résoudre les buts sus-nommés. La figure 3.2 ci-dessous permet de visualiser les représentations temporelles de ces tâches, où figurent également leurs effets. On constate notamment que pour que la tâche *Aller_A (Albert, bureau, ascenseur)* ait lieu, il faut qu'Albert soit dans le bureau. A la fin de la tâche, il est entré dans l'ascenseur, à condition que la porte de l'ascenseur ait été ouverte. La tâche *Délivrer_Message (Albert, secrétaire)* quant à elle nécessite la présence dans le bureau de la secrétaire, laquelle ne peut se déplacer et noter le message en même temps, alors qu'Albert peut se déplacer tout en délivrant son message, à condition de rester sur l'étage. La secrétaire aura terminé de noter le message à un instant indéterminé après la fin de la tâche.

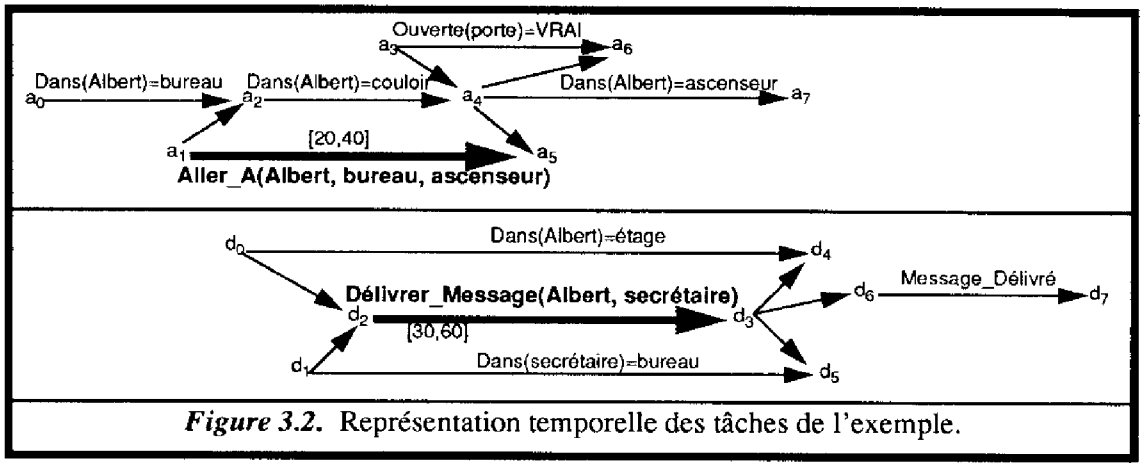


Figure 3.2. Représentation temporelle des tâches de l'exemple.

Dès lors, au plan final, dans lequel auront été successivement insérées les deux tâches susnommées, sera associé le graphe temporel de la figure 3.3, où les arcs valués d'une contrainte numérique explicite sont représentés par une flèche de largeur supérieure, et où le lecteur pourra retrouver la signification de chaque instant. On peut remarquer que *Dans(Albert)= étage* est un effet secondaire (cf chapitre 2, § 1.1), sur lequel un raisonnement approprié a conduit à l'ajout d'une contrainte entre les instants 11 et 2: Albert doit avoir fini de parler avant que l'ascenseur ne se soit refermé.

Remarquons pour finir que notre exemple est très contraint et concerne une situation à planifier dans un futur immédiat. Il relève donc plus a priori de techniques réactives. En fait, ce qui est important, c'est qu'il illustre de manière très réaliste une situation mettant en jeu des relations temporelles complexes, ce qui permettra de montrer clairement l'efficacité de notre méthode dans de telles situations. Rien n'empêche en fait d'imaginer une tout autre application à laquelle le graphe pourrait aussi bien correspondre, avec une échelle temporelle différente.

Par exemple, on peut imaginer un robot devant envoyer des données à une station, et en même temps poser un objet sur un tapis roulant qui doit s'immobiliser devant lui pendant quelques instants. On aboutirait en effet à un graphe similaire à celui figurant ci-dessous. L'exemple concret ne sert de toutes façons qu'à fixer les idées, les raisonnements qui suivent étant essentiellement illustrés par le seul graphe d'instants, quelle que soit la signification de ceux-ci.

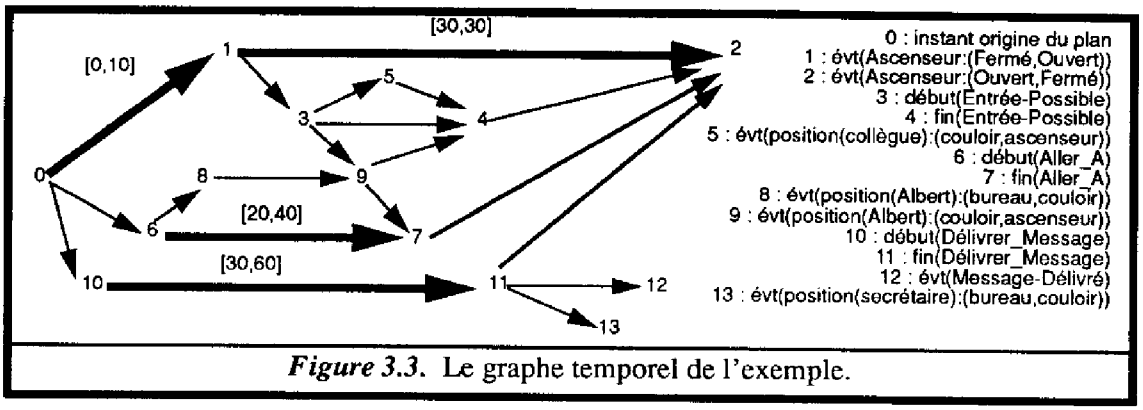


Figure 3.3. Le graphe temporel de l'exemple.

2. Principe Général

Le chapitre précédent concluait sur une note plutôt pessimiste: il semble en effet difficile de se passer du graphe numérique complet, et donc de la complexité en $O(n^3)$ des algorithmes de propagation correspondants, si l'on veut pouvoir facilement et correctement répondre à une requête temporelle provenant du planificateur. De toute évidence, nos exigences en matière de complétude coûtent cher; un compromis raisonnable serait donc hors de portée ? La section suivante va heureusement démontrer le contraire. En effet, s'il n'est pas envisageable de réduire la dimension cubique inhérente aux algorithmes de cohérence de chemin, rien ne nous empêche par contre de jouer sur le paramètre n ...

2.1. La Proportion de Contraintes Numériques en Planification

Rappelons que pour le gestionnaire de contraintes temporelles, une contrainte de type durée ou date est "anonyme", c'est-à-dire que l'on ne connaît pas sa signification logique. Elles peuvent donc affecter a priori aussi bien des couples de points correspondant à des tâches, ou à des postconditions ou préconditions de celles-ci. Nous utiliserons dans ce paragraphe le terme d'effets de manière générique, pour désigner à la fois postconditions et préconditions, ceci pour simplifier le propos.

En fait, la tâche constitue indubitablement un acte clairement identifié, exécuté par l'acteur pour lequel on planifie. De ce fait, il est possible et même nécessaire de l'inscrire dans un laps de temps, pas forcément précisément connu à l'avance, mais tout au moins borné par définition. Il n'en est pas forcément de même pour ses effets, lesquels représentent des conditions nécessaires ou au contraire des conséquences de la tâche, et ne sont utilisés par le planificateur que comme éléments de synchronisation. En d'autres termes, ils servent simplement à imposer des contraintes d'ordonnancement entre les diverses tâches.

Tout ça pour en arriver où ? Tout simplement au fait qu'un effet se positionne dans le temps **relativement** à une tâche, et non dans un laps de temps prédéfini. Nous allons d'abord illustrer cela par un exemple très simple issu d'applications habituellement traitées par IxTeT, essentiellement l'application du chantier présentée dans [Laruelle94], après quoi nous reviendrons sur notre exemple introductif. Tout ceci nous permettra de tirer les conclusions qui s'imposent.

Analyse au travers d'un exemple applicatif réel

Nous nommerons tout d'abord intuitivement, avant de le présenter plus formellement plus loin, "**instant numérique**" un instant début ou fin d'une contrainte numérique. Considérons le macro-opérateur *Déplacer-Objet* présenté au § 2.2.2 du chapitre 2. Celui-ci est composé de trois tâches *Saisir*, *Poser*, et une tâche de déplacement entre les deux. Nous avons donné la représentation de la tâche *Saisir* (cf figure 2.1), où seule la tâche elle-même est affectée d'une durée. L'instant intermédiaire t_1, lié à ses effets, est simplement placé symboliquement entre *début* et *fin*. Ces deux derniers sont donc numériques, alors que t_1 est un instant "**non numérique**" a priori. La tâche *Poser* se définit de manière symétrique. Ces deux tâches supposent donc l'ajout dans le graphe de six instants, quatre instants numériques correspondant aux débuts et fins des tâches, et deux instants supplémentaires, non numériques, liés aux effets des tâches.

Pour ce qui est de la tâche de déplacement, de la même manière, seul le couple d'instant correspondant à la tâche elle-même est affecté d'une contrainte de durée. Pour ce qui est des effets liés à la tâche, tout dépend du lieu de départ et du lieu d'arrivée. Considérons tour à tour trois cas de figures.

1. Si l'objet reste dans la même pièce, alors nous utiliserons la tâche élémentaire *Aller-à*, qui ne nécessite la prise en compte d'aucun instant supplémentaire lié aux effets. On obtient alors en tout **six** instants "numériques" (les instants de débuts et de fins des trois tâches) et **deux** instants supplémentaires non numériques.

2. L'objet doit changer de pièce sur un même étage. Alors nous utiliserons une tâche *Changer-Pièce* qui dispose d'effets supplémentaires et par suite nécessite l'utilisation de deux nouveaux instants, le premier correspondant au moment où le robot sort de la première pièce, et

le deuxième le moment où le robot entre dans la deuxième pièce. Encore une fois, ces deux instants n'ont pas besoin d'être précisément situés dans le temps. Ils ne sont donc affectés d'aucune contrainte numérique. Nous obtenons donc au total **six** instants numériques pour **quatre** instants non numériques.

3. L'objet doit changer d'étage. Nous utilisons la tâche *Changer-Etage*, pour lequel nous disposons d'une durée totale, et qui nécessite l'utilisation de six instants supplémentaires non numériques: sortie de la première pièce, entrée dans l'ascenseur, départ de l'ascenseur, arrêt de l'ascenseur, sortie du robot de l'ascenseur, et entrée dans la deuxième pièce. ce qui donne au total **six** instants numériques pour **huit** instants non numériques.

Ce petit exemple illustre le fait que plus les tâches sont complexes, plus les effets sont nombreux, et plus le nombre d'instants nécessaire pour représenter la tâche augmente. Par contre, seule la tâche elle-même est affectée d'une durée explicite. C'est-à-dire que seuls ses instants de début et de fin sont à considérer comme étant "numériques". Donc, plus les tâches sont complexes, plus on a besoin d'utiliser des instants "non numériques".

Bref, pour résumer, on peut aisément conclure que les instants extrémités d'un effet sont généralement reliés aux autres instants du graphe par l'intermédiaire uniquement de contraintes symboliques, au contraire d'une tâche, qui elle définit en plus une contrainte numérique (sa durée) entre ses extrémités. Les événements ponctuels, extérieurs à l'activité de l'agent, mais connus à l'avance, se situent également dans cette deuxième catégorie, puisqu'ils sont généralement datés. Nous avons déterminé qu'en moyenne, dans les plans produits dans le cadre de l'application du chantier, le graphe final se composait à **50%** seulement d'instants numériques.

Retour à notre exemple illustratif

Si l'on considère la tâche *Délivrer_Message (Albert, secrétaire)* de la section précédente, elle a pour effet *Message-Délivré*, qui devient vrai après la fin de la tâche. On ne souhaite pas imposer le fait que cet effet devienne vrai immédiatement à l'issue de la tâche, c'est-à-dire que la secrétaire peut terminer de noter le message après qu'Albert soit parti. Mais il n'est pas nécessaire pour autant d'imposer un délai sous la forme d'une contrainte numérique. On a juste besoin de savoir que la tâche, une fois exécutée, provoque la transition à VRAI de la proposition *Message-Délivré*, ce qui permettra au planificateur de choisir cette tâche pour l'insérer dans le plan. Par ailleurs, aucune durée de l'effet n'est à considérer, puisqu'il perdure a priori indéfiniment. On a donc là de nouveau un exemple d'instant non numérique.

Le lecteur pourra généraliser ce principe à l'ensemble de l'exemple et constater sur la figure 3.3 que l'on obtient finalement 7 instants numériques contre 7 instants non numériques.

Enseignements

Cette distinction formelle entre des propositions nécessitant l'usage de contraintes numériques pour les caractériser complètement (tâches et événements), et des propositions qui au contraire ne se situent temporellement que relativement à d'autres (effets), est étroitement liée au domaine de la planification, comme en témoigne l'approche développée dans DEVISER [Vere83], ou dans TRIPTIC [Rutten93], où l'on retrouve des constatations similaires sur la proportion d'instants numériques.

L'idée qui nous est donc naturellement venue à l'esprit, et à partir de laquelle toute la méthode qui suit a été développée, est de fait la suivante: réduisons le graphe initial, de n instants, à un **sous-graphe** ne contenant que les m instants numériques, et auquel la propagation numérique pourra être restreinte, et nous obtiendrons alors une complexité globale de la gestion des contraintes numériques en $O(m^3)$! Ceci étant dit, il est clair que cette technique ne peut s'avérer payante que si

1. d'une part la proportion d'instants numériques est effectivement faible, c'est-à-dire si $m \ll n$,
2. d'autre part les processus d'**interaction** entre le sous-graphe et le graphe symbolique global s'avèrent relativement peu coûteux, rendant compte d'un fonctionnement global notablement plus efficace,
3. enfin l'exigence de **complétude** est vérifiée, c'est-à-dire que le passage à un sous-graphe se fait sans perte d'information: la cohérence ou l'incohérence de $\mathcal{G}$ devra pouvoir être vérifiée au niveau du sous-graphe, et l'on devra donner à une requête temporelle la même réponse que si celle-ci avait été effectuée dans un graphe global $\mathcal{G}$ propagé.

Les sections suivantes vont s'attacher, à travers la description de la méthode, à démontrer les points 2 et 3. Le 1er point, argumenté succinctement ci-dessus, sera discuté de manière plus approfondi dans le dernier chapitre du mémoire, lorsque nous chercherons à marquer les limitations et les prospectives liées à notre méthode.

2.2. Vers une Typologie des Contraintes Numériques

Selon ce qui précède, un graphe temporel tel que celui de notre exemple est composé d'instants et de contraintes, lesquelles se décomposent en trois types distincts: les relations de précédence, représentées par des arcs dirigés et constituant le réseau symbolique $\mathcal{S}$, des intervalles de valeurs étiquetant les arcs, constituant le réseau numérique $\mathcal{G}$, et des contraintes de disjonction symbolique $\neq$ constituant l'ensemble I. Concernant ces dernières, nous avons déjà argué du fait qu'elles étaient rares dans nos applications. Nous n'avons d'ailleurs pas eu besoin d'utiliser de telles contraintes dans notre exemple. Nous les ignorerons de ce fait dans un premier

temps, et étudierons leur prise en compte explicitement dans la section 8. Donc, notre graphe peut être vu comme une représentation de $\mathcal{S}$, si l'on ne considère que les arcs dirigés, ou comme une représentation de $\mathcal{G}$, si l'on considère au contraire les valuations.

Notations : Désormais, une contrainte sera notée selon son type:

- Une contrainte symbolique de $\mathcal{S}$, entre deux instants i et j, sera notée S_{ij} lorsque cela sera nécessaire, mais nous utiliserons généralement une notation directe ($i \leq j$) ou (i **?** j), traduisant les deux seuls cas de figures possibles, comme cela a été explicité dans le chapitre précédent.
- Une contrainte d'inégalité de I sera également notée sous la forme directe ($i \neq j$)
- Une contrainte numérique sera par contre désignée sous une forme plus classique dans la littérature CSP: $N_{ij} = [inf_{ij}, sup_{ij}]$.
- Nous conserverons également une notation générique C_{ij}, notamment pour désigner une précédence dans sa généralité: C_{ij} est une **précédence** ssi i est situé temporellement avant j, ce qui s'exprime dans le formalisme symbolique par un arc dirigé ($i \leq j$), et dans le formalisme numérique par un intervalle $[inf_{ij}, sup_{ij}] \subseteq [0,+\infty[$.

Dès lors, on peut distinguer plusieurs types de contrainte numérique, selon la contrainte symbolique qu'elle sous-tend:

- Les contraintes **fortes**: une contrainte numérique N_{ij} est dite forte dès lors qu'elle implique une précédence:
  N_{ij} forte $\Rightarrow (i \leq j \vee j \leq i)$,
  ce qui s'écrit: $C_{ij} = [inf_{ij}, sup_{ij}]$ est forte ssi $[inf_{ij}, sup_{ij}] \subseteq [0,+\infty[\vee [inf_{ij}, sup_{ij}] \subseteq]-\infty,0]$,
  ou encore: $C_{ij} = [inf_{ij}, sup_{ij}]$ est forte ssi $inf_{ij}.sup_{ij} > 0$.

- Les contraintes **faibles**: par opposition, N_{ij} sera dite faible si elle n'implique aucun ordre strict, donc si inf_{ij} et sup_{ij} sont de signe différent:
  N_{ij} faible $\Rightarrow i$ **?** j,
  ce qui s'écrit: $C_{ij} = [inf_{ij}, sup_{ij}]$ est faible ssi $inf_{ij}.sup_{ij} < 0$.

Par ailleurs, il convient de distinguer les contraintes numériques **d'entrée** des contraintes **propagées**, les premières étant ajoutées dans le gestionnaire de contraintes temporelles par le planificateur, alors que les dernières ne sont par définition rendues explicites qu'à l'issue de la propagation numérique, laquelle, rappelons-le, rend compte d'un graphe complet.

Pour ce qui est des contraintes d'entrée, il convient de les séparer en deux camps distincts:

- Les contraintes **étiquetées**: il s'agit des dates et des durées proprement dites, liées à l'événement ou à la tâche qu'elles qualifient. Ces contraintes sont généralement bornées des deux côtés, c'est-à-dire que $N_{ij}=[inf_{ij}, sup_{ij}]$, avec $-\infty < inf_{ij} \leq sup_{ij} < +\infty$. En guise de remarque, signalons que si N_{ij} est une contrainte étiquetée, alors par définition N_{ji} est également une contrainte étiquetée.

- Les **précédences héritées**: nous dirons que N_{ij} est une précédence héritée (ou simplement par abus précédence, lorsqu'il n'y aura pas d'ambiguïté) si ($i \leq j$) dans $\mathcal{S}$ et qu'il n'existe aucune contrainte étiquetée entre i et j. On induit alors $N_{ij}=[0,+\infty[$ dans $\mathcal{G}$. Nous verrons dans la prochaine section quels types de précédences héritées sont susceptibles d'être ajoutés dans $\mathcal{G}$.

Il est à noter que les précédences héritées sont par définition des contraintes fortes, et que les contraintes étiquetées le seront aussi généralement. Pour avoir un exemple de contrainte d'entrée faible, on peut considérer deux événements devant arriver "à peu près en même temps", mais dans n'importe quel ordre, ce qui pourra par exemple s'exprimer par la contrainte [-10,10]. Les applications de planification que nous traitons présentent rarement ce type de contraintes en entrée. Nous n'en parlerons donc pas explicitement pour l'instant. Que le lecteur garde cependant à l'esprit que notre système autorise tout-à-fait ce type de contrainte.

Si l'on regarde maintenant du côté des contraintes propagées, celles-ci pourront par contre forcément être aussi bien fortes que faibles. Parmi les fortes, qui induisent par définition une précédence, une distinction peut être à nouveau faite:

- Les précédences **implicites**: lorsque la relation de précédence était vraie avant la propagation, et peut donc être vérifiée dans $\mathcal{S}$. Formellement, cela donne:
  N_{ij} est une précédence **implicite** dans $\mathcal{G}$ ssi ($i \leq j \vee j \leq i$) dans $\mathcal{S}$.

- Les précédences **induites**: lorsqu'il s'agit au contraire d'une précédence résultant de la combinaison de contraintes numériques, et dont seule la propagation numérique a permis de révéler l'existence (cf chapitre 2, § 2.4.1). Cette précédence ne peut donc pas être retrouvée dans $\mathcal{S}$:
  N_{ij} est une précédence **induite** dans $\mathcal{G}$ ssi i **?** j dans $\mathcal{S}$.

Les définitions précédentes conduisent à une typologie des contraintes numériques schématisée dans la figure suivante, où apparaissent en italique les deux types de contraintes auxquelles nous allons devoir plus particulièrement nous intéresser dans la suite du chapitre:

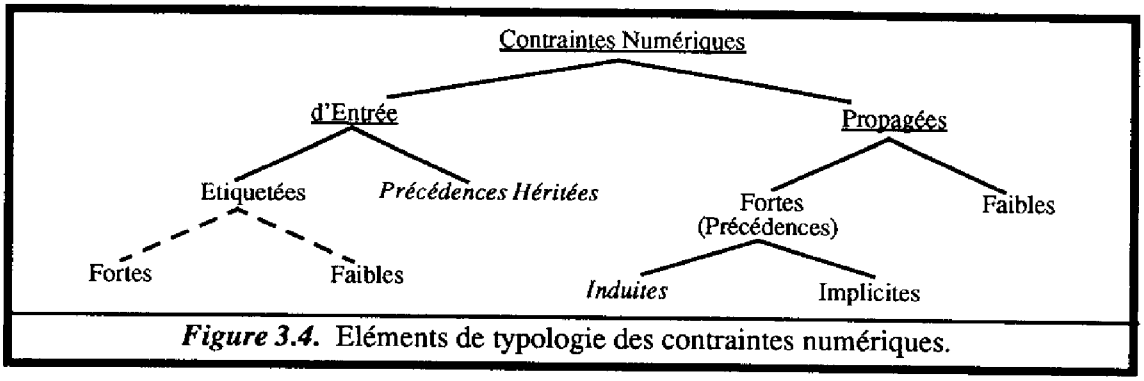


Figure 3.4. Eléments de typologie des contraintes numériques.

Cette typologie conduit en particulier à la définition d'un instant numérique:

- Un **instant numérique** est un instant début ou fin d'une contrainte étiquetée, c'est-à-dire:
  L'instant i est numérique ssi $\exists$ un instant j tel que N_{ij} est une contrainte étiquetée.
- Nous appellerons désormais m le nombre d'instants numériques dans le graphe $\mathcal{G}$ courant.

Dès lors, on peut intuitivement définir le sous-graphe numérique comme étant la restriction du graphe $\mathcal{G}$ aux seuls m instants numériques. Au niveau des contraintes, cette restriction permet de conserver dans le sous-graphe les contraintes étiquetées. Il sera cependant également nécessaire a priori d'y reporter les précédences héritées, qui sont les précédences entre instants numériques présentes dans $\mathcal{S}$, et devant être prises en compte dans $\mathcal{G}$. Intuitivement, elles servent à caractériser le positionnement temporel relatif des tâches et événements qualifiés par les contraintes étiquetées. Nous allons voir cependant que nous pouvons **filtrer** cet ensemble de précédences, de manière à ignorer celles qui sont redondantes, grâce à la définition de la **relation de dominance**.

2.3. Illustration par l'Exemple

Mais avant cela, essayons de nous représenter un peu plus clairement ce que nous souhaitons réaliser. Reportons-nous pour cela au graphe de la figure 3.3. Nous constatons que ce graphe comprend 7 instants numériques qui sont 0, 1, 2, 6, 7, 10 et 11. Notre objectif est de nous restreindre à un sous-graphe composé de ces seuls instants, et dans lequel figureront naturellement les contraintes étiquetées que sont $N_{0,1}$, $N_{1,2}$, $N_{6,7}$ et $N_{10,11}$. Par contre, il sera également nécessaire pour des raisons de complétude de retrouver dans le sous-graphe des précédences héritées telles que celles entre 0 et 6, entre 0 et 10 ou entre 7 et 2 par exemple. Sans cela, l'information contenue dans le sous-graphe serait incomplète, et il en serait donc de même pour l'algorithme de propagation. Mais toutes les précédences entre des instants numériques ne sont pas nécessaires. Par exemple, on peut établir dans $\mathcal{G}$ une précédence entre 6 et 2, qui

est redondante vis-à-vis de la contrainte étiquetée $N_{6,7}$ et de la précédence entre 7 et 2. Celle-ci n'a donc pas besoin d'être insérée dans le sous-graphe.

Tout ceci va être développé tout d'abord au travers de définitions formelles du concept de dominance et de notre sous-graphe numérique, après quoi nous pourrons donner une vision globale de la méthode en la replaçant dans le cadre d'une construction incrémentale du graphe.

3. Définitions et Propriétés Formelles

3.1. Relation de Dominance

La relation de dominance définit une relation d'ordre partiel sur les précédences au sens large, c'est-à-dire qu'elle concerne aussi bien les contraintes symboliques dans S que les précédences héritées correspondantes dans $\mathcal{G}$. Schématiquement, une précédence C_{ij} domine une précédence $C_{i'j'}$ ssi $C_{i'j'}$ peut être établie par un chemin empruntant C_{ij}. Nous donnons ci-dessous la définition formelle ainsi qu'une représentation graphique des divers cas.

Définition 3.1 : Soient C_{ij} et $C_{i'j'}$ deux relations de précédence.
C_{ij} **domine** $C_{i'j'}$ ssi $[(i' \leq i) \vee (i' = i)] \wedge [(j \leq j') \vee (j = j')]$.

Notation : On notera dans la suite $C_{ij} \angle C_{i'j'}$.

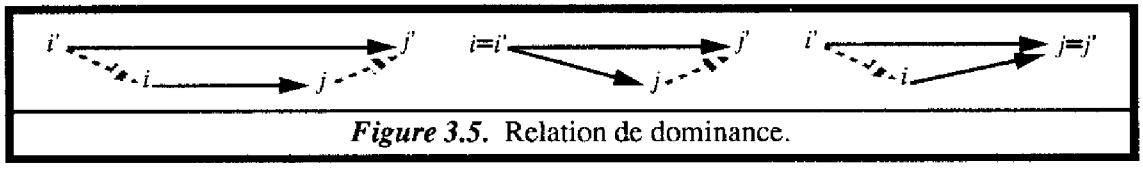


Figure 3.5. Relation de dominance.

On vérifie aisément que cette définition est en accord avec les propriétés d'une relation d'ordre (réflexivité, transitivité, antisymmétrie). Considérons l'ensemble des contraintes de précédence $\wp$, muni de la relation $\angle$. Cette relation est un ordre partiel, puisque deux éléments de $\wp$ peuvent ne pas être reliés par la relation de dominance. Il ne peut donc s'agir d'un bon ordre, c'est-à-dire qu'il n'existe pas nécessairement de plus petit élément. Cependant, comme $\wp$ est discret et borné par définition, on en déduit aisément que $\{\wp, \angle\}$ admet un ou plusieurs éléments minimaux, répondant à la définition suivante:

Définition 3.2 :

Un élément minimal de $\{\wp,\angle\}$ est une précédence C_{ij} pour laquelle il n'existe pas de $C_{i'j'}$ telles que $C_{i'j'} \angle C_{ij}$.

Nous nommerons dans la suite ces éléments minimaux du graphe $\mathcal{S}$ [resp. $\mathcal{G}$] **précédences minimales** ou **précédences non dominées**.

Il est à noter que cette notion de redondance des précédences obtenues par transitivité existe déjà dans $\mathcal{S}$ (quoique de manière non systématique), où la plupart de ces précédences (que nous appellerons désormais **dominées**) ne sont pas représentées explicitement, mais peuvent être établies par les algorithmes de requête symbolique. Mais il est temps désormais de donner la définition du graphe numérique, qui s'impose d'elle-même au vu des définitions précédentes.

3.2. Définition du Sous-Graphe Numérique

Définition 3.3 : Soit un gestionnaire de contraintes temporelles maintenant les trois réseaux de contraintes I, $\mathcal{S}$ et $\mathcal{G}$ préalablement définis. Le **sous-graphe numérique** $\mathcal{N}$ de $\mathcal{G}$ est alors $\mathcal{N}=\{\mathbf{V_n}, \mathbf{R_n}=\mathbf{R_e}\cup\mathbf{R_p}\}$, avec:

- $\mathbf{V_n} = \{i / i$ est un instant numérique$\}$, avec $\mathbf{V_n}\subseteq\mathbf{V}$ et $\text{card}(\mathbf{V_n})=m$,
- $\mathbf{R_e} = \{N_{ij} / (i,j)\in V_n^2$, et N_{ij} est une contrainte étiquetée$\}$,
- $\mathbf{R_p} = \{N_{ij} / (i,j)\in V_n^2$, et N_{ij} est une **précédence minimale héritée** ou **Lien**$\}$.
- $\mathbf{R_e}\cap\mathbf{R_p} = \emptyset$.

En d'autres termes, R_p est l'ensemble des précédences non dominées liant deux instants numériques n'étant pas déjà liés par une contrainte étiquetée. Ces **Liens** représentent donc les seules précédences héritées (cf la typologie figure 3.4) de $\mathcal{G}$ devant être effectivement insérées dans $\mathcal{N}$ avant propagation. Il reste à vérifier que notre exigence de complétude au niveau du passage de $\mathcal{G}$ à $\mathcal{N}$, dont nous parlions pour introduire notre méthode, est bel et bien satisfaite.

3.3. Propriété de Complétude

Propriété 3.1 :

1. Le réseau minimal obtenu après propagation numérique dans $\mathcal{N}$ est identique au réseau minimal obtenu après propagation numérique dans $\mathcal{G}$ et restriction de ce réseau au sous-ensemble des instants numériques V_n.
2. $\mathcal{G}$ est cohérent ssi $\mathcal{N}$ est cohérent.

Preuve:

Considérons en parallèle les deux graphes $\mathcal{N}$et $\mathcal{G}$ tels que définis, tous deux propagés après un ajout d'une contrainte quelconque.

1) Si $\mathcal{N}$est incohérent, alors nécessairement $\mathcal{G}$ est incohérent, puisque $\mathcal{N}$est un sous-réseau de contraintes de $\mathcal{G}$.

2) Il reste à prouver que si $\mathcal{G}$ est incohérent, alors $\mathcal{N}$l'est aussi. Raisonnons par l'absurde en supposant que $\mathcal{G}$ est incohérent et $\mathcal{N}$est cohérent. C'est-à-dire qu'il existe au moins une contrainte $N_{ij} \in R_n$ cohérente dans $\mathcal{N}$, mais incohérente vis-à-vis d'un chemin de contraintes d'entrée empruntant des instants de $\mathcal{G}$, c'est-à-dire:
$\exists$ *Chemin*= $\{i, k_1, ..., k_n, j\}$ tel qu'un de ses éléments au moins n'appartient pas à V_n, et tel que $N_{ij}'=(N_{ik_1} \oplus ... \oplus N_{k_nj})$, et $N_{ij}' \cap N_{ij} = \emptyset$.

Considérons les sous-chemins de *Chemin* du type $\{k, ... k'\}$ tels que k et $k' \in V_n$, et toutes les contraintes le long de ces sous-chemins sont hors de R_n. Il s'agit donc uniquement de précédences en entrée. La composition de ces précédences rend compte
- soit d'une contrainte (k ? k'), qui se traduit par une contrainte numérique $]-\infty, +\infty[$, ce qui induit nécessairement par composition $N_{ij}'=]-\infty, +\infty[$. Ce résultat contredit l'hypothèse $N_{ij} \cap N_{ij}' = \emptyset$.
- soit d'une contrainte ($k \leq k'$) ou ($k' \leq k$), qui se traduit par une contrainte numérique $[0, +\infty[$. Cette contrainte représente une précédence héritée de $\mathcal{G}$ entre k et k'. Soit elle est minimale et a donc été ajoutée dans $\mathcal{N}$ par définition de celui-ci. Soit elle est dominée par une précédence héritée minimale, et la propagation a donc par transitivité rendu compte dans $\mathcal{N}$d'une contrainte $N_{kk'} \subseteq [0, +\infty[$. C'est-à-dire que le sous-chemin $\{k ... k'\}$ fournit par composition une contrainte moins restrictive que la contrainte $N_{kk'}$ présente dans $\mathcal{N}$. On peut donc remplacer ce sous-chemin par la contrainte $N_{kk'}$ qui est nécessairement plus contraignante.

Si l'on fait de même pour chaque sous-chemin empruntant des instants de $\mathcal{G}$, on se ramène à un chemin de contraintes plus restrictives toutes contenues dans R_n. C'est-à-dire que $N_{ij}' \subseteq N_{ij}$. Ce qui contredit encore une fois l'hypothèse $N_{ij} \cap N_{ij}' = \emptyset$. Nous obtenons donc la contradiction recherchée.

CQFD.

En résumé, le sous-graphe numérique se définit simplement en reportant dans $\mathcal{N}$les Liens. Nous pouvons désormais oublier le graphe numérique global $\mathcal{G}$. Le gestionnaire de relations temporelles va en effet reposer sur les deux graphes virtuels que sont $\mathcal{S}$ (pour gérer les contraintes symboliques) et $\mathcal{N}$(défini à l'intérieur de $\mathcal{S}$, pour gérer les contraintes numériques),

ainsi que sur l'ensemble I (pour les inégalités, défini également à l'intérieur de $\mathcal{S}$). La section suivante va s'attacher à replacer ce principe général dans le cadre du fonctionnement incrémental de nos algorithmes. Par ailleurs, la propriété ci-dessus permet d'assurer

1. la vérification de la cohérence de $\mathcal{G}$ par l'intermédiaire d'une propagation dans $\mathcal{N}$
2. une réponse en temps constant à une requête temporelle entre deux points appartenant à $\mathcal{N}$

Nous verrons en section 6 comment il convient de traiter une requête numérique entre deux instants situés en-dehors de $\mathcal{N}$

4. Vue d'Ensemble de la Méthode

4.1. Une Construction Incrémentale

Nous souhaitons rappeler ici brièvement que le processus de construction du plan, et donc du graphe temporel, est incrémental: à chaque ajout d'une contrainte, cet ajout devra être géré à la fois au niveau du réseau de contraintes symboliques global $\mathcal{S}$ et au niveau du sous-réseau de contraintes numériques $\mathcal{N}$. Plusieurs cas de figures peuvent alors se présenter:

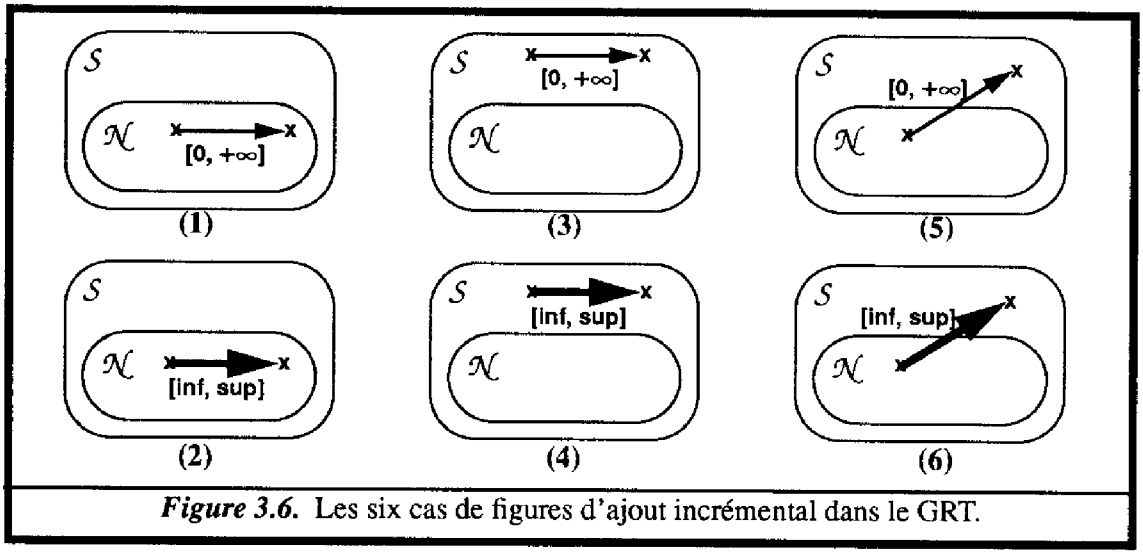


Figure 3.6. Les six cas de figures d'ajout incrémental dans le GRT.

1. Ajout d'une précédence entre i et j, tels que $i \in \mathcal{N}$ et $j \in \mathcal{N}$
2. Ajout d'une contrainte étiquetée entre i et j, tels que $i \in \mathcal{N}$ et $j \in \mathcal{N}$
3. Ajout d'une précédence entre i et j, tels que $i \notin \mathcal{N}$ et $j \notin \mathcal{N}$

4. Ajout d'une contrainte étiquetée entre i et j, tels que $i \notin \mathcal{N}$ et $j \notin \mathcal{N}$.
5. Ajout d'une précédence entre i et j, tels que l'un des deux seulement $\in \mathcal{N}$.
6. Ajout d'une contrainte étiquetée entre i et j, tels que l'un des deux seulement $\in \mathcal{N}$.

Ces six cas de figures convergent sur deux aspects: la nécessité de propager cette contrainte aussi bien au niveau symbolique (dans le cas d'une contrainte forte), que numérique, et la nécessité de détecter les précédences induites par la propagation numérique. Par contre, il y a divergence au moment de l'ajout: les ensembles $\mathbf{V_n}$, $\mathbf{R_e}$ et $\mathbf{R_p}$ sont alors complétés différemment selon les cas.

Les points 1 et 2, c'est-à-dire lorsqu'on est à l'intérieur du sous-graphe, sont naturellement identique à un ajout dans $\mathcal{G}$: on vérifie a priori la cohérence de la nouvelle contrainte N_{ij}', en la comparant avec celle (N_{ij}) existant déjà entre i et j, on la met à jour et on la propage au reste du sous-graphe numérique. On devra simplement modifier $\mathbf{R_e}$ et $\mathbf{R_p}$ de la manière suivante:

- (cas 1) si $N_{ij} \in \mathbf{R_e}$ [resp. $\mathbf{R_p}$], alors $N_{ij} \leftarrow N_{ij} \cap N_{ij}'$ dans $\mathbf{R_e}$ [resp. $\mathbf{R_p}$],
  sinon, ajouter N_{ij}' dans $\mathbf{R_p}$.

- (cas 2) si $N_{ij} \in \mathbf{R_e}$, alors $N_{ij} \leftarrow N_{ij} \cap N_{ij}'$ dans $\mathbf{R_e}$,
  sinon, ajouter N_{ij} dans $\mathbf{R_e}$, et l'enlever si besoin est de $\mathbf{R_p}$.

Dans les quatre autres cas de figures, on augmente l'ensemble des instants numériques V_n d'un [resp. deux] éléments. L'ajout de variables dans un CSP nécessite de spécifier les contraintes les liant aux variables déjà présentes. Il va donc s'agir pour nous de déterminer les contraintes devant être ajoutées à R_e et R_p. Etudions les divers cas, en considérant tout d'abord l'ajout d'une contrainte étiquetée:

- (cas 4 et 6) Les deux instants i et j [resp. un seul d'entre eux] doivent être ajoutés à V_n. La seule et unique contrainte à ajouter à R_e est la contrainte étiquetée N_{ij} entre i et j. Il reste à trouver les précédences héritées de $\mathcal{S}$ liant chacun des instants i et j [resp. un seul d'entre eux] à un instant de V_n, et du fait de la définition de $\mathcal{N}$ à ne conserver parmi eux que les seules relations non dominées. Dans le cas de figure (4), cela donne:
  - $V_n' \leftarrow V_n \cup \{i,j\}$
  - $R_e' \leftarrow R_e \cup \{N_{ij}\}$
  - $R_p' \leftarrow R_p \cup \{\text{Liens}\}$

- (cas 3 et 5) Ici, la contrainte ajoutée n'étant pas numérique, ni V_n ni R_e ne sont modifiés. Par contre la nouvelle précédence entre i et j est susceptible d'induire par transitivité de nouvelles précédences entre instants numériques. Il s'agira donc simplement de déterminer les Liens correspondants devant être rajoutées dans R_p.

Dans la section suivante, nous allons étudier en détail le point 4, qui est de toute évidence le plus complexe, après quoi nous regarderons rapidement ce qui doit être retenu pour traiter les cas de figures plus simples 3, 5 et 6. Pour résumer, disons simplement que lors de l'ajout d'une nouvelle contrainte, nous avons deux problèmes spécifiques à résoudre: la **détermination des Liens** qui doivent être également ajoutés dans $\mathcal{N}$ avant de propager (pour les points 3 à 6), et la **détection des précédences induites** issues de la propagation (dans tous les cas). Mais tout d'abord, illustrons rapidement ce fonctionnement incrémental par l'intermédiaire de l'exemple.

4.2. Illustration par l'Exemple

La construction incrémentale de notre graphe illustratif comprend deux étapes d'ajout du type 4 énuméré au paragraphe précédent. La figure 3.7 ci-dessous présente les graphes $\mathcal{S}$ et $\mathcal{N}$ en parallèle (a) dans la situation initiale, (b) à l'issue du premier ajout, et (c) dans la situation finale. A chaque étape nous identifions dans $\mathcal{N}$ la contrainte ajoutée, les Liens identifiés, et les précédences induites par la propagation.

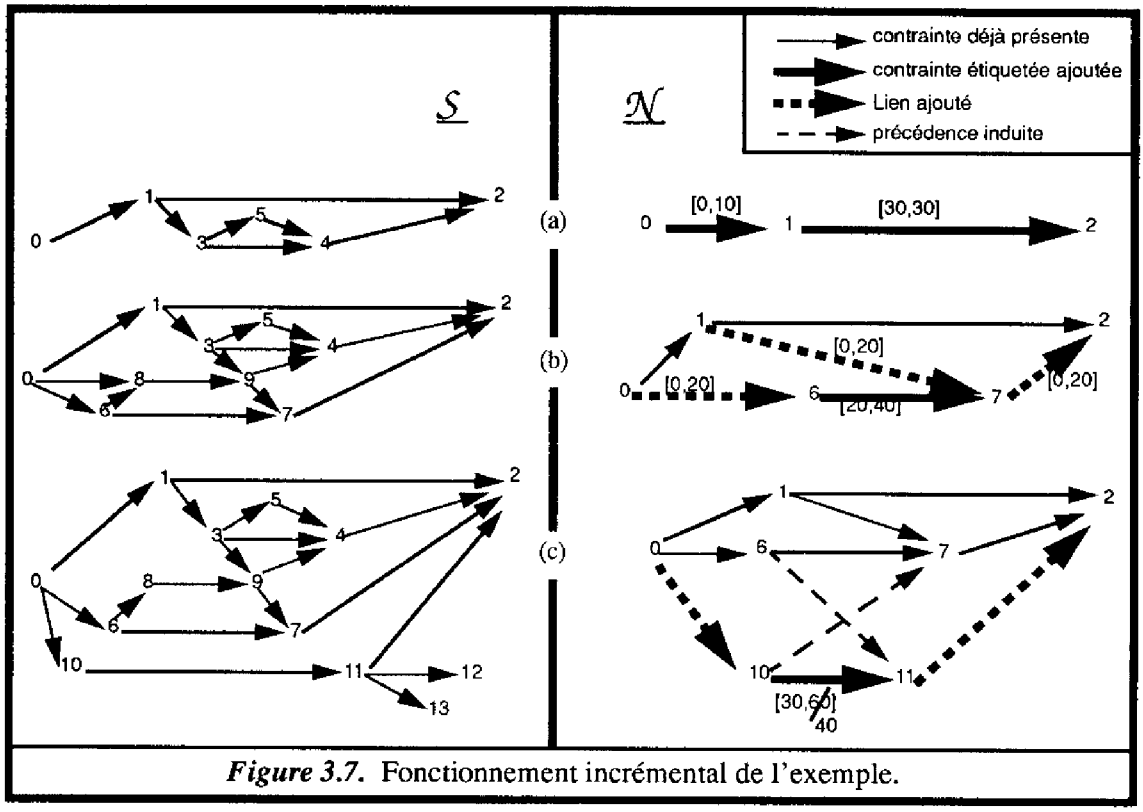


Figure 3.7. Fonctionnement incrémental de l'exemple.

Analysons en détail le passage de (a) à (b). L'ajout de la tâche entre les instants 6 et 7 s'accompagne de l'ajout dans $\mathcal{S}$ d'un ensemble d'instants et de contraintes représentant les

effets de cette tâche et les précédences les reliant au graphe courant. Les seuls nouveaux instants numériques sont 6 et 7. Ils doivent donc être ajoutées dans $\mathcal{N}$avec la contrainte étiquetée $N_{6,7}$. Il reste à chercher des chemins dans $\mathcal{S}$ induisant des précédences entre ces nouveaux instants 6 et 7 et les instants 0, 1 et 2 déjà présents dans $\mathcal{N}$. En fait, quatre précédences peuvent être établies: $(0 \leq 6)$, $(1 \leq 7)$, $(6 \leq 2)$ et $(7 \leq 2)$. L'une d'entre elles, $(6 \leq 2)$, est dominée par $(7 \leq 2)$. Elle ne doit donc pas être retenue. Les trois autres constituent les Liens qui doivent être ajoutés dans $\mathcal{N}$en tant que précédences héritées, et l'ensemble des contraintes ajoutées est propagé dans $\mathcal{N}$par l'intermédiaire d'un algorithme de type PC-2.

Le passage de (b) à (c) suit le même principe. En plus de cela, la propagation va ici induire deux nouvelles précédences entre 10 et 7 d'une part, et 6 et 11 d'autre part. Ces deux contraintes sont en effet égales toutes deux à [10, 40], c'est-à-dire qu'il s'agit de nouvelles contraintes fortes. Ces précédences induites devront être détectées durant la propagation, puis reportées dans $\mathcal{S}$, afin de disposer dans ce réseau de requêtes complètes (cf chapitre 2, § 2.4.1). On peut voir aussi que le domaine de la contrainte numérique entre les instants 10 et 11 a été réduit à [30,40], d'où l'on conclura qu'Albert devra adopter un débit de parole élevé, afin de réduire la durée de la tâche correspondante.

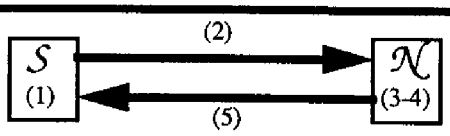
Avant de poursuivre, nous prions le lecteur de noter dès à présent que la figure 3.7 ci-dessus va être abondamment référencée dans toute la suite de ce chapitre.

5. Description d'une Etape d'Ajout Incrémental

Dans toute cette section, nous considérerons que N_{ij} est la contrainte étiquetée ajoutée. Nous étudierons tout d'abord exclusivement le cas numéro 4 où i et j sont deux instants non numériques, les autres cas étant décrits en fin de section. Les instants déjà présents dans $\mathcal{N}$ seront notés k, k', k_0 ... k_m, selon les besoins.

5.1. Vue d'Ensemble

Nous avons choisi de donner tout d'abord une figure illustrant l'ensemble des processus d'interactions entre $\mathcal{S}$ et $\mathcal{N}$lors d'une étape d'ajout. Ces processus seront successivement détaillés dans cette section. Nous voyons clairement apparaître les deux aspects essentiels évoqués précédemment, notamment au travers de l'exemple détaillé au paragraphe précédent, et qui sont la détermination des Liens à insérer dans $\mathcal{N}$en même temps que la contrainte étiquetée, et la détection des précédences induites par la propagation numérique.



(1) propagation symbolique
(2) détermination des Liens
(3) propagation numérique
(4) détection des précédences induites
(5) report des précédences induites

Figure 3.8. Processus général d'ajout incrémental - Interactions entre $\mathcal{S}$ et $\mathcal{N}$

5.2. Quelques Propriétés et Principes Fondamentaux

Reprenons les deux principaux points évoqués ci-dessus. Tout d'abord, pour ce qui est de l'**identification des Liens** entre i [resp. j] et les instants du réseau $\mathcal{N}$courant, il va s'agir de rechercher les instants numériques qui sont prédécesseurs ou successeurs de i [resp. j]. La recherche de ces liens symboliques ne peut bien sûr avoir lieu que dans $\mathcal{S}$, lequel doit de ce fait être à jour. En particulier:

> si N_{ij} est une contrainte forte, et si i **?** j, alors commencer par ajouter et propager $i \leq j$ [resp. $j \leq i$] dans $\mathcal{S}$.

Les instants et contraintes ajoutés par le planificateur doivent donc toujours être traités **d'abord au niveau symbolique** (par l'intermédiaire des algorithmes évoqués au chapitre 2), et ensuite au niveau numérique.

Par ailleurs, si l'on considère par exemple les prédécesseurs de i, alors seuls ceux qui définissent une précédence minimale (c'est-à-dire non dominée) avec i doivent être retenus. Nous allons voir que cette sélection peut être aisément et efficacement effectuée grâce à la structure particulière de $\mathcal{S}$.

♣♣♣

Etudions ensuite le deuxième problème évoqué, à savoir **l'identification des précédences induites**. Celle-ci peut être effectuée **en cours de propagation**. En effet, selon la définition donnée au § 2.2, il faut pouvoir détecter le fait qu'une contrainte faible $N_{kk'}$ devienne forte dans $\mathcal{N}$. Mais cela ne suffit pas: il est également a priori indispensable de vérifier que k **?** k' dans $\mathcal{S}$. En effet, contrairement à ce que pourrait suggérer l'intuition, si une contrainte $N_{kk'}$ est faible, cela ne signifie pas forcément qu'il n'existe pas de précédence entre k et k', comme le rappelle l'énoncé suivant:

Propriété 3.2 :

(1) Si $\mathcal{N}$ est le graphe complet minimal obtenu après propagation, on a:
$N_{kk'}$ est faible ssi k ? k'.
(2) Si $\mathcal{N}$ n'est pas propagé, on a seulement:
si k ? k', alors $N_{kk'}$ est faible.

Le premier point provient de la définition même du réseau minimal, et le second n'est autre que la contrapposée de l'énoncé donné en début de paragraphe: au moment de propager $\mathcal{N}$, si $N_{kk'}$ est forte, alors nécessairement $k \leq k'$ [resp. $k' \leq k$], puisque le réseau $\mathcal{S}$ a préalablement été mis à jour.

Que signifie alors le fait que N_{ij} soit faible dans un graphe non propagé ? D'après les définitions données au paragraphe 2.2, si $N_{kk'}$ est faible, alors

- soit il s'agit d'une précédence **induite**, et donc k ? k'.
- soit il s'agit d'une précédence **implicite**, et donc $k \leq k'$ ou $k' \leq k$.

En effet, lorsqu'on ajoute un instant i au graphe minimal $\mathcal{N}$ courant, on obtient implicitement du même coup m contraintes entre i et les m instants de $\mathcal{N}$. Parmi ces couples d'instants, certains sont reliés par des précédences. Or, notre méthode repose sur le fait que seules les précédences minimales (les Liens) sont explicitement ajoutées, les autres étant affectées de l'intervalle par défaut $]-\infty,+\infty[$. Dans l'exemple décrit dans la figure 3.7, à l'étape (b), $N_{6,2}$ est une telle précédence implicite, initialisée à $]-\infty,+\infty[$, donc faible au début de la propagation. C'est la propagation qui va se charger par la suite de réduire son domaine jusqu'à rendre compte d'une contrainte forte. En conclusion, retenons qu'il existe a priori des **précédences non explicites** dans $\mathcal{N}$ avant propagation.

Nous allons cependant montrer qu'il est possible, sans coût supplémentaire, de rendre explicites dans $\mathcal{N}$ toutes ces précédences dominées, et ce durant la phase de recherche des Liens, donc avant propagation. Cela nous permettra de remplacer des requêtes symboliques ("est-ce que k ? k' " ?) en temps linéaire par une simple observation de la contrainte numérique ("est-ce que $N_{kk'}$ est faible ?"), ce qui requiert une complexité constante. Tout ceci sera détaillé dans les paragraphes suivants.

5.3. Détermination et Ajout des Liens

Notons tout d'abord que cette opération est effectuée dans un réseau $\mathcal{N}$ minimal obtenu lors d'une précédente propagation. Donnons d'abord quelques définitions formelles. La première

définit la relation d'antériorité symbolique dans le graphe numérique $\mathcal{N}$, qui traduit la prise en compte implicite des relations symboliques dans les STPs, invoquée au chapitre 1 (§ 4.3.2). La seconde définition classifie les Liens par rapport à cette relation d'antériorité.

Définition 3.4 : **Ancêtres et descendants** dans le réseau minimal de $\mathcal{N}$:
- $\forall (k, k') \in V_n^2$, k' est **ancêtre** de k ssi $inf_{k'k} \geq 0$.
- $\forall (k, k') \in V_n^2$, k' est **descendant** de k ssi $inf_{kk'} \geq 0$.

Définition 3.5 : **Liens convergents, Liens divergents**: Soit N_{ij} la contrainte ajoutée dont on cherche à déterminer les Liens la reliant au reste de $\mathcal{N}$. Alors,
- N_{ki} [resp. N_{kj}] est un Lien **convergent vers** i [resp. j] ssi N_{ki} [resp. N_{kj}] est un Lien et k est ancêtre de i [resp. j],
- N_{ik} [resp. N_{jk}] est un Lien **divergent de** i [resp. j] ssi N_{ik} [resp. N_{jk}] est un Lien et k est descendant de i [resp. j].

Si nous considérons l'exemple de la figure 3.7(b), où la contrainte $N_{6,7}$ est ajoutée, alors $N_{0,6}$ est un Lien convergent vers 6, $N_{1,7}$ est un Lien convergent vers 7, et $N_{7,2}$ est un Lien divergent de 7. Nous allons maintenant étudier dans un premier temps la détermination des **Liens convergents**. Le cas des Liens divergents en découlera par simple symétrie.

a) Candidats potentiels

Nous savons (cf § 2.3.2 au chapitre 2) que dans notre structure symbolique, si $i \leq j$, alors rang(i) < rang(j), ce qui nous permet d'écrire que:
- une condition nécessaire pour que N_{ki} [resp. N_{kj}] soit un Lien convergent vers i [resp. j] est: rang(k) < rang(i) [resp. rang(k) < rang(j)].

Nous avons donc besoin d'initialiser l'ensemble des candidats potentiels $Conv_i$ [resp. $Conv_j$], susceptibles de définir un Lien convergent vers i [resp. j] comme étant l'ensemble des instants de V_n de rang strictement inférieur à i [resp. j].

b) Filtrage par dominance

Par ailleurs, la définition de la relation de dominance nous permet d'énoncer la propriété suivante:

Propriété 3.3 :

- si N_{ki} est un Lien convergent vers i, alors (1) N_{kj} n'est pas un Lien, et (2) $\forall k'$ tel que k' est ancêtre de k, $N_{k'i}$ et $N_{k'j}$ ne sont pas des Liens (cf figure 3.9 ci-dessous).
- si N_{kj} est un Lien convergent vers j, alors (3) $\forall k'$ tel que k' est ancêtre de k, $N_{k'j}$ n'est pas un Lien.

Figure 3.9. Lien convergent et précédences dominées.

La preuve est immédiate en constatant par exemple pour les points (1) et (2) que $N_{ki} \angle N_{kj}$, $N_{ki} \angle N_{k'i}$ et $N_{ki} \angle N_{k'j}$ et que $\forall k'$ ancêtre de k, $N_{ki} \angle N_{k'i}$ et $N_{ki} \angle N_{k'j}$, ce que suggère la figure ci-dessus. Donc, dès que l'on a déterminé un Lien convergent N_{ki}, tous les ancêtres de k ne sont plus candidats et peuvent être retirés de $Conv_i$ et $Conv_j$, et dès que l'on a déterminé un Lien convergent N_{kj}, tous les ancêtres de k peuvent être retirés de $Conv_j$. Pour revenir à l'exemple de la figure 3.7(b), à partir du moment où $N_{0,6}$ est un Lien, on peut en déduire que $N_{0,7}$ n'est pas un Lien. Par contre, 1 n'étant pas symboliquement relié à 6, il reste candidat comme prédécesseur possible de 7.

Tout ceci suggère un ordre de recherche des candidats potentiels, en cherchant les Liens convergents vers i avant de se préoccuper des Liens convergents vers j, et en commençant dans tous les cas par regarder les instants "les plus proches" de i [resp. j], c'est-à-dire ceux de rang supérieur. Ceci nous conduit de fait à classer les éléments de $Conv_i$ et $Conv_j$ par ordre de rang décroissant. Ces deux ensembles se définissent donc ainsi:

- $Conv_i = \{k_C \dots k_0 \,/\, \forall \lambda = 0 \dots C, k_\lambda \in \mathcal{N}$, et $\text{rang}(k_0) \leq \dots \leq \text{rang}(k_C) < \text{rang}(i)\}$,
- $Conv_j = \{k_{C'} \dots k_0 \,/\, \forall \lambda = 0 \dots C', k_\lambda \in \mathcal{N}$, et $\text{rang}(k_0) \leq \dots \leq \text{rang}(k_{C'}) < \text{rang}(j)\}$.

Ces deux ensembles seront dès lors gérés en tant que piles, les éléments étant dépilés les uns après les autres, du "plus proche" de i [resp. j] au "plus éloigné", comme le suggère l'algorithme de recherche des Liens convergents ci-dessous, où E_{init} représente l'ensemble des contraintes à rajouter dans $\mathcal{N}$. Les instructions repérées par une astérisque seront explicités dans le prochain paragraphe.

AJOUTE-ARCS-CONVERGENTS (i, j)
  TantQue $Conv_i \neq \emptyset$ Faire
  $x \leftarrow$ Dépiler($Conv_i$)
  Si $x \leq i$ Alors
    $E_{init} \leftarrow E_{init} \cup \{(x,i)\}$.
    Retirer x de $Conv_i$.
    Pour tout $x' \in Conv_i$ / $inf_{x'x} \geq 0$ [i.e. x' ancêtre de x]
      Retirer x' de $Conv_i$ et de $Conv_j$.
      $N_{x'x} \leftarrow N_{x'x} \cap [0, +\infty[$ (*)
  TantQue $Conv_j$ non vide Faire
  $x \leftarrow$ Dépiler($Conv_j$)
  Si $x \leq j$ Alors
    $E_{init} \leftarrow E_{init} \cup \{(x,j)\}$.
    Pour tout $x' \in Conv_j$ / $inf_{x'x} \geq 0$ [i.e. x' ancêtre de x]
      Retirer x' de $Conv_j$.
      $N_{x'x} \leftarrow N_{x'x} \cap [0, +\infty[$ (*)

♣♣♣

Le lecteur aura aisément deviné que la recherche des **Liens divergents** se base sur le même principe, en définissant les divers éléments de manière symétrique, notamment les ensembles de candidats potentiels:

- $Div_i = \{k_0 \ldots k_D$ / $\forall \lambda = 0 \ldots D$, $k_\lambda \in \mathcal{N}$, et rang($i$) < rang($k_0$) ≤ ... ≤ rang($k_D$) $\}$,
- $Div_j = \{k_0 \ldots k_{D'}$ / $\forall \lambda = 0 \ldots D'$, $k_\lambda \in \mathcal{N}$, et rang($j$) < rang($k_0$) ≤ ... ≤ rang($k_{D'}$) $\}$.

L'initialisation des quatre ensembles $Conv_i$, $Conv_j$, Div_i et Div_j se fait par un simple parcours linéaire de V_n, en $O(m)$. L'algorithme de recherche des Liens divergents AJOUTE-ARCS-DIVERGENTS est alors parfaitement symétrique du précédent.

♣♣♣

Nous avons donc utilisé la structure de rang de $\mathcal{S}$ pour organiser la recherche, et permettre d'éliminer bon nombre de candidats avant qu'ils ne soient testés comme prédécesseurs ou successeurs potentiels de i et j, ce qui nécessite des requêtes symboliques en $O(n)$. Au contraire, la notion d'ancêtre et descendant dans $\mathcal{N}$ permet d'utiliser des **requêtes en temps constant** pour éliminer les instants qui ne sont plus susceptibles de conduire à la détermination d'un Lien.

5.4. Détection et Report des Précédences Induites

Comme nous l'avons dit plus haut, avant la propagation, les précédences dominées entre i [resp. j] et d'autres instants numériques k subsistent dans $\mathcal{N}$ sous la forme de contraintes fai-

bles]-∞,+∞[. Analysons maintenant les algorithmes du paragraphe précédent. Il est tout d'abord aisé d'établir la propriété suivante:

Propriété 3.4 : $N_{k'i}$, avec $k' \leq i$, est une précédence dominée ssi $\exists\, k \,/\, k'$ est ancêtre de k et N_{ki} est un Lien.

La condition nécessaire est immédiate: si $k' \leq k$ et N_{ki} est un Lien, alors $N_{k'i}$ est dominée par N_{ki}. La condition suffisante n'est guère plus difficile à établir: si $N_{k'i}$ est une précédence dominée, alors, comme $\{\wp,\angle\}$ est borné, et d'après la caractérisation des Liens comme éléments minimaux de $\{\wp,\angle\}$ (cf définition 3.2), $\exists$ au moins un Lien N_{ki} tel que N_{ki} domine $N_{k'i}$; alors nécessairement k' est un ancêtre de k.

La propriété équivalente peut être établie par rapport aux descendants. Les deux propriétés ainsi mises en avant étant applicables à i aussi bien qu'à j, on a donc:

- **tous les ancêtres et descendants éliminés durant le processus de recherche de Liens, et eux seuls, définissent des précédences dominées avec i [resp. j].**

Dès lors, il suffit pour chaque k ainsi éliminé de $Conv_i$ [resp. $Conv_j$, Div_i, Div_j] de rendre explicite la précédence dominée N_{ki} [resp. N_{kj}, N_{ik}, N_{jk}] en forçant sa borne inférieure à la valeur 0. Le lecteur observateur aura pu remarquer que cela était fait dans l'algorithme présenté au paragraphe précédent (instructions marquées d'une astérisque).

Pour faire le lien avec l'exemple, il suffit de jeter un oeil à nouveau sur figure 3.7(b), en considérant, comme nous l'avons vu, qu'il existe une précédence implicite $6 \leq 2$. L'algorithme du paragraphe précédent élimine à un moment donné l'instant 6 comme n'étant pas susceptible d'appartenir à un Lien. Il suffit alors à ce moment-là de rendre explicite la précédence $N_{6,2}$ dans $\mathcal{N}$. Selon ce que nous avons dit au paragraphe 5.2, cela revient à dire que nous pouvons rendre explicites les précédences implicites dans $\mathcal{N}$ **avant** de propager. Le résultat fondamental suivant s'impose alors désormais de lui-même. Rappelons que V_n' est le nouvel ensemble d'instants numériques après l'ajout de i et j: $V_n' \leftarrow V_n \cup \{i, j\}$ (cf § 4.1).

Propriété 3.5 : Avant propagation dans $\mathcal{N}$, $\forall\, (k, k') \in V_n'^2$,

- $N_{kk'}$ est faible ssi k **?** k',
- et donc: $N_{kk'}$ est une contrainte faible devenant forte en cours de propagation ssi $N_{kk'}$ est une précédence induite.

Du même coup, il suffit, en cours de propagation, de tester si une contrainte faible devient forte pour la considérer comme une précédence induite, ce qui conduit à l'algorithme de propagation modifié suivant (cf chapitre 2, § 2.4.2):

PROPAGER (E_{init})
  $MAJ \leftarrow E_{init}$.
  $REPORTS \leftarrow \emptyset$
  Tant Que $MAJ \neq \emptyset$ Faire
    $(i, j) \leftarrow$ Dépiler (MAJ)
    Pour $k = 1$ à m, $k \neq i,j$ Faire
      Si N_{ik} est faible Alors $Faible_{ik} \leftarrow$ Vrai
      Si N_{kj} est faible Alors $Faible_{kj} \leftarrow$ Vrai
      Si $MODIFIE$ (i, k, j)
        Alors
          $MAJ \leftarrow MAJ \cup (i, k)$.
          Si $Faible_{ik}$ et N_{ik} est forte Alors AJOUT-DOMIN($i,k,REPORTS$).
      Si $MODIFIE$ (k, j, i)
        Alors
          $MAJ \leftarrow MAJ \cup (k, j)$.
          Si $Faible_{kj}$ et N_{kj} est forte Alors AJOUT-DOMIN($k,j,REPORTS$).

La fonction AJOUT-DOMIN(k, k', $REPORTS$), donnée ci-après, permet de **filtrer par la relation de dominance** les précédences induites détectées. Ne sont rajoutées dans l'ensemble $REPORTS$, initialisé à l'ensemble vide en début de propagation, que les précédences induites minimales qui devront être reportées dans $\mathcal{S}$. De plus, toute précédence ajoutée est susceptible de dominer des précédences déjà présentes, qu'il convient alors de supprimer.

AJOUT-DOMIN ($u, v, REPORTS$) :
  Si $REPORTS = \emptyset$ Alors $REPORTS \leftarrow \{(u,v)\}$.
  Sinon
    TantQue $REPORTS \neq \emptyset$ Faire
      $(x,y) \leftarrow$ Dépiler ($REPORTS$)
      Si $(x,y) \angle (u,v)$ Alors EXIT [sortie forcé avec $REPORTS$ inchangé].
      Si $(u,v) \angle (x,y)$ Alors $REPORTS \leftarrow REPORTS \setminus (x,y)$.
    $REPORTS \leftarrow REPORTS \cup \{(u,v)\}$.

Dans l'exemple de la figure 3.7(c), deux précédences induites minimales sont identifiées et reportées dans $\mathcal{S}$: il s'agit de $N_{10,7}$ et $N_{6,11}$, qui sont devenues fortes en cours de propagation, induisant de ce fait $10 \leq 7$ et $6 \leq 11$. Notons au passage pour clarifier ce qui a été explicité plus haut que $N_{0,11}$ et $N_{10,2}$ constituent également des précédences induites, mais elles sont dominées respectivement par $N_{6,11}$ et $N_{10,7}$, et ne nécessitent donc pas d'être reportées dans $\mathcal{S}$, puisqu'elles pourront être retrouvées par transitivité. Ceci illustre parfaitement le rôle de la fonction AJOUT-DOMIN.

Disposer des précédences induites est primordial pour le planificateur, comme nous l'avons déjà signalé au chapitre 2. Dans notre exemple, ces nouvelles relations symboliques impliquent un recouvrement nécessaire des deux tâches *Délivrer_Message (Albert, secrétaire)* et

Aller_A (Albert, bureau, ascenseur), ce qui signifie qu'Albert doit commencer à courir vers l'ascenseur avant d'avoir terminé de délivrer son message. Cette information est rendue accessible par le planificateur, qui ne pourra donc par la suite forcer les deux tâches à se succéder l'une à l'autre. Si un tel ordonnancement des tâches s'avérait nécessaire, il serait alors refusé au niveau symbolique. Ceci permet de réagir au plus tôt à un ajout incohérent, et de déclencher éventuellement les techniques de replanification appropriées.

5.5. Bilan en Termes de Complexité

Nous commencerons par nous munir de deux définitions qui vont nous être utiles pour étudier la complexité en temps des algorithmes préalablement définis.

Définition 3.6 : Degré de parallélisme.

1. Une **composante parallèle** d'un graphe $\mathcal{N}$ est un sous-graphe tel que pour tout couple d'instants (k, k') de ce sous-graphe, $N_{kk'}$ est une contrainte faible (c'est-à-dire qu'il n'existe aucune précédence à l'intérieur d'une telle composante).
2. Le **degré de parallélisme** p d'un graphe est la dimension de la composante parallèle de taille maximale.

Dans le graphe $\mathcal{N}$ final de notre exemple (cf figure 3.7(c)), {1,6,10} est une composante parallèle de taille maximale et donc p=3. Dans l'algorithme de **recherche de Liens**, les seuls instants pour lesquels on interrogera le graphe symbolique sont par définition non contraints entre eux. On aura donc au maximum $2.p$ instants à comparer à i et j, chaque comparaison nécessitant une requête symbolique en $O(n)$. A quoi s'ajoutent à chaque fois les m parcours des ensembles de candidats potentiels pour éliminer les ancêtres et les descendants. La complexité de cette phase est donc $O(p.(n+m)) \equiv \mathbf{O(p.n)}$, en négligeant la phase de construction des ensembles de candidats, en $O(m)$. De plus, la dimension n exprime la complexité en pire cas de chaque contrainte symbolique censée identifier un prédécesseur ou successeur de i ou j. Comme ces instants seront le plus souvent "proches" de i et j, eu égard à la structure de rang, ces requêtes symboliques auront une complexité en moyenne nettement plus faible.

L'algorithme PC-2 dans un graphe de dimension m a une complexité globale en $O(m^3)$, et en $O(m^2)$ pour ce qui est d'une seule étape d'ajout incrémental d'une contrainte dans le graphe [Mackworth85]. La méthode du sous-graphe numérique amène à ajouter une contrainte étiquetée ainsi qu'un ensemble de Liens, $2.p$ au maximum. La complexité de la **propagation** proprement dite est donc de l'ordre de $\mathbf{O(p.m^2)}$ en pire cas pour chaque ajout d'une contrainte étiquetée entre deux instants non numérique.

A quoi il faut ajouter la complexité de la **détection des précédences induites**. Celle-ci n'est autre que le nombre d'appels à la fonction AJOUT-DOMIN, c'est-à-dire le nombre de précédences induites détectées à chaque propagation, multiplié par la longueur maximale de la liste *REPORTS*, c'est-à-dire le nombre de précédences induites minimales, durant une phase de propagation.

Pour le nombre de précédences induites, il est de toute évidence au maximum égal à $2.m$, c'est-à-dire que tout nouvel instant i ou j est susceptible d'induire une précédence avec chaque instant déjà présent dans $\mathcal{N}$. En ce qui concerne le nombre de précédences induites minimales présentes dans *REPORTS*, remarquons que ces précédences affectent nécessairement l'un des deux instants i et j ajoutés. Considérons deux précédences induites minimales affectant l'instant i, par exemple N_{ik} et $N_{ik'}$. S'il existe une précédence entre k et k', alors forcément l'une des deux contraintes N_{ik} et $N_{ik'}$ domine l'autre, ce qui contredit la définition de précédence induite minimale. Donc k et k' appartiennent nécessairement à une même composante parallèle. Ceci peut être généralisé, c'est-à-dire que tous les instants reliés à i par des précédences induites minimales appartiennent à la même composante parallèle. Il y en a donc p au maximum. Comme il nous faut considérer des précédences convergentes aussi bien que divergentes, et i aussi bien que j, nous obtenons donc $4.p$ précédences induites minimales au maximum.

Ceci nous permet de rendre compte d'une complexité en pire cas de $\mathbf{O(p.m)}$ pour la phase de détection des précédences induites minimales. Ce résultat doit s'ajouter à la complexité de la phase de propagation, elle-même en $O(p.m^2)$, ce qui n'ajoute rien en ordre de grandeur. En d'autres termes, la phase de détection des précédences induites n'induit aucune complexité supplémentaire. La phase de report et propagation de ces précédences dans $\mathcal{S}$ nécessite p propagations symboliques, ce qui nous donne $\mathbf{O(p.n)}$ en pire cas. Si nous récapitulons, la phase totale d'ajout incrémental d'une contrainte étiquetée entre deux instants non numérique a par conséquent une complexité en $\mathbf{O(p.(n + m^2))}$.

♣♣♣

La complexité totale doit être comparée à la complexité d'un ajout incrémental dans $\mathcal{G}$, qui rappelons-le est $O(n^2)$. Tout dépend du degré de parallélisme de $\mathcal{N}$ et de la proportion d'instants numériques dans $\mathcal{G}$ identifiée comme étant $r=m/n$. Le pire cas correspond à un graphe "fortement parallèle", où p est proportionnel à m, et à un ratio r proche de 1, et notre méthode apparaît alors comme peu payante. Si l'on observe les applications classiques de planification, nous avons dit au début de ce chapitre que nous pouvions nous attendre à un ratio de l'ordre de 50%. Les résultats expérimentaux qui suivront (cf section 7) permettront de quantifier le rôle de r plus précisément. Concernant p, les applications de planification rendent compte de réseaux "séquentiels" (cf [Dorn92]): un plan en robotique représente la séquence de tâches devant être réalisées par un agent (un robot) pour résoudre un ou plusieurs buts. L'agent exécu-

tant les tâches étant le plus souvent unique, le plan produit, quoique non linéaire, sera donc essentiellement séquentiel, ce qui se traduit par un p borné. On obtient alors une complexité en $\mathbf{O}(n + m^2)$ pour laquelle le gain est évident, même pour un ratio r dépassant largement 50%.

En résumé, plus le graphe temporel sera séquentiel, plus la méthode sera avantageuse. Dans le cas le plus courant où p est borné, la méthode apporte un gain qui sera d'autant plus important que le ratio r sera faible, et ce gain sera effectif aussi bien en temps qu'en espace mémoire: $\mathcal{N}$permet en effet de ne maintenir que m^2 contraintes au lieu de n^2.

5.6. Autres Cas de Figures

Par rapport au distinguo effectué au paragraphe 4.1, nous n'avons jusqu'à présent étudié explicitement que le cas de figure 4. Il reste à voir les 5 autres qui sont a priori plus simples. Nous donnons tout d'abord ci-après l'algorithme global d'une phase d'ajout incrémental d'une contrainte étiquetée, qui résume les cas 2, 4 et 6, c'est-à-dire aussi bien lorsque les instants sont numériques que lorsqu'il s'agit d'instants non-numériques. La complexité obtenue dans ces trois cas, quoique réduite en valeur absolue, est du même ordre.

INSERE (i, j, C)
  Si $i \in V_n$ et $j \in V_n$
    Alors
      Si $C_{ij} \cap C = \varnothing$ Alors Rejeter l'ajout: incohérence détectée à priori.
      Sinon
        $C_{ij} \leftarrow C_{ij} \cap C$.
        $E_{init} \leftarrow \{(i,j)\}$.
  Sinon
    Si $i \notin V_n$ Alors
        construire les ensembles $Conv_i$ et Div_i
        $V_n \leftarrow V_n \cup \{i\}$
      Sinon $Conv_i \leftarrow \varnothing$ et $Div_i \leftarrow \varnothing$
    Si $j \notin V_n$ Alors
        construire les ensembles $Conv_j$ et Div_j
        $V_n \leftarrow V_n \cup \{j\}$
      Sinon $Conv_j \leftarrow \varnothing$ et $Div_j \leftarrow \varnothing$
    AJOUTE-ARCS-CONVERGENTS (i, j)
    AJOUTE-ARCS-DIVERGENTS (i, j)
    $C_{ij} \leftarrow C$.
    $E_{init} \leftarrow E_{init} \cup \{(i,j)\}$.
  PROPAGER (E_{init})

Si l'on considère par exemple le cas 6, c'est-à-dire l'ajout d'une contrainte étiquetée entre deux instants dont l'un seulement est numérique, on voit que seuls les Liens associé à i [resp. j]

doivent être recherchés, donc seuls les ensembles $Conv_i$ et Div_i [resp. $Conv_j$ et Div_j] sont construits.

Ajout d'une précédence en-dehors de $\mathcal{N}$

Les points 3 et 5, quoique eux aussi plus aisés à traiter, posent par contre des problèmes différents. Focalisons-nous d'abord sur le premier, c'est-à-dire l'ajout d'une précédence C_{ij}, entre deux instants $i \notin \mathcal{N}$ et $j \notin \mathcal{N}$. Là aussi, la contrainte $i \leq j$ est d'abord ajoutée dans $\mathcal{S}$ de manière à vérifier sa cohérence et mettre à jour le graphe symbolique. Après quoi, comme nous l'avons dit dans la section précédente, seul l'ensemble R_p est susceptible d'être modifié. En effet, la précédence C_{ij} est susceptible par transitivité d'induire de nouvelles précédences dans $\mathcal{N}$. Formellement (nous utilisons ici exceptionnellement la notation type CSP pour les contraintes symboliques):

$\forall (k, k') \in V_n^2$, l'ajout de S_{ij} dans $\mathcal{S}$ induit $S_{kk'} = S_{kk'} \cap (S_{ki} \oplus S_{ij} \oplus S_{jk'})$.

Les propriétés de l'algèbre d'instants nous permettent d'affirmer que: dès lors que k, i, j et k' sont distincts et $i \leq j$, le seul cas de figure où $S_{kk'}$ est susceptible d'être modifiée est $S_{ki} = S_{k'j} = '\leq'$. Auquel cas une précédence entre k et k' apparaît. Si $N_{kk'}$ est une contrainte faible, alors $N_{kk'} \leftarrow N_{kk'} \cap [0, +\infty[$. Cette modification est propagée dans $\mathcal{N}$.

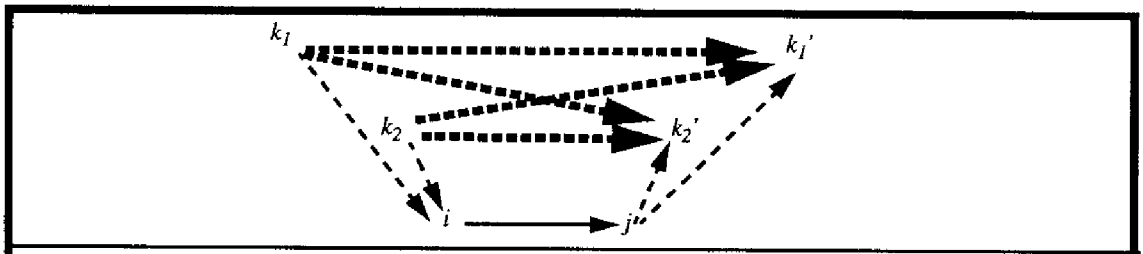


Figure 3.10. Liens induits par l'ajout d'une nouvelle précédence.

Le but du jeu, comme l'illustre la figure ci-dessus, est donc de déterminer les prédécesseurs k_π de i et les successeurs k_σ' de j, et pour chaque couple (k_π, k_σ'), effectuer l'opération d'intersection décrite plus haut. La détermination des prédécesseurs et des successeurs se fait selon le même principe que la recherche de Liens, en simplifiant les algorithmes: les ensembles $Conv_j$ et Div_i, ainsi que les instructions associées, sont ici inutiles. Cela conduit encore une fois à ne considérer que les précédences C_{ki} et $C_{k'j}$ minimales.

La complexité de ce type d'ajout découle de la combinaison d'une recherche type recherche de Liens, en $O(p.n)$, de p^2 opérations d'intersection, et de l'ajout et la propagation dans $\mathcal{N}$ de p nouvelles précédences minimales au maximum, soit au total $\mathbf{O(p.n + p^2 + p.m^2)}$ en pire cas. Ici aussi, même si l'on peut s'attendre à des temps de calcul moyens bien inférieurs à la complexité en pire cas, la dimension p reste un critère majeur d'efficacité, et la méthode devient

réellement performante dès que l'on a p faible, voire même borné, c'est-à-dire dans le cas de graphes fortement séquentiels.

Le dernier cas de figure, c'est-à-dire l'ajout d'une précédence entre deux instants dont l'un seulement est numérique, est de toute évidence le plus simple, et nous ne doutons pas que le lecteur saura aisément déduire les mécanismes utilisés de ce qui précède.

6. Requêtes Numériques Généralisées

6.1. Motivation

Nous allons dans cette section exposer très rapidement une autre fonctionnalité indispensable que nous avons provisoirement écartée au début de ce chapitre: comment gérer une requête numérique entre deux instants situés en-dehors de $\mathcal{N}$? En d'autres termes, le recours à un sous-graphe n'est qu'un processus interne permettant d'améliorer les performances du système, mais tout se passe pour le planificateur comme s'il ajoutait une contrainte dans un graphe numérique global $\mathcal{G}$. Il doit en être de même pour les requêtes: nous souhaitons pouvoir répondre à une requête numérique de telle manière que la propriété suivante soit vérifiée:

Propriété 3.6 : $\forall(i, j)\in V^2$, la réponse à une requête sur la distance numérique entre i et j dans la structure temporelle reposant sur $\mathcal{S}$ et $\mathcal{N}$ doit être identique à la réponse donnée à la même requête effectuée dans $\mathcal{G}$.

Du fait des propriétés de complétude de la méthode d'ajout dans le sous-graphe, si l'on ajoute i et j dans $\mathcal{N}$ et que l'on propage, on obtiendra forcément directement la contrainte minimale désirée. Cette solution n'est bien entendu pas souhaitable, augmentant inutilement la taille de $\mathcal{N}$, et allant donc à l'encontre du but visé, mais elle nous permet par contre d'énoncer différemment la propriété précédente:

Propriété 3.7 : $\forall(i, j)\in V^2$, la réponse à une requête sur la distance numérique entre i et j dans la structure temporelle reposant sur $\mathcal{S}$ et $\mathcal{N}$ doit être identique à la réponse donnée à la même requête effectuée dans un graphe $\mathcal{N}\cup\{i,j\}$ propagé.

En guise d'exemple introductif, remarquons que dans le graphe de la figure 3.7(c), nous pourrions par exemple chercher à connaître la contrainte numérique entre les instants non numériques 3 et 4. Dans ce cas-là, nous devons chercher des chemins empruntant des instants

numériques. Dans le cas des instants 3 et 4, rien ne vient contraindre la borne minimale, qui reste égale à 0 (du fait de la précédence entre 3 et 4). Par contre, la borne maximale de la contrainte numérique est bornée par la contrainte $N_{1,2}$, puisque $1 \leq 3$ et $4 \leq 2$. Nous devrons donc obtenir comme réponse à notre requête la contrainte minimale [0, 30] entre 3 et 4.

6.2. Description du Processus

Nous recherchons en fait la contrainte minimale entre i et j, dans un graphe global $\mathcal{N} \cup \{i, j\}$ qui par nature **n'est pas minimal**, c'est-à-dire que nous devons a priori considérer tous les chemins possibles. Nous cherchons donc inf_{ij} et sup_{ij} tels que:

- $inf_{ij}=\max_{k_1,...,k_n \in V_n} (inf_{ik_1}+inf_{k_1k_2}+...+inf_{k_nj})$,
- $sup_{ij}=\min_{k_1,...,k_n \in V_n} (sup_{ik_1}+sup_{k_1k_2}+...+sup_{k_nj})$.

Concentrons-nous sur la recherche de la **borne inférieure** inf_{ij}. Nous cherchons le chemin de contraintes pour lequel la somme des bornes inférieures est maximale. Ce chemin se compose de deux bornes inf_{ik_1} et inf_{k_nj} correspondant à deux contraintes situées en-dehors de $\mathcal{N}$ et donc par définition uniquement symboliques. Nous devons donc considérer leurs équivalents numériques qui sont

- $[0,+\infty[$ pour '$\leq$',
- $]-\infty,0]$ pour '$\geq$',
- $]-\infty,+\infty[$ pour '**?**'.

Une contrainte du type '$\geq$' ou '**?**' donnera nécessairement $inf_{ik_1}+inf_{k_1k_2}+...+inf_{k_nj}=-\infty$. Seules les contraintes $i \leq k_1$ et $k_n \leq j$ doivent donc être considérées. Pour les bornes supérieures, le même raisonnement conduit à la conclusion inverse: seules les contraintes $k_1 \leq i$ et $j \leq k_n$ sont dignes d'intérêt. De plus, nous pourrons bien sûr ici aussi nous restreindre aux seules précédences minimales.

Par ailleurs, comme $\mathcal{N}$ est complet, nécessairement:

$\forall k_1,...,k_n \in V_n$, $N_{k_1k_2} \oplus ... \oplus N_{k_{n-1}k_n} \supseteq N_{k_1k_n}$,

et de plus, il va falloir tenir compte des cas de figures où

- il n'existe qu'un seul instant k de $\mathcal{N}$ situé "entre" i et j (c'est-à-dire $i \leq k \leq j$, ou le contraire), auquel cas cela implique automatiquement une simple précédence, c'est-à-dire $inf_{ij}=0$, ou dans l'autre sens, c'est-à-dire $sup_{ij}=0$.
- il n'existe aucun instant "entre" i et j, donc aucun chemin dans $\mathcal{N}$ permettant de relier les deux instants, auquel cas il va falloir requérir la contrainte symbolique entre i et j directement dans S pour tester la présence éventuelle d'une précédence.

Toutes ces considérations conduisent à l'énoncé suivant, illustré par la figure 3.11:

Propriété 3.8 : $\forall i \notin \mathcal{N}$ et $j \notin \mathcal{N}$, la requête numérique entre i et j est traitée comme suit:

- Pour la borne inférieure:
  0. $inf_{ij} \leftarrow -\infty$;
  1. $inf_{ij} \leftarrow \max \{ inf_{kk'} / k,k' \in V_n$ ET $inf_{ik} \geq 0$ et $inf_{k'j} \geq 0) \}$;
  2. si $inf_{ij}=-\infty$, et si $\exists k \in V_n / inf_{ik}=inf_{kj}=0$, alors $inf_{ij} \leftarrow 0$;
  3. si $inf_{ij}=-\infty$, et si $i \leq j$ dans $\mathcal{S}$, alors $inf_{ij} \leftarrow 0$.
- Pour la borne supérieure:
  0. $sup_{ij} \leftarrow +\infty$;
  1. $sup_{ij} \leftarrow \min \{ sup_{kk'} / k,k' \in V_n$ ET $inf_{ki} \geq 0$ et $inf_{jk} \geq 0) \}$;
  2. si $sup_{ij}=+\infty$, et si $\exists k \in V_n / inf_{jk}=inf_{ki}=0$, alors $sup_{ij} \leftarrow 0$;
  3. si $sup_{ij}=+\infty$, et si $j \leq i$ dans $\mathcal{S}$, alors $sup_{ij} \leftarrow 0$.

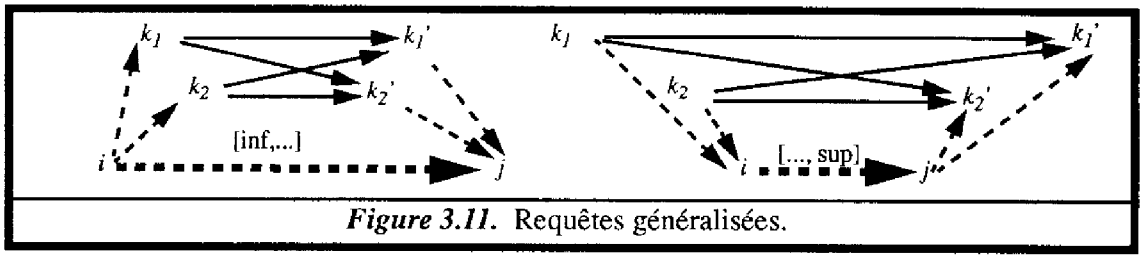


Figure 3.11. Requêtes généralisées.

A partir de là, les algorithmes à mettre en place se déduisent assez facilement, aussi bien pour ce cas général que dans le cas où l'un seulement des deux instants i et j est non numérique. Nous nous contenterons de donner les grandes lignes. Il s'agit encore une fois de déterminer les prédécesseurs et successeurs de i et j, par l'intermédiaire du calcul des ensembles $Conv_i$, $Conv_j$, Div_i et Div_j. Les figures témoignent d'ailleurs d'une similitude entre ce problème et la prise en compte dans $\mathcal{N}$ d'une précédence ajoutée (voir plus haut au § 5.7). Là où l'on cherchait toutes les précédences entre deux instants numériques dominées par C_{ij}, on va chercher ici toutes les contraintes numériques qui dominent et qui sont dominées par C_{ij}, mais les mécanismes sont identiques: il s'agit de calculer toutes les contraintes entre les prédécesseurs de i et les successeurs de j, mais aussi les contraintes entre les prédécesseurs de j et les successeurs de i.

Comme nous n'avons ici aucune propagation à déclencher, nous obtenons finalement une complexité en $\mathbf{O}(p.n + p^2)$, en tenant compte de la requête linéaire faite éventuellement en dernier recours entre i et j (étapes 3 dans la propriété ci-dessus). On peut remarquer que celle-ci n'est requise que si l'on ne trouve aucun instant "entre" i et j, ce qui arrive généralement dans les cas où i et j sont "proches" eu égard à la structure de rang de $\mathcal{S}$. Cette requête symbolique, comme dans la section précédente, sera donc en moyenne très peu coûteuse.

En guise de conclusion, signalons que l'avantage de disposer d'un réseau complet, et donc de requêtes en temps constant, n'est effectif qu'à l'intérieur de $\mathcal{N}$. Cependant, outre le fait que

l'on peut s'attendre à ce que notre planificateur ait plus fréquemment besoin de requêtes numériques entre instants numériques, nous pouvons remarquer que

1. notre système est **complet vis-à-vis des requêtes,**
2. et que les requêtes généralisées rendent compte d'une **complexité en O(n)** comparable aux requêtes symboliques dès lors que le graphe est séquentiel, c'est-à-dire dès que p est borné.

6.3. Vérification de Cohérence A Priori

Comme le lecteur a certainement pu le constater, la plupart des algorithmes présentés dans ces dernières sections reposent sur des principes similaires: relier des instants "étrangers" au sous-graphe $\mathcal{N}$en utilisant la structure de $\mathcal{S}$ pour déterminer leurs prédécesseurs et successeurs dans $\mathcal{N}$.

Un autre processus pourrait facilement être construit sur le même principe: vérifier la cohérence d'un ajout a priori. En effet, lorsqu'on ajoute une contrainte entre deux instants numériques, il est possible de commencer par regarder la contrainte courante dans le graphe complet $\mathcal{N}$pour voir si l'ajout est cohérent. Lorsque l'ajout est effectué entre deux instants situés hors de $\mathcal{N}$, cela n'est a priori plus possible: l'ajout est effectué de la manière décrite, et s'il y a incohérence, c'est en cours de propagation qu'elle sera détectée, laissant un graphe partiellement modifié. Ceci nécessite le rejet de la contrainte ajoutée et pose le problème du retour à la situation précédente.

En fait, les requêtes généralisées nous permettent, comme dans le cas d'un graphe complet, de tester dans tous les cas la valeur de la distance courante entre deux instants avant d'ajouter une contrainte entre ces mêmes instants. Cette vérification a priori doit donc être prise en compte dans le calcul de la complexité théorique globale d'une étape d'ajout incrémental, ce qui donne $O(p.(p + n + m^2))$, et $O(n + m^2)$ pour des réseaux fortement séquentiels.

7. Résultats Expérimentaux

Nous avons voulu effectuer des mesures expérimentales de la complexité de nos algorithmes, de manière à les comparer à une approche classique gérant le graphe numérique global $\mathcal{G}$, et donc à estimer le gain réel de notre méthode. En d'autres termes, nous désirons vérifier que le gain indiscutable apporté lors de la propagation proprement dite n'est pas contre-balancé par la complexité des algorithmes d'interaction entre $\mathcal{S}$ et $\mathcal{N}$.

Les algorithmes ont été écrits en CommonLisp et testés sur des Sun 4/75. Nous avons choisi de générer aléatoirement des graphes de tailles variables, de manière incrémentale. Nous construisons en parallèle $\mathcal{G}$ et $\mathcal{N}$ en appliquant les algorithmes précédemment explicités. A chaque étape, nous effectuons dans l'un et l'autre:

- l'**ajout** d'une contrainte étiquetée entre deux instants quelconques de $\mathcal{S}$,
- une **requête** numérique entre deux instants quelconques de $\mathcal{S}$.

Nous simulons donc de manière aléatoire tous les cas de figures présentés, ajouts entre instants numériques et entre instants non numériques, requêtes dans $\mathcal{N}$ et requêtes généralisées, de manière à obtenir une mesure moyenne des temps de calcul pour l'ensemble des cas. Notons qu'il s'agit de mesures statistiques réalisées sur 50 graphes aléatoires différents, dont la taille augmente jusqu'à 300 à 1000 points selon le type de mesure.

Enfin, une telle expérimentation est menée en maintenant constant le ration $r=m/n$. Nous reproduisons alors le même schéma pour des valeurs de r différentes, de manière à fournir des résultats en fonction

- du **type de méthode**: graphe global $\mathcal{G}$ ou sous-graphe $\mathcal{N}$,
- de la **taille** du graphe global n,
- du **ratio** $r=m/n$.

Par contre, nous ne mesurons pas l'influence de p, c'est-à-dire que le degré de parallélisme de nos graphes aléatoires augmente statistiquement de manière proportionnelle à m. Autant dire que dans le cas d'applications de planification où p est borné, nous obtiendrions des performances supérieures à celles qui apparaissent dans les figures suivantes.

7.1. Analyse d'une Opération d'Ajout

La figure 3.12(a) ci-après démontre que même pour un ratio de 50%, notre méthode s'avère payante au-dessus d'environ 50 instants dans le graphe. En-dessous de cette taille limite, une analyse plus fine donne des temps de calcul comparables pour les deux méthodes. Les écarts-types permettent également de constater que la méthode "classique" présente un taux de dispersion nettement plus important, rendant compte, sur certains cas particuliers d'ajout numérique, de temps de calcul dépassant la minute pour des tailles à peine supérieures à 100 instants.

Au-delà de 300 points, un ajout dans $\mathcal{G}$ prend plus de une minute en moyenne, et le taux d'occupation mémoire conduit fréquemment à une saturation de la mémoire dynamique allouée par défaut au processus d'exécution, rendant du même coup inutile la poursuite des tests. En effet, rappelons que la gestion d'un graphe $\mathcal{G}$ complet nécessite le maintien de n^2 contraintes numériques contre m^2 dans le cas de l'utilisation du sous-graphe numérique.

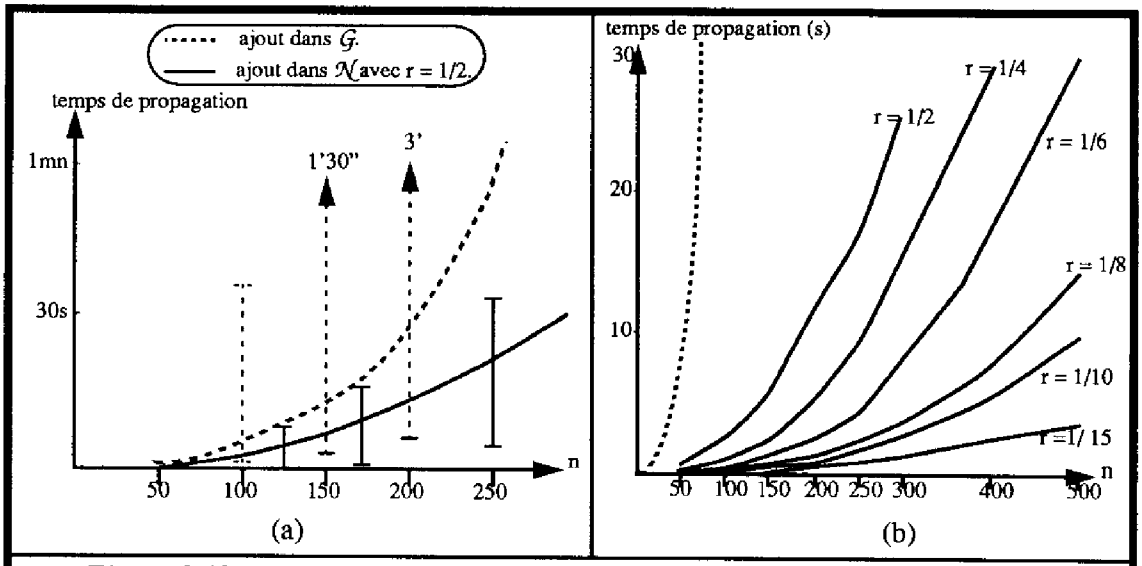


Figure 3.12. Temps de calcul expérimentaux d'une phase d'ajout dans $\mathcal{G}$ et $\mathcal{N}$.

La figure 3.12(b) analyse l'influence du ratio r sur le temps de calcul d'un ajout dans $\mathcal{N}$. Comme on pouvait s'y attendre, plus r est faible, plus la méthode est efficace, sans parler du gain en termes d'occupation mémoire.

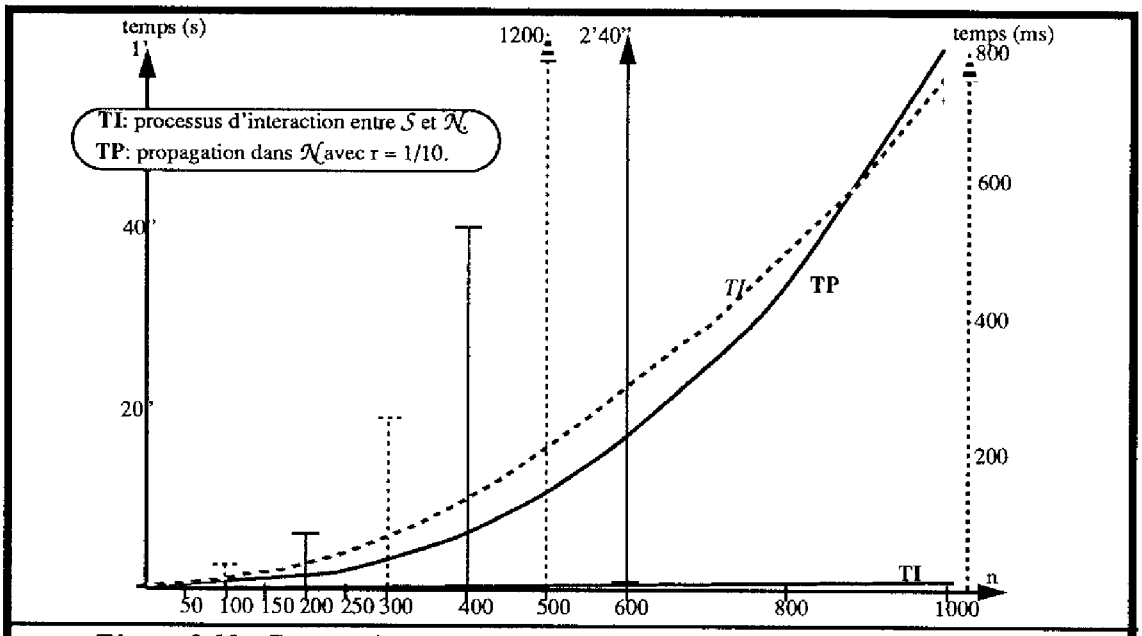


Figure 3.13. Comparaison des phases d'interaction et de propagation dans $\mathcal{N}$.

La figure 3.13 enfin analyse la place occupée par les algorithmes d'interaction (détection de Liens + report des précédences induites) dans le processus total d'ajout dans $\mathcal{N}$. Les mesures ont été faites pour un ratio r=10%, et démontrent que les algorithmes d'interaction sont négligeables aussi bien en valeur absolue qu'en ordre de complexité (en observant la courbe en

pointillés des processus d'interaction, reproduite à une échelle différente), ce à quoi l'étude théorique de complexité nous avait également préparés.

7.2. Opérations d'Interrogation de la Base

Les mesure de performances des opérations de requête sur la base conduisent à une courbe qui se confond dans la figure 3.13 ci-dessus avec la courbe TI. Cela n'est pas surprenant, les algorithmes étant en effet très proches. La complexité d'une requête généralisée est théoriquement plus élevée (cf § 6.2), mais ceci semble contre-balancé par la présence dans nos mesures des requêtes dans $\mathcal{N}$ par définition en temps constant. On obtient donc expérimentalement des complexités quasi-linéaires en moyenne pour une requête quelconque.

7.3. Bilan

Les mesures effectuées renforcent l'étude théorique effectuée notamment au § 5.6, et nous permettent d'apprécier le gain de notre méthode par rapport à l'approche classique. On peut faire les remarques générales suivantes:

- La méthode sera d'autant plus appréciée que l'on aura à gérer des graphes de taille importantes (de l'ordre de quelques centaines d'instants),
- Même si la méthode est avantageuse pour des ratios relativement faibles, elle perd vite de sa légitimité lorsque r dépasse les 50%: on peut s'attendre à ce que notre méthode ne commence à devenir payante que pour une taille de graphe n relativement importante au regard de nos applications, et d'autant plus importante que r se rapprochera de 100%.
- Enfin, si l'on rajoute le fait que p est le plus souvent borné lorsqu'on planifie pour un agent autonome, on peut s'attendre dans le cadre de telles applications à des complexités expérimentales inférieures, notamment linéaires en moyenne pour les processus d'interaction et de requêtes généralisées. Notre méthode est dans ce cas-là toujours avantageuse.

8. Inégalités et Contraintes Numériques

Les sections qui précèdent se sont attachées à faire le tour des interactions nécessaires entre $\mathcal{S}$ et $\mathcal{N}$pour rendre compte d'un fonctionnement global complet et sain. Il n'a cependant jamais été fait mention de la prise en compte des relations d'inégalité $\neq$ entre instants dans ce schéma

général. Les interactions entre $\mathcal{S}$ et I ayant été décrites dans le chapitre 2, il nous reste donc à étudier les **interactions entre $\mathcal{N}$et I**, c'est-à-dire tour à tour la **prise en compte d'inégalités** entre instants dans le sous-graphe numérique et la détection dans ce même sous-graphe de **nouvelles inégalités induites**.

Pour fixer les idées, on peut illustrer le cas d'une relation d'inégalité dans notre exemple. On pourrait en effet rajouter une contrainte $\neq$ entre les points 5 et 9, pour exprimer le fait qu'Albert et sa collègue de travail ne puissent pas pénétrer en même temps dans l'ascenseur.

8.1. Prise en Compte d'Inégalités dans le Graphe Numérique

L'existence d'une relation $(i \neq j)$ entre deux points i et j signifie en termes numériques qu'ils ne peuvent être séparés par une durée égale à 0. En d'autres termes, lorsqu'on ajoute une inégalité dans I, on doit a priori retirer la valeur 0 du domaine de la contrainte numérique N_{ij}. Cela a le mérite d'être à la fois simple et suffisant. Malheureusement, un inconvénient de taille apparaît dès que l'on cherche à l'appliquer au cas d'une contrainte faible:

> Soit $N_{ij}=[inf_{ij}, sup_{ij}]$, avec $inf_{ij}<0$ et $sup_{ij}>0$. Alors la prise en compte de $(i \neq j)$ modifie N_{ij} en $N_{ij}'=[inf_{ij},0[\cup]0,sup_{ij}]$.

La nouvelle contrainte n'est plus convexe, c'est-à-dire que nous sortons du cadre strict des STPs de [Dechter91], et ne pouvons plus prétendre à l'utilisation d'algorithmes polynômiaux. Heureusement, nous disposons d'une propriété qui va nous permettre de gérer les inégalités entre instants par l'intermédiaire d'une technique similaire à celle utilisée dans $\mathcal{S}$:

Propriété 3.9 : **Cohérence d'un réseau numérique en présence d'inégalités.**

- Soit $\mathcal{N}$un sous-graphe numérique minimal (donc cohérent) dans lequel aucune relation $\neq$ n'a été prise en compte explicitement.
- Soit $\mathcal{N}^{\neq}$ le même graphe minimalisé après la prise en compte, par l'opération $N_{ij}' \leftarrow N_{ij}/\{0\}$, de la contrainte $(i \neq j)$.

Alors:
Si $\mathcal{N}^{\neq}$ est incohérent Alors $N_{ij}=\{0\}$ dans $\mathcal{N}$.

Preuve:

En d'autres termes, si $\mathcal{N}^{\neq}$ est incohérent et $\mathcal{N}$est cohérent, cela signifie forcément que l'incohérence provient d'une inégalité $(i \neq j)$. A nous de démontrer qu'alors nécessairement $N_{ij}=\{0\}$ dans $\mathcal{N}$. Remarquons que nous noterons N_{ij} une contrainte de $\mathcal{N}$et $N_{ij}^{\neq}$ la contrainte correspondante dans $\mathcal{N}^{\neq}$.

La démonstration se base sur un résultat démontré par Amar Isli [Isli94], qui a étendu le paradigme des STPs aux cas des contraintes "trouées". Une contrainte trouée est du type $N_{ij}= [inf_{ij}, sup_{ij}] \setminus \{h_1, ..., h_\tau\}$, c'est-à-dire que l'on exclut de la contrainte un nombre fini de valeurs. Il montre que l'on obtient une algèbre, c'est-à-dire que les types de contraintes en entrée restent stables par composition et intersection. il nomme EPA son algèbre et démontre que l'on obtient pour EPA les mêmes résultats que ceux établis par P. van Beek pour $PA^{\neq}$, notamment la possibilité de démontrer la cohérence par l'intermédiaire d'un algorithme de filtrage, ainsi que la possibilité de déterminer le réseau minimal.

Nous pouvons considérer que nous nous situons dans un cas particulier obéissant à ce paradigme, c'est-à-dire que notre réseau $\mathcal{N}^{\neq}$ prend en entrée des contraintes du type classique $N_{kl}= [inf_{kl}, sup_{kl}]$ et des contraintes du type $N_{kl}= [inf_{kl}, sup_{kl}]\setminus\{0\}$. $\mathcal{N}^{\neq}$ est alors également constitué après propagation de contraintes du type $N_{kl}= [inf_{kl}, sup_{kl}] \setminus \{h_1, ..., h_\tau\}$. En observant les tables de composition et d'intersection présentes dans [Isli94], nous remarquons qu'un trou h_i apparaît dans une contrainte en cours de propagation du fait de l'existence d'un trou h_i' dans une autre contrainte. Pour le reste, les bornes inf_{kl} et sup_{kl} se calculent de la même manière. D'où l'on établit un premier résultat:

(1) si $N_{kl}^{\neq}=[inf_{kl}, sup_{kl}] \setminus \{h_1, ..., h_\tau\}$ dans $\mathcal{N}^{\neq}$ après propagation, alors $N_{kl} =[inf_{kl}, sup_{kl}]$ dans $\mathcal{N}$propagé

Un deuxième résultat peut être rapidement établi dans le cas où pour toute contrainte N_{kl} de $\mathcal{N}$, nous avons $inf_{kl} \neq sup_{kl}$, c'est-à-dire qu'aucune contrainte ne se réduit à un singleton. Dans ce cas-là, toutes les contraintes de $\mathcal{N}$ sont cohérentes, et chaque contrainte étant continue et non réduite à un singleton, alors il existe pour chaque contrainte N_{kl} une infinité de valeurs cohérentes. Comme N_{kl} et $N_{kl}^{\neq}$ ne se distinguent que par un nombre fini de valeurs h_i, alors nécessairement il existe également un nombre infini de valeurs cohérentes dans chaque contrainte $N_{kl}^{\neq}$. Donc $\mathcal{N}^{\neq}$ est cohérent.

Inversement, si $\mathcal{N}^{\neq}$ est incohérent (c'est-à-dire qu'il existe au moins un $N_{kl}^{\neq}= \emptyset$), alors nécessairement la contrainte correspondante dans $\mathcal{N}$ se réduit à un singleton: $N_{kl} = \{v_{kl}\}$, c'est-à-dire que $v_{kl}=h_i$ tel que h_i est un "trou" de $N_{kl}^{\neq}$. Ce qui nous donne:

(2) si $\mathcal{N}$ est cohérent et $\mathcal{N}^{\neq}$ est incohérent, alors $\exists\ N_{kl} = \{v_{kl}\}$ dans $\mathcal{N}$ telle que $N_{kl}^{\neq}= \emptyset$ dans $\mathcal{N}^{\neq}$.

Cette incohérence provient nécessairement de la contrainte initialement ajoutée $N_{ij}^{\neq}=N_{ij}\setminus\{0\}$. C'est-à-dire qu'il existe un chemin passant par $N_{ij}^{\neq}$ qui rend $N_{kl}^{\neq}$ incohérente. Nous allons montrer que N_{ij} se réduit elle aussi à un singleton dans $\mathcal{N}$ et que ce singleton est égal à $\{0\}$.

- Supposons tout d'abord que le chemin en question se réduise à deux contraintes $N_{ij}=[inf, sup]$ et $N_{i'j'}=[inf', sup']$, telles que $N_{ij} \oplus N_{i'j'} = N_{kl} = \{v_{kl}\}$ dans $\mathcal{N}$. Alors dans $\mathcal{N}^{\neq}$: $N_{ij}^{\neq}=[inf, sup]\backslash\{0\}$ et $N_{kl}^{\neq}= \emptyset$. L'incohérence de $N_{kl}^{\neq}$ provient de l'absence de la valeur 0 dans $N_{ij}^{\neq}$,c'est-à-dire que

$$\exists\, x' \in [inf', sup'] \,/\, v_{kl} = 0 + x'.$$

- Mais alors, il est également possible que

$$\exists\, \varepsilon \,/\, v_{kl} = (0+\varepsilon) + (x'-\varepsilon), \text{ avec } (0+\varepsilon) \in [inf, sup] \text{ et } (x'-\varepsilon) \in [inf', sup']$$

$$\exists\, \varepsilon' \,/\, v_{kl} = (0-\varepsilon') + (x'+\varepsilon'), \text{ avec } (0-\varepsilon') \in [inf, sup] \text{ et } (x'+\varepsilon') \in [inf', sup'],$$

auxquels cas v_{kl} serait une valeur cohérente de $N_{kl}^{\neq}$, ce qui contredit $N_{kl}^{\neq}= \emptyset$.

- En écrivant la contrapposée de ce qui précède, on obtient en particulier:

$$\nexists\, \varepsilon \,/\, (0+\varepsilon) \in [inf, sup] \text{ et } \nexists\, \varepsilon' \,/\, (0+\varepsilon') \in [inf, sup].$$

En d'autres termes, N_{ij} se réduit au singleton $\{0\}$ dans le réseau minimal de $\mathcal{N}$.

- Par simple récursivité, on déduit le même résultat pour un chemin de longueur quelconque.

CQFD.

Si l'on considère la contrapposée de la propriété énoncée, on obtient: S' il n'existe pas de $N_{ij}=\{0\}$ dans $\mathcal{N}$/ $i \neq j$ Alors $\mathcal{N}^{\neq}$ est cohérent. Ce qui suggère que l'on peut se contenter d'un graphe $\mathcal{N}$dans lequel on ignore a priori les relations $\neq$. En effet, une incohérence due à celles-ci pourra être facilement détectée durant la propagation: chaque fois qu'une contrainte est réduite à la seule valeur $\{0\}$, on va vérifier qu'il n'existe aucune relation $\neq$ entre les mêmes instants. Ceci est fait automatiquement par la fonction symbolique Collapse (cf chapitre2, § 2.3.3). L'algorithme de propagation numérique doit donc être modifié de la manière suivante:

- Durant la propagation numérique, $\forall (k,k') \in V_n^{\,2} \,/\, N_{kk'} \leftarrow \{0\}$,
  1. Effectuer au niveau symbolique Collapse(k,k'),
  2. si OK, poursuivre la propagation numérique: MODIF$\leftarrow$MODIF$\cup\{N_{kk'}\}$.
  2'. sinon, SORTIR("Incohérence due à une contrainte $\neq$!!!")

On est donc ramené au même cas de figure qu'au niveau symbolique: les relations $\neq$ peuvent être globalement ignorées dans la gestion des contraintes numériques, leur présence étant seulement testée pour vérifier la cohérence d'une agrégation d'instants induites par la propagation. De plus ce test se fait automatiquement par l'opération, de toutes façons nécessaire, d'agrégation symbolique.

On voit donc que l'on peut gérer de manière saine et complète les relations d'inégalité dans notre paradigme symbolique-numérique, augmentant ainsi sa richesse expressive, tout en con-

servant l'efficacité liée au formalisme des STPs. Par contre, il subsiste deux inconvénients:

- l'absence de complétude des requêtes vis-à-vis des inégalités, signalée au paragraphe 2.3.3 du chapitre 2,
- et le fait qu'une incohérence provenant d'une contrainte d'inégalité ne peut être détectée **qu'a posteriori**, c'est-à-dire en cours de propagation.

8.2. Détection d'Inégalités Induites

Dans le cadre des interactions entre $\mathcal{S}$ et I, nous avons vu que l'algèbre réduite PA à laquelle nous nous référons implicitement, ne peut, du fait de ses propriétés d'algèbre, induire de relations $\neq$, lesquelles appartiennent exclusivement à l'algèbre de points complète $PA^{\neq}$. Dans le cadre des interactions entre $\mathcal{N}$ et I, il est au contraire possible d'avoir de nouvelles contraintes d'inégalité induites par la propagation numérique. Cela survient chaque fois que la valeur 0 disparaît d'une contrainte numérique. Il suffit donc là aussi de rajouter le test suivant dans l'algorithme PC-2:

si $N_{kk'}=[inf_{kk'}, sup_{kk'}]$, avec $inf_{kk'} \leq 0 \leq sup_{kk'}$,
et si au cours d'une mise-à-jour, $N_{kk'} \leftarrow [inf_{kk'}', sup_{kk'}']$, avec $inf_{kk'}' > 0$ ou $sup_{kk'}' < 0$,
alors $R_i \leftarrow R_i \cup \{(k \neq k')\}$

8.3. Complexités

Les seules opérations supplémentaires sont des opérations de test en temps constant, et des opérations d'agrégation pour lesquelles nous renvoyons le lecteur au chapitre 2. Ces opérations d'agrégations sont en $O(n^2)$ en pire cas, mais les relations $\neq$ étant relativement rares dans nos applications de planification, nous pouvons de nouveau conclure que la prise en compte des relations d'inégalité entre instants n'a qu'un effet marginal en termes de complexité sur le fonctionnement global de notre système temporel.

9. Architecture Résultante du Gestionnaire de Relations Temporelles

Afin de donner au lecteur une vision globale des mécanismes utilisés par notre gestionnaire de contraintes temporelles hétérogènes, la figure suivante résume l'ensemble des interactions

entre les trois structures que sont le graphe symbolique $\mathcal{S}$, l'ensemble des relations $\neq$ I, et le graphe numérique $\mathcal{N}$. Il est à noter que l'arc (6) de la figure est représenté en pointillé car il utilise essentiellement, comme nous venons de le voir, l'opération Collapse présente au niveau de l'arc (0).

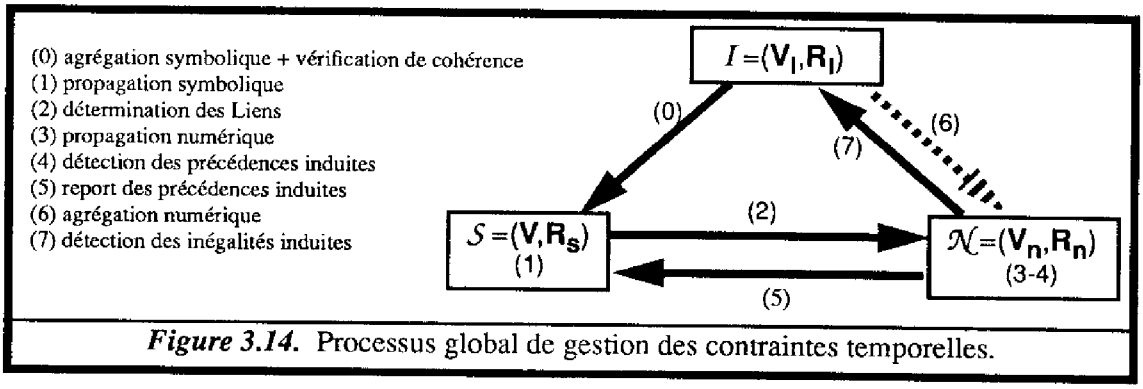


Figure 3.14. Processus global de gestion des contraintes temporelles.

N'oublions que ces trois graphes ne sont que virtuellement séparés, puisqu'ils représentent des CSPs distincts, mais reposant en fait sur le même ensemble de variables: rappelons que $V_i \subseteq V$, et $V_n \subseteq V$, ce qui se retrouve dans l'implémentation de notre méthode, I et $\mathcal{N}$étant codés à l'intérieur même du graphe d'instants global sur lequel repose $\mathcal{S}$.

10. Comparaison avec d'Autres Approches

Nous nous focaliserons ici sur les approches qui ont cherché à intégrer contraintes symboliques et numériques dans un même système de gestion de contraintes temporelles, que ce soit en planification ou dans d'autres domaines d'application. Les mêmes exigences au niveau temporel se retrouvent dans la plupart de ces domaines: **maintien de cohérence** et **requêtes**. On peut déjà évoquer rapidement des approches **incomplètes** telles que celle de [Dean89], dans le cadre de bases de données temporelles de grande taille, contenant des informations aussi bien sur le futur proche que sur le futur lointain. T.Dean propose des techniques de **focalisation** qui permettent de réduire la combinatoire du problème. Ces techniques de propagation locale sont cependant par nature incomplètes. Dans des domaines tels que la planification, l'ordonnancement, le diagnostic, on cherchera au contraire en priorité à s'assurer d'une technique complète en termes de vérification de cohérence.

T.Dean utilise également des techniques de **structuration hiérarchique** des informations temporelles pour accélérer les performances. Dans le domaine de l'exécution de plans conditionnels [Williamson93], une technique similaire est présentée, qui **dépend fortement de la**

structure inhérente au domaine. On obtient des complexités linéaires aussi bien au niveau des mises-à-jour que des requêtes. Dans le domaine de la gestion d'emplois du temps [Poesio91] nous est proposée une technique de hiérarchisation dépendant également du domaine, qui se base sur les STPs de Dechter, Meiri et Pearl. Néanmoins, dans le domaine de la planification, nous avons vu en analysant les travaux de J.Koomen [Koomen88] par exemple (cf chapitre 2, § 1.4), qu'il était difficile, et de fait peu bénéfique, de chercher à dégager une structure temporelle particulière liée au domaine.

Focalisons-nous sur les techniques complètes et indépendantes du domaine reposant sur des **réseaux de contraintes temporelles**. Nous avons déjà parlé des travaux de [Dorn92&93, Kautz91, Meiri91, Hertzberg91, Barber93], en indiquant notamment que la plupart choisissaient de se limiter au niveau de l'expressivité aux **classes polynômiales** de problèmes, c'est-à-dire l'algèbre d'instants et des contraintes de durée convexes. Un autre aspect important est la manière dont contraintes symboliques et numériques sont représentées. Dans certains cas, on ne conserve que des réseaux numériques, lesquels permettent de retrouver les contraintes symboliques [Hertzberg91, Barber93, Brusoni94] (cf chapitre 1, § 4.3.2). Dans d'autres travaux au contraire, on a préféré associer aux contraintes numériques et symboliques des **représentations et des techniques de propagation distinctes** [Kautz91, Meiri91, Tolba91&92, Gerevini93, Dorn93]. La raison en est souvent historique: des systèmes purement qualitatifs auxquels on a souhaité ajouter la prise en compte d'une dimension numérique. Dans notre cas, la raison principale est liée, comme nous l'avons dit, à la différence de complexité entre les deux types de propagation. Nous disposons de plus, au même titre que [Meiri91, Tolba92] et surtout [Gerevini93], dans TimeGraphII, de l'avantage de disposer d'**un seul réseau** pour supporter les deux CSPs, ce qui permet une meilleure flexibilité et donc une efficacité accrue, si l'on compare notamment avec les travaux de H.Kautz et P.Ladkin [Kautz91]. Cette plus grande **homogénéité de traitement** propre à notre système peut être également opposée à l'approche de J.Dorn [Dorn93], qui préfère disposer d'un ensemble d'intervalles reliés par des relations d'Allen, alors que les contraintes numériques s'expriment sous la forme de contraintes entre instants extrémités des intervalles, ce qui nécessite de transcrire constamment d'un formalisme vers l'autre. On trouve un principe similaire dans TACHYON [Stillman93], mais la représentation interne, tout comme dans [Meiri91], repose sur un réseau de type STP. Dans notre cas, la combinaison entre algèbre d'instant et formalisme type STP nous paraît constituer le choix le plus cohérent qui soit.

Un autre point de comparaison intéressant est le **compromis entre requêtes et mises à jour**. [Barber93], comme nous l'avons signalé au chapitre 2, propose deux types d'algorithmes de propagation. Le premier est avantageux en termes de complexité, mais ne rend pas compte des contraintes minimales. Si l'on souhaite disposer de requêtes complètes, celles-ci seront alors relativement complexes. Le deuxième type d'algorithme permet l'obtention du réseau minimal, ce qui est plus coûteux au niveau des mises-à-jour, mais rend compte par contre de

requêtes en temps constant. [Gerevini93] par exemple a choisi avec TimeGraphII d'utiliser le premier type d'approche. Le problème, comme nous l'avons clairement exposé au chapitre 2, est que cette approche ne permet pas de rendre compte des **précédences induites**. [Barber93] montre bien que si l'on souhaite vraiment disposer des contraintes minimales en réponse à une requête, celle-ci nécessite alors une complexité en $O(n.e)$, où n est le nombre d'instants et e le nombre de contraintes en entrée. [Dorn93] propose quant à lui un algorithme de propagation qui rend compte des précédences induites, mais cet algorithme est malheureusement soit incomplet, soit beaucoup trop complexe dès qu'il s'agit de traiter des relations de type *during*. De même, citons [Tolba91], qui utilise un algorithme de propagation assez lourd, du fait qu'il prend en compte des contraintes n-aires. Son algorithme rend compte de complexités comparables aux nôtres, mais n'est pas nécessairement complet. Signalons également que H.Tolba représente des "événements" disposant de durées fixes et de fenêtres de déclenchement, un peu à la manière de [Vere83]. Par ailleurs, H.Tolba développe tout comme nous une approche **incrémentale** de l'insertion des contraintes temporelles, au même titre que de nombreux travaux tels que [Barber93, Dorn93] par exemple.

Notre approche relève du deuxième type d'algorithme présenté ci-dessus, du fait de notre exigence en matière de **requêtes complètes** et de la **criticité** de ces requêtes en planification, qui doivent donc être privilégiées. Nous nous rapprochons en cela de l'approche de [Brusoni94] pour le système temporel générique LaTeR, qui calcule également le réseau minimal d'un STP par un algorithme de cohérence de chemin, de manière à pouvoir développer là-dessus des algorithmes de requêtes très performants, distinguant notamment, à la manière de [vanBeek91], les requêtes possibles des requêtes nécessaires. Les auteurs de LaTeR proposent également une technique de requêtes hypothétiques qui leur permet de repousser la propagation de leurs mises-à-jour à la fin d'une session, tout en répondant avec la même complétude aux requêtes émises. La différence principale entre LaTeR et le gestionnaire de relations temporelles d'IxTeT apparaît dans notre volonté de conserver une structure strictement symbolique en parallèle avec le réseau STP.

Signalons pour finir l'article de E.Yampratoom et JF. Allen [Yampratoom93] qui compare les **performances** de plusieurs systèmes temporels tels que Tachyon [Stillman93], TimeGraphII [Gerevini93], TimeLogic [Koomen88], ainsi que les systèmes temporels MATS de Allen et TMM de McDermott. Les conclusions principales sont les suivantes: MATS et TimeLogic rendent compte d'une propagation complète, et offrent à ce titre des performances plus faibles en comparaison avec TMM et surtout TimeGraphII, dont le compromis entre mises-à-jour et requêtes permet une meilleure efficacité globale. TimeGraphII offre les meilleures performances ainsi que la plus grande expressivité du fait de l'utilisation d'une structure de chaînes au niveau symbolique. Quant à Tachyon, il se contente de calculer une solution, mais ne permet pas d'effectuer des requêtes.

Cet article n'aborde cependant pas le problème de la complétude des requêtes. TimeGraphII et IxTeT sont par exemple comparable au niveau symbolique. Au niveau numérique par contre, TimeGraphII est plus performant au niveau des mises-à-jour, mais notre système est en mesure de rendre compte de bonnes performances à ce niveau dès lors que la proportion de contraintes numériques est suffisamment faible. Par contre, TimeGraphII, au contraire d'IxTeT, ne peut rendre compte des précédences induites que par l'intermédiaire de requêtes coûteuses. En fait, les deux approches sont défendables, **tout dépend du domaine d'application** visé par le système de gestion de contraintes temporelles.

11. Conclusion: Efficacité, Complétude et Flexibilité

Notre contribution se situe au niveau d'un réseau de **contraintes temporelles hétérogènes**, puisque nous distinguons non seulement contraintes symboliques et numériques, mais aussi inégalités. Mais ceci est fait au sein d'une **structure interne homogène**, puisque basée sur le même graphe réel d'instants, ce qui permet, outre une optimisation de la place mémoire occupée, une meilleure flexibilité, et donc des techniques d'interaction complètes, très efficaces et aisées à mettre en oeuvre.

Isoler les inégalités nous permet de profiter pleinement de l'efficacité de l'algèbre PA et des STPs, sans perte d'information, grâce à des techniques complètes et peu coûteuses d'interaction entre $\mathcal{I}$ et $\mathcal{S}$ d'une part, et $\mathcal{I}$ et $\mathcal{N}$ d'autre part. Cela nous différencie de techniques prenant directement en compte les inégalités telles que celles de [Gerevini93] pour la prise en compte des relations $\neq$ au niveau symbolique, ou de [Isli94] pour le traitement au niveau des STPs d'une notion plus large encore que celle des inégalités, à savoir les contraintes "trouées".

Séparer les contraintes symboliques et numériques nous permet par ailleurs de restreindre la propagation numérique coûteuse à un **sous-graphe**, ce que notre domaine d'application nous permet de faire. C'est-à-dire que nous parvenons à obtenir des **temps de mises-à-jour raisonnables**, tout en disposant de **requêtes complètes et efficaces** (en temps linéaire en pire cas, aussi bien dans le cas du symbolique que des requêtes numériques généralisées), ce qui constitue notre exigence primordiale. L'efficacité et la complétude des processus d'interaction sont à l'origine de ce résultat, qui permet notamment de disposer très facilement des précédences induites par la propagation numérique. A notre connaissance, IxTeT est le seul système à ce jour à proposer une telle complétude et efficacité conjointe dans un système intégrant deux structures distinctes de gestion de contraintes symboliques et numériques, et privilégiant les requêtes par rapport aux mises-à-jour.

Chapitre IV. Contrôlabilité des Contraintes Numériques: vers une Réelle Prise en Compte de l'Incertitude

Des deux problèmes évoqués à la fin du chapitre 2, il nous en reste encore un à traiter, à savoir la prise en compte de la notion de contrôlabilité d'une contrainte numérique. En effet, dire qu'une contrainte numérique prendra au moment de l'exécution du plan une valeur appartenant à un intervalle donné ne suffit pas. Il faut distinguer les contraintes dont on peut librement choisir la valeur, de celles pour lesquelles cette valeur est aléatoire. Seules les premières sont considérées implicitement dans une approche CSP. Les deuxièmes quant à elles introduisent une notion d'incertitude pour laquelle notre paradigme basé sur les STPs ne semble pas adapté a priori.

Une remarque préliminaire s'impose. La notion de contingence ne concerne que les données numériques. Nous reviendrons à la fin du mémoire sur ce qu'il convient de penser d'éventuelles contraintes symboliques contingentes. Pour l'instant, nous nous limiterons donc au cas d'un réseau purement numérique, de type STP, avec une origine 0, dans lequel on cherche à inclure des contraintes contingentes. Mais avant de définir plus précisément notre problématique, donnons-nous un petit exemple sur lequel nous pourrons appuyer les discussions qui vont suivre.

1. Un Exemple Illustratif

1.1. Description de l'Exemple

Ce qui va être décrit ci-après n'est encore une fois qu'un exemple purement informel et intuitif, dont le seul but est de fixer quelque peu les idées. La scène que nous cherchons à illustrer va d'abord être introduite dans une représentation simplifiée. La description sera complétée par la suite de manière à fournir un graphe un peu plus étoffé qui nous permettra d'illustrer des concepts plus avancés. Par ailleurs, comme nous allons le voir, l'exemple illustre un cas de figure où plusieurs agents effectuent des tâches en parallèle. Le domaine de la robotique, dans lequel nous nous sommes surtout intéressés à des applications de planification pour un agent unique, présente en effet également des applications dans lesquelles les activités de plusieurs robots doivent être gérées en parallèle (voir pour s'en convaincre les exemples de robotique dans le mémoire d'H.Laruelle [Laruelle94]). Ce type d'applications conduit le plus souvent à la nécessité de représenter des relations temporelles plus riches et donc plus complexes, pour lesquelles notre méthode va se montrer particulièrement adaptée.

Notre chercheur Albert est désormais en week-end, savourant un repos bien mérité. Il a décidé d'en profiter pour changer la télévision du salon. Nous nous plaçons dans une situation

initiale où son cousin Frank, qui travaille dans un magasin de hi-fi, vient d'arriver avec le nouveau récepteur. Il doit repartir immédiatement avec l'ancien, l'achat ayant été réalisé dans le cadre d'une opération classique de marketing "reprise de votre ancien appareil". Albert doit donc amener l'ancienne télévision jusqu'à la voiture de Frank, et revenir avec la nouvelle. Ceci constituera pour l'instant une seule et unique tâche appelée "Echange-Télé". Elsa, la femme d'Albert, se dit qu'elle en profiterait bien pour nettoyer un peu le meuble-télé, qui en a bien besoin, pendant que la place est libre. Ceci constitue une deuxième tâche que nous appellerons "Nettoyage". Nous avons donc deux tâches, représentées dans le graphe temporel par deux contraintes étiquetées, et reliées par une relation de type *during*, comme l'illustre les trois figures ci-dessous.

Ces trois figures représentent la même situation, mais avec des valuations différentes, données en minutes. Dans le deuxième cas par exemple (figure 4.1(b)), la tâche "Echange-Télé" prendra entre 7 et 9 mns, le minimum exprimant le cas "optimiste" où Albert est suffisamment en forme pour forcer l'allure, et le maximum étant le temps limite au-delà duquel Albert aura de toutes façons rejoint le salon, et où le poids important de l'appareil le forcera à le poser sans plus de délai. De même, la tâche "Nettoyage" prendra entre 6 et 8 mns, selon le degré de résistance de la poussière, dont Elsa n'a bien sûr qu'une estimation a priori.

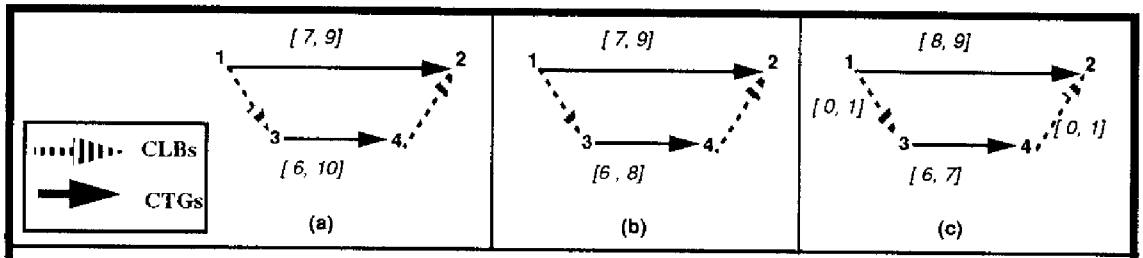


Figure 4.1. Un exemple graphique avec contraintes contingentes.

1.2. Présentation Intuitive du Problème

Nous sommes typiquement en présence de deux contraintes numériques $N_{1,2}$ et $N_{3,4}$ que l'on peut qualifier de contingentes, c'est-à-dire que ni Albert ni Elsa ne savent a priori combien de temps la tâche qu'ils doivent effectuer va leur prendre. Ils n'ont qu'une estimation sous la forme d'intervalles de durées possibles, qui ne peuvent être ni restreints ni étendus. Par exemple, si l'on applique un algorithme de filtrage type PC-2 au réseau 4.1(a), on obtient une réduction de la contrainte $N_{3,4}$ à l'intervalle [6, 9]. C'est-à-dire que les valeurs réelles comprises entre 9 et 10 sont localement incohérentes pour la contrainte $N_{3,4}$. Le problème est qu'il est **possible** que la contrainte $N_{3,4}$ soit effectivement instanciée avec la valeur 10 par exemple,

puisqu'on ne contrôle pas cette instanciation. Dans [Dorn94a], où l'on présente succinctement une dichotomie entre contraintes du même type que la nôtre, ce problème est traité en étendant simplement la notion de cohérence d'un CSP, de manière à retourner une incohérence dès qu'une contrainte contingente est restreinte.

Cela est malheureusement insuffisant, comme l'illustre la figure 4.1(b). Dans ce cas-là, un algorithme de filtrage reste sans effet sur les contraintes, d'où J.Dorn conclurait que le réseau est cohérent. Pourtant ce réseau est inacceptable. En effet, supposons que $N_{1,2}$=7, et $N_{3,4}$=8 (ce qui est susceptible d'avoir lieu pendant l'exécution puisque ces instanciations sont incontrôlables). Alors on voit clairement que l'on ne pourra jamais vérifier à la fois $1 \leq 3$ et $4 \leq 2$. C'est dire que ce réseau doit être également rejeté, puisque l'on n'est **pas sûr** que l'ensemble des contraintes soient satisfaites. En d'autres termes, nous voyons que ce réseau ne représente plus un problème de type STP. Il est de fait nécessaire de reformuler le problème en termes formels pour parvenir à dégager une méthode de résolution qui soit satisfaisante. Intuitivement, on peut deviner qu'il va s'agir de vérifier que l'on peut trouver une instanciation de toutes les variables quelles que soient les valeurs prises par les contraintes contingentes. Ceci va être formulé de manière plus rigoureuse dans les sections suivantes.

Avant cela, donnons simplement quelques commentaires sur le dernier cas, illustré par la figure 4.1(c), qui lui est pleinement acceptable. Le premier problème consiste comme nous venons de le voir à être capable de vérifier qu'il est effectivement "acceptable" (notion de **vérification de cohérence**). Mais il est également nécessaire d'être capable de déterminer les valeurs "acceptables" des contraintes qui ne sont pas contingentes, telles $N_{1,3}$ et $N_{4,2}$, et dont l'instanciation est donc de notre ressort (notion de **contraintes minimales**). La figure exprime le fait qu'il ne doit pas y avoir plus de 1 minute entre les instants 3 et 1. Dans le cas contraire, on retombera dans le cas (b), c'est-à-dire qu'il ne sera pas possible de s'assurer que $4 \leq 2$. Remarquons au passage qu'une propagation par cohérence de chemin dans le graphe de la figure 4.1(c) rendrait compte de contraintes $N_{1,3}$ et $N_{4,2}$ égales à [0, 3], ce qui dans le cas présent ne serait pas porteur d'information.

2. Définitions Préliminaires

Cette section a pour objet de définir clairement la terminologie employée et le problème posé. Nous allons notamment essayer de donner une première définition de ce que nous souhaitons vérifier dans notre nouveau réseau de contraintes, et de montrer clairement dans quelle mesure notre problème s'écarte des spécifications des STPs de Dechter, Meiri et Pearl, nécessitant de ce fait une formulation alternative.

2.1. Contingence et Liberté

Définition 4.1 :

- Une contrainte **contingente** est une contrainte numérique dont la valeur est incontrôlable et imprévisible, c'est-à-dire que la contrainte est assimilable à une variable aléatoire sur l'intervalle $[inf_{ij}, sup_{ij}]$. Nous utiliserons dans la suite l'abréviation **CTG**, et nous noterons une telle contrainte G_{ij}.

- Une contrainte **libre** est une contrainte numérique dont la valeur peut être a priori librement choisie à l'intérieur de son domaine $[inf_{ij}, sup_{ij}]$. Nous utiliserons dans la suite l'abréviation **CLB**, et nous noterons une telle contrainte L_{ij}.

Remarques :

1. Nous conserverons la notation générique N_{ij} utilisée jusqu'à présent pour illustrer des propriétés communes aux deux types de contraintes.
2. Nous supposons que les durées contingentes sont indépendantes les unes des autres. En effet, nous verrons que notre méthode ne convient que si les CTGs sont des variables aléatoires indépendantes (cf § 4.5 et § 6.3).
3. Une contrainte dont la durée est fixe $N_{ij}=\{v\}$ est par définition une CLB, puisque l'on peut "choisir" la valeur **v** dans son domaine. Par conséquent, une CTG $G_{ij}=[inf_{ij}, sup_{ij}]$ sera toujours telle que $inf_{ij} \neq sup_{ij}$.
4. Contrairement aux apparences, l'abréviation CLB ne se réfère pas au terme naturel et plus couramment utilisé dans la littérature [Dubois93] de contrainte "contrôlable", mais à celui plus inattendu de "contrainte libre" ... Nous verrons plus loin les raisons qui nous ont amené à effectuer ce choix.

Histoire de fixer quelque peu les idées, nous allons simplement évoquer l'exemple du chapitre 3 (cf figure 3.3), où nous avons quatre contraintes étiquetées:

- $N_{0,1}$: Albert est incapable de décider du moment où la porte de l'ascenseur va effectivement s'ouvrir, puisqu'il s'agit d'un événement qu'il ne contrôle pas, mais dont il connaît seulement la fenêtre d'occurrences possibles. Il devrait donc s'agir en fait d'une contrainte contingente.
- $N_{1,2}$: ici, la durée est fixe, c'est-à-dire qu'une seule valeur est possible; c'est donc une CLB par définition.
- $N_{6,7}$ et $N_{1,2}$: ces contraintes représentent les durées de deux tâches dont Albert maîtrise a priori totalement l'exécution. Il peut notamment courir [resp. parler] plus ou moins vite de manière à réduire ou augmenter la durée de la tâche correspondante (cf figure 3.6(c)). Il s'agit donc dans cet exemple de deux CLBs.

Signalons par ailleurs que nous pourrions faire une distinction similaire au niveau des variables, en considérant simplement le cas particulier des contraintes N_{0k}, dont la valeur est identique à la valeur prise par la variable k (cf chapitre 1, § 4.3.2). Dans la mesure où N_{0k} est une CTG, nous pourrions par exemple définir k comme étant une variable contingente, et de même pour les variables libres. Cette distinction n'apporte cependant rien, et rend compte d'une sémantique assez floue, susceptible d'induire le lecteur en erreur. Nous préférerons nous en tenir à la stricte démarcation entre deux types de contraintes en entrée. Une différentiation plus formelle entre deux types de variables sera introduite dans la prochaine section.

2.2. Observations et Décisions

Par contre, séparer deux types de contraintes conduit naturellement à distinguer deux types de valeurs pouvant être affectées à ces contraintes. Rappelons que le lien entre valeur d'une contrainte et contrainte proprement dite est du même ordre que le lien entre variable et domaine de la variable.

Définition 4.2 : **Durée effective.**

Nous appellerons durée effective la valeur affectée à une contrainte dans le cadre d'une solution particulière. Nous la nommerons et la noterons différemment selon le type de la contrainte:

- Une **durée observée** est la valeur effective d'une CTG. Nous la noterons ω_{ij}.
- Une **durée décidée** est la valeur effective d'une CLB. Nous la noterons x_{ij}.

2.3. Problème Temporel avec Incertitude (STPU)

Dès lors, il est possible de donner une première définition de notre problème temporel, qui se démarque des STPs par l'introduction des dichotomies précédemment définies. Il s'agit donc du même coup pour ce qui nous concerne d'une redéfinition de notre graphe numérique $\mathcal{N}$. Nous utiliserons pour dénommer notre nouveau type de problème l'acronyme anglo-saxon STPU (pour "Simple Temporal Problem under Uncertainty"), par référence aux STPs dont il est issu.

Définition 4.3 : Un **Problème Temporel avec Incertitude** (**STPU**) est la donnée d'un quadruplet $\mathcal{N}=\{V_N, D_N, \Gamma, \Lambda\}$, où

- $V_N=\{x_1, ..., x_m\}$ est l'ensemble des m instants numériques (assimilés à leurs dates),
- $D_N=D_1 \times ... \times D_m$ est le produit cartésien de leurs domaines,

- $\Gamma = \{G_{ij} \in R_n / (i,j) \in V_N^2$, et G_{ij} est une CTG$\}$ est l'ensemble des CTGs d'entrée,
- $\Lambda = \{L_{ij} \in R_p / (i,j) \in V_N^2$, et L_{ij} est une CLB$\}$ est l'ensemble des CLBs d'entrée,

Notons que ceci redéfinit du même coup notre réseau $\mathcal{N}=\{V_N, \Gamma \cup \Lambda\}$. Pour simplifier le propos, nous confondrons dans la suite le STPU et sa représentation graphique $\mathcal{N}$. Par ailleurs, Γ et Λ expriment une partition en fonction de deux types de contraintes d'entrée (les CTGs et les CLBs). Les ensembles R_n et R_p ne sont donnés ici que pour faire le lien avec la partition définie dans le cadre du sous-graphe numérique au chapitre précédent (au § 3.2 pour être précis), où l'on distinguait parmi les contraintes numériques ($R_n=R_e \cup R_p$) les contraintes étiquetées (R_e) et les Liens (R_p). Ils permettent de voir qu'un Lien sera par définition une contrainte libre, puisqu'il exprime une simple contrainte de précédence, initialisée à $[0,+\infty[$, mais qui peut être restreinte par la suite en fonction des autres contraintes présentes, sans que cela remette en cause son statut de précédence. Une contrainte étiquetée pourra par contre être soit contingente, soit libre. Nous ne reviendrons pas par la suite sur cette caractérisation, car nous considérons tout d'abord le STPU sous un angle général d'extension d'un STP classique, donc purement numérique, et non en lien avec le gestionnaire de relations temporelles de IxTeT. Ce lien sera détaillé de manière plus ou moins informelle à la fin de ce chapitre (cf section 5).

Nous allons donc simplement considérer la partition des contraintes entre Γ et Λ pour caractériser un STPU. On constate à partir des définitions précédentes que cette partition est à la fois **stricte** et **complète**: une contrainte d'entrée sera soit une CLB, soit une CTG, et ne pourra être les deux à la fois. En particulier, nous interdisons la possibilité d'avoir pour un couple d'instants (i, j) à la fois une contrainte G_{ij} et une contrainte L_{ij}. Nous interdisons également d'avoir deux CTGs concurrentes sur un même couple de points, ce qui exprimerait deux variables aléatoire différentes sur un même événement. Le seul cas de figure autorisé est de fait la conjonction de deux CLBs différentes entre deux instants, auquel cas on peut appliquer l'opération d'intersection classique.

2.4. Une Mise en Perspective

Comme nous l'avons suggéré en introduction à ce chapitre, un CSP dans sa généralité, et un STP en particulier, ne gère implicitement que des contraintes libres. En effet, le domaine de la contrainte représente l'ensemble des valeurs que l'on peut **choisir** de lui affecter, cet ensemble pouvant être réduit pour éliminer les valeurs qui sont incohérentes avec les autres contraintes présentes dans le réseau. C'est là le rôle des algorithmes de filtrage tels que l'algorithme PC-2 utilisé par notre gestionnaire de contraintes temporelles numériques. Comme nous l'avons illustré intuitivement au travers de notre exemple, prendre en compte des CTGs amène donc à considérer des contraintes qui sortent littéralement du cadre formel des CSPs.

Par contraste, on peut s'intéresser au cas des réseaux PERT [Kaufman69], dans lesquels on retrouve des tâches partiellement ordonnées, chacune étant affectée d'une durée. Cette durée peut être fixe, dans les cas les plus simples, ou bien représentée par une variable aléatoire. C'est-à-dire que l'on considère que cette durée ne sera connue que pendant le déroulement du programme. Ces durées représentent donc des contraintes contingentes, sur lesquelles il va falloir raisonner pour déterminer la **faisabilité du programme** dans les délais imposés. Un réseau PERT peut donc être vu comme un réseau où l'essentiel des contraintes d'entrée sont contingentes, un degré de liberté apparaissant néanmoins au niveau des contraintes de précédence entre tâches, pour lesquelles on peut définir une notion de "marge".

Notre problématique est similaire dans le sens où nous souhaitons également vérifier la faisabilité du plan en cours, en fonction de durées et de dates aléatoires. Contentons-nous dans un premier temps d'indiquer intuitivement (nous reviendrons sur ces aspects à la fin du chapitre) les deux axes essentiels selon lesquels nous allons nous démarquer des techniques PERT:

1. Les CLBs jouent dans nos réseaux un rôle plus important, définissant des **contraintes plus complexes** que les simples précédences entre tâches.
2. Nous chercherons à nous ramener à une **formulation de type CSP**, afin de bénéficier des techniques efficaces de détermination du réseau minimal par filtrage.

2.5. Contrôlabilité et Solution d'un STPU

Nous allons tenter dans ce paragraphe de donner une première définition formelle de notre problème. En d'autres termes: que souhaitons-nous vérifier dans un STPU ? Puisque un STPU ne répond plus aux caractéristiques de type CSP des STPs, la cohérence du réseau ne peut pas être définie dans les mêmes termes. C'est pourquoi nous allons introduire ci-après un nouveau concept que nous dénommerons **contrôlabilité** du STPU. Mais procédons par ordre, en donnant tout d'abord un sens précis à la notion de *situation*:

Définition 4.4 :

- L'**espace des observations** Ω d'un STPU $\mathcal{N}$est le produit cartésien des CTGs: si $\Gamma = \{G_1=[inf_1, sup_1], G_2=[inf_2, sup_2], \ldots G_g=[inf_g, sup_g]\}$, alors $\Omega=[inf_1, sup_1]\times[inf_2, sup_2]\times\ldots\times[inf_g, sup_g]$.
- Une **situation** ω d'un STPU $\mathcal{N}$est alors une occurrence possible de l'espace des observations, c'est-à-dire un point $\omega=\{\omega_1, \omega_2, \ldots \omega_g\}\in\Omega$

Dans le cas de l'exemple, $\omega=\{\omega_{1,2},\omega_{3,4}\}$ est la *situation* que l'on "subit". Il est possible d'étudier notre problème initial dans une *situation* particulière ω, ce qui définit un nouveau

problème que nous noterons $\mathcal{N}_\omega$, et pour lequel il existe éventuellement une solution. Pour pouvoir définir formellement $\mathcal{N}_\omega$, on notera par convention $\{\omega\}=\{\ \{\omega_{ij}\} / \omega_{ij}\in\omega\}$, c'est-à-dire que l'on se donne un ensemble de contraintes réduites à des singletons, où chaque singleton contient la valeur de la contrainte présente dans ω. Dès lors, $\mathcal{N}_\omega$ se définit comme suit:

Définition 4.5 : $\forall\omega\in\Omega$, l'**instance** $\mathcal{N}_\omega$ de $\mathcal{N}$ dans la *situation* ω est la transformée du STPU $\mathcal{N}$ définie comme suit:
- $\forall\ G_{ij}\in\Gamma$, G_{ij} est remplacée par $\{\omega_{ij}\}$ tel que $\omega_{ij}\in\omega$.

Formellement, on a alors: $\mathcal{N}_\omega = \{V_N, D_N, \Lambda'=\Lambda\cup\{\omega\}\}$, où $\{\omega\}=\{\ \{\omega_{ij}\} / \omega_{ij}\in\omega\}$.

Du même coup, $\Lambda'=\Lambda\cup\{\omega\}$ constitue par définition un ensemble de CLBs, puisque nous avons vu qu'une contrainte réduite à un singleton était par définition une CLB (cf § 2.1). Ce qui induit la propriété suivante:

Propriété 3.1 : $\forall\omega\in\Omega$, $\mathcal{N}_\omega$ est un STP répondant aux définitions de [Dechter91].

Nous pouvons désormais définir notre problème. Il s'agit pour nous de vérifier que l'on peut affecter une valeur à chaque variable quelles que soient les valeurs des CTGs, c'est-à-dire quelle que soit la *situation* ω. Ceci définit ce que nous appellerons la contrôlabilité du STPU $\mathcal{N}$. Cela revient à dire qu'il existe une solution pour toute instance. Formellement, nous pouvons donner les énoncés équivalents suivants, qui découlent des définitions précédentes:

Définition 4.6 : Un STPU $\mathcal{N}$ est **contrôlable**
ssi $\forall\omega\in\Omega$, $\mathcal{N}_\omega$ est cohérent.
ssi $\forall\omega\in\Omega$, $\exists\ \delta=\{x_1\in D_1, ..., x_n\in D_n\} / \forall C_{ij}\in\Lambda'$, $(x_j-x_i)\in C_{ij}$.

Dans le cadre de notre exemple, cela donne, par exemple dans le cas de la figure 4.1(b): $\forall\omega_{1,2}\in[7,9]$, $\forall\omega_{3,4}\in[6,8]$, existe-t-il $\delta=\{x_{1,3}, x_{4,2}\}$ tel que
- $x_{1,3}\geq 0$,
- $x_{4,2}\geq 0$,
- $x_{1,3} + \omega_{3,4} + x_{4,2} = \omega_{1,2}$?

Remarques :

1. On peut exprimer les choses de manière légèrement différente, en considérant simplement qu'un STPU est un problème de décision dont Γ représente l'ensemble des **paramètres** aléatoires. Une instance $\mathcal{N}_\omega$ est alors un problème disposant de paramètres constants, ce qui permet de le traiter comme un CSP classique.

2. Nous pourrions également définir, à l'image des travaux de H. Fargier, J. Lang et T. Schiex [Fargier95] (sur lesquels nous reviendrons à la fin de ce chapitre), une notion de **contrôlabilité forte**. Celle-ci se définit comme l'existence d'une solution unique pour toutes les instances $\mathcal{N}_\omega$, c'est-à-dire que l'on peut construire une solution a priori, valable dans toute *situation*. Par définition, la contrôlabilité forte induit la contrôlabilité. Ceci constitue une exigence qui n'est pas nécessaire dans nos applications de planification, puisque, rappelons-le, une affectation des variables sera effectuée en fait en cours d'exécution, donc au fur et à mesure que les *durées observées* sont connues. De plus, la forte contrôlabilité prévaut en fait assez rarement dans des STPUs tels que ceux que nous avons à traiter en planification. Nous nous contenterons de la contrôlabilité, qui est moins exigeante et correspond précisément à ce que nous souhaitons vérifier, comme nous venons de le voir. Signalons simplement que notre exemple illustratif est non seulement contrôlable, mais de plus fortement contrôlable.

Il ne reste plus alors qu'à définir ce que nous entendons par solution d'un STPU, ce qui va pouvoir être fait sans plus de délais:

Définition 4.7 : Une **solution** d'un STPU $\mathcal{N}$ sera une application σ qui à tout $\omega \in \Omega$ associe un élément $\delta=\{x_1 \in D_1, ..., x_m \in D_m\} = \sigma(\omega)$ tel que δ est solution de $\mathcal{N}_\omega$.
A partir de quoi la contrôlabilité peut être redéfinie ainsi:
- $\mathcal{N}$ sera contrôlable ssi $\exists$ une solution σ ainsi définie.
- $\mathcal{N}$ sera fortement contrôlable ssi $\exists\ \sigma$ application constante sur Ω ($\exists \delta$ / $\forall \omega \in \Omega$, $\delta=\sigma(\omega)$).

De fait, nous disposons désormais de définitions formelles fixant le cadre de notre recherche, à savoir la vérification de la contrôlabilité d'un STPU. Les CTGs étant par définition des contraintes continues, il est bien entendu impensable de se ramener effectivement à une famille de CSPs $\mathcal{N}_\omega$. Néanmoins, ces définitions préliminaires vont nous conduire petit à petit à la construction d'un CSP global, représentation duale d'un STPU, dans lequel nous pourrons vérifier aisément la contrôlabilité du STPU initial.

2.6. Réseau Minimal: une Exigence Toujours Présente

Nous avons vu aux chapitres 2 et 3 que vérifier la cohérence d'un réseau de contraintes temporelles lors d'une mise-à-jour ne suffisait pas. Il fallait également être en mesure de rendre compte de manière efficace des **contraintes minimales**. Cela passait nécessairement dans le cas numérique par une propagation des contraintes rendant compte du réseau minimal. Ce rappel figure ici dans le seul but de signaler au lecteur que cette exigence se retrouve forcément dans le cas des STPUs. Nous devons donc non seulement maintenir la contrôlabilité du STPU,

mais également chercher à rendre compte d'un "réseau minimal" qui reste à définir. Que le lecteur garde donc cette préoccupation essentielle à l'esprit. Nous y reviendrons lorsque nous aurons défini notre nouveau paradigme temporel.

3. Vers la Définition d'un Problème Dual

L'idéal consiste en fait à se ramener à un CSP classique. Il va pour cela falloir identifier au sein du STPU un nouvel ensemble de variables et extraire les contraintes qui pèsent sur ces variables. Après quoi il faudra non seulement prouver que le problème dual ainsi construit autorise l'utilisation d'algorithmes de propagation classiques, mais encore qu'il permet de vérifier la contrôlabilité du problème initial. Comme nous allons le voir, il suffit pour cela de se ramener à un sous-ensemble des variables de départ. Mais n'anticipons pas trop sur la suite, et commençons plutôt par restreindre notre problème initial par l'intermédiaire d'**hypothèses restrictives**. Plusieurs raisons à cela:

1. la plupart des problèmes réels que nous avons à traiter obéissent à cette restriction,
2. la formalisation du problème dans ce cadre peut être présentée d'une manière **plus claire** pour le lecteur, car plus proche de l'intuition,
3. et les propriétés sur lesquelles nous allons nous appuyer sont **aisément démontrables** dans ce cadre.

Néanmoins, nous chercherons par la suite à lever ces hypothèses pour étudier dans quelle mesure notre paradigme peut être étendu à des représentations plus complètes. Ce travail, qui reste inachevé sur le plan formel, sera simplement esquissé à la fin de ce chapitre (section 6).

3.1. Hypothèses Restrictives

Les hypothèses suivantes ont pour objet de restreindre les types d'insertions pouvant être effectuées dans le réseau. A la fin du chapitre 2, nous avons présenté le problème de l'incertitude temporelle en planification, en expliquant que les durées des tâches ou les dates des événements étaient en fait le plus souvent des contraintes contingentes. C'est ce cas de figure fréquent qui a motivé le développement de la méthode présentée dans ce chapitre. En d'autres termes, dans le cadre du processus incrémental décrit dans les chapitre précédents, il s'agira le plus souvent d'ajouter une CTG forte dans un réseau $\mathcal{N}$, ce qui suppose comme nous l'avons vu la détermination et l'ajout conjoint de Liens, c'est-à-dire de CLBs fortes initialisées à

$[0,+\infty[$. On élimine de fait les cas d'ajout d'une contrainte étiquetée de type CLB, et plus généralement d'ajout de contraintes faibles, pour se limiter aux cas suivants:

1. Une simple précédence pourra être ajoutée dans $\mathcal{N}$, ce qui se traduira par une CLB $L_{ij}=[0,+\infty[$.
2. Une contrainte étiquetée ajoutée sera nécessairement CTG et forte, c'est-à-dire qu'il s'agira de contraintes $G_{ij}=[inf_{ij}, sup_{ij}]$, avec $inf_{ij}.sup_{ij} \geq 0$.
3. L'ajout d'une CTG s'accompagnera également de l'ajout de Liens, c'est-à-dire de CLBs $L_{ij}=[0,+\infty[$.

$\mathcal{N}$sera donc exclusivement composé de **CTGs fortes reliées par de simples précédences**. Une CTG sera toujours forte, donc orientée d'un instant de début vers un instant de fin, ce qui conduit aux notations suivantes:

Notation :

Nous noterons désormais G_k un élément quelconque de Γ.
$\forall G_k \in \Gamma$, son instant de début sera noté d_k et son instant de fin f_k.

Cette notation provient du fait que nous sommes amenés ici à définir les variables à partir des contraintes qui pèsent sur elles, et non le contraire. En effet, nous sommes partie d'une dichotomie des contraintes d'entrée pour arriver dans le paragraphe qui suit à une dichotomie des variables du STPU.

Signalons également en guise de restriction supplémentaire que nous ne prendrons pas en compte dans la suite les inégalités $\neq$ entre instants, sur lesquelles nous reviendrons succinctement à la fin de la section 6 consacrée aux extensions.

3.2. Classification des Variables et des Contraintes

3.2.1. Dichotomie des Variables

Une typologie des contraintes peut être désormais aisément introduite. En effet, trouver une solution à notre problème consiste à affecter une date à chaque instant-variable. Or les instants de fin de chaque CTG sont affectés par définition d'une date qui dépend de la valeur prise par la CTG elle-même, c'est-à-dire que $\forall G_k \in \Gamma$, la date de f_k dépend de ω_k. Par contre, les variables début d_k de CTGs constituent les véritables **variables de décision** du STPU, dont on doit pouvoir "choisir" la valeur pour déterminer une solution. Ceci conduit à la définition suivante:

Définition 4.8 : Variables d'Attente, Variables de Déclenchement.

- Une **variable de Déclenchement** sera un instant i du graphe tel que $\exists\ G_k \in \Gamma\ /\ i \equiv d_k$.
- Une **variable d'Attente** sera un instant i du graphe tel que $\exists\ G_k \in \Gamma\ /\ i \equiv f_k$.

Notation :

L'ensemble des variables de Déclenchement sera dénoté $\mathbf{V_D}$ et l'ensemble des variables d'Attente $\mathbf{V_A}$.

Les termes choisis sont liés à la manière dont une solution sera déterminée, c'est-à-dire qu'à l'exécution, les variables d_k représentent le moment où l'on choisit de déclencher une CTG (laquelle correspond généralement à une tâche), alors que les variables f_k représentent des événements dont le planificateur attend l'occurrence pour pouvoir poursuivre l'exécution du plan en cours (fins de tâches ou événements attendus).

Il est aisé de démontrer que nous avons défini là à nouveau une **partition stricte et complète**: $V_N = V_D \cup V_A$. Stricte car une variable ne peut être à la fois attendue et déclenchée, complète car tout instant appartient forcément à une catégorie ou bien l'autre. En effet, $\mathcal{N}$ ne contient que les instants numériques (cf chapitre 3), c'est-à-dire les instants début ou fin d'une contrainte étiquetée, donc, dans le cadre de nos hypothèses, les extrémités des CTGs.

3.2.2. Extension de la Typologie des Contraintes

Cette dichotomie entre variables conduit à une extension de la typologie des contraintes d'entrée. Parmi celles-ci, nous avons distingué les contraintes étiquetées et les Liens. Un Lien est forcément une CLB forte, et une contrainte étiquetée est par hypothèse une CTG forte. A partir de là, on peut distinguer deux types de CLBs. L'exemple de la figure 4.1 nous présente deux spécimens de CLBs fortes que sont $L_{1,3}$ et $L_{4,2}$. Leur sémantique est cependant très différente. $L_{1,3}$ exprime un laps de temps précédant le déclenchement de la tâche "Nettoyage", $L_{4,2}$ exprime un laps de temps précédant l'occurrence de la fin de "Echanger-Télé", c'est-à-dire qu'elles se synchronisent, l'une sur une variable de Déclenchement, l'autre sur une variable d'Attente. Nous les appellerons de ce fait contraintes de Déclenchement ou contraintes d'Attente.

Définition 4.9 :

- Une **contrainte de Déclenchement** est une CLB $L_{kk'}$ telle que k' est une variable de Déclenchement.
- Une **contrainte d'Attente** est une CLB $L_{kk'}$ telle que k' est une variable d'Attente.

La figure ci-dessous résume la typologie ainsi définie. Nous avons représenté en pointillés les cas de figures qui sont exclus du fait de nos hypothèses restrictives.

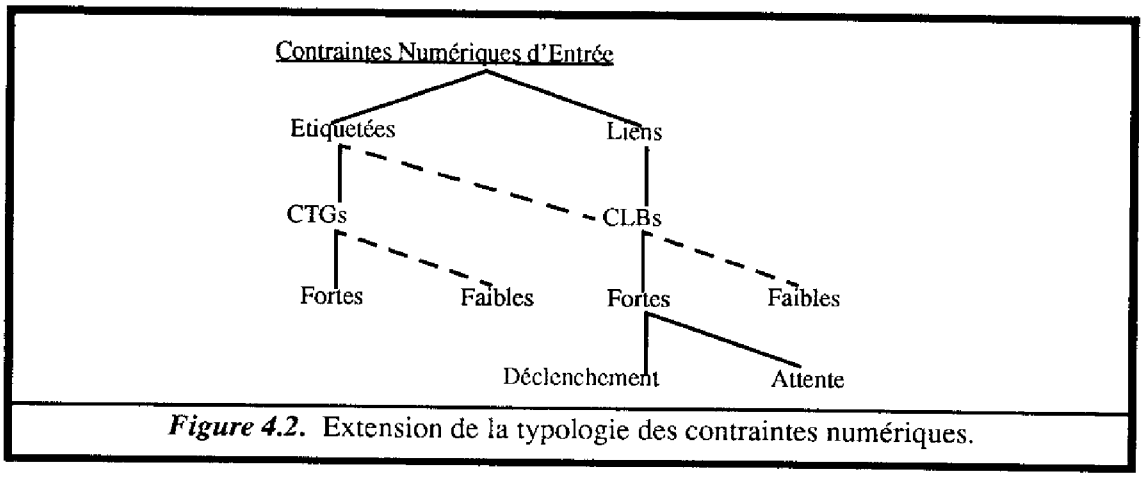


Figure 4.2. Extension de la typologie des contraintes numériques.

Remarque : Une contrainte de Déclenchement est directement contrôlable, c'est-à-dire que l'on pourra effectivement choisir la valeur à lui affecter. Par contre, cela n'est pas vrai pour une contrainte d'Attente. Il s'agit cependant d'une contrainte **libre** dans le sens où l'on peut librement la restreindre par filtrage. En d'autres termes, on ne peut pas "décider" qu'une contrainte d'Attente prenne telle ou telle valeur, mais on peut "décider" qu'elle **ne prendra pas** telle ou telle valeur. Dans notre exemple (figure 4.1(c)), on peut être amené à interdire à la contrainte $L_{4,2}$ de durer moins d'une minute, en la restreignant à [1,1]={1}, ce qui revient à forcer $L_{1,3}$ à durer 0. Ceci explique pourquoi nous avons préféré utiliser le terme de contrainte libre en lieu et place du terme plus courant de contrainte contrôlable (cf § 2.1).

3.3. Redéfinition du Problème

Le lecteur aura deviné de lui-même où nous voulons en venir. Le problème initial va être restreint aux seules variables de Déclenchement. Cela suppose de transformer l'ensemble des contraintes du STPU de départ en contraintes pesant sur ces seules variables. Pour cela nous allons tout d'abord donner une formulation équivalente de la contrôlabilité du graphe. L'idée est la suivante: une instance $\mathcal{N}_\omega$ est un STP, donc elle est cohérente si et seulement et si elle est 2-cohérente, ce qui donne

- Un STPU $\mathcal{N}$ est **contrôlable**

ssi $\forall \omega \in \Omega$, $\mathcal{N}_\omega$ est 2-cohérent.

ssi $\forall \omega \in \Omega$, $\forall i,j=1,..,n,\ i \neq j$, $\exists x_i \in D_i$, $\exists x_j \in D_j$ / $(x_j - x_i) \in C_{ij}$.

Rappelons-nous que dans une instance $\mathcal{N}_\omega$, chaque contrainte G_k se réduit à $f_k - d_k = \omega_k$. L'objectif est de se ramener à une condition où seules les variables de Déclenchement entrent en jeu. Que fait-on alors des variables d'Attente ? Dans la formulation ci-dessus, i et j peuvent être soit des variables de Déclenchement soit des variables d'Attente, ce qui donne quatre cas de figures distincts:

- si $i \equiv d_k$ et $j \equiv d_{k'}$, alors on obtient naturellement une contrainte entre deux variables de Déclenchement;
- si $i \equiv d_k$ et $j \equiv f_{k'}$, alors on peut se ramener à une contrainte entre deux variables de Déclenchement d_k et $d_{k'}$ du fait que: $f_{k'} - d_k = d_{k'} + \omega_{k'} - d_k$;
- $i \equiv f_k$ et $j \equiv d_{k'}$: c'est le cas symétrique $d_{k'} - f_k = d_{k'} - (d_k + \omega_k)$;
- $i \equiv f_k$ et $j \equiv f_{k'}$: c'est l'addition des deux précédents: $f_{k'} - f_k = d_{k'} + \omega_{k'} - (d_k + \omega_k)$.

En résumé, on peut se ramener à une formulation équivalente portant sur les seules variables de Déclenchement (en confondant l'instant d_k et sa date):

Définition 4.10 : *Local-Control*$(d_k, d_{k'}, \omega_k, \omega_{k'})$ est la donnée des quatre contraintes:
- $(d_{k'} - d_k) \in L_{dkdk'}$
- $(d_{k'} + \omega_{k'} - d_k) \in L_{dkfk'}$
- $(d_{k'} - (d_k + \omega_k)) \in L_{fkdk'}$
- $(d_{k'} + \omega_{k'} - (d_k + \omega_k)) \in L_{fkfk'}$

Propriété 3.2 : Un STPU $\mathcal{N}$ est **contrôlable**
ssi $\forall \omega \in \Omega$, $\forall (G_k, G_{k'}) \in \Gamma^2$, $\exists\ (d_k, d_{k'})$ / *Local-Control*$(d_k, d_{k'}, \omega_k, \omega_{k'})$,

Rappelons également que dans le cadre de nos hypothèses, les contraintes $L_{dkdk'}$, $L_{dkfk'}$, $L_{fkdk'}$ et $L_{fkfk'}$ sont en entrée de simples contraintes symboliques traduites dans le formalisme numérique. Pour visualiser ceci, la figure ci-dessous montre que deux CTGs sont en fait deux couples d'instants définissant entre eux quatre contraintes symboliques possibles. Si l'on regarde de plus près ce qui précède, on peut voir que l'on a caractérisé ainsi des relations **entre CTGs**, qui peuvent être ramenées à des relations entre leurs instants de début, qui sont les variables de notre problème de décision.

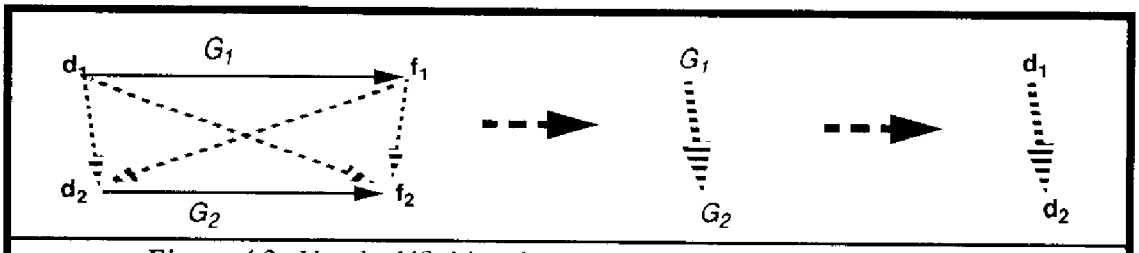


Figure 4.3. Vers la définition de meta-contraintes et d'un graphe dual

Nous avons là toutes les briques de base pour définir notre problème dual, que nous appellerons Graphe de Décision. Les variables du Graphe de Décision seront les variables de Déclenchement, et les contraintes se définiront à partir des quadruplets de contraintes ci-dessus identifiés. Nous les nommerons **meta-contraintes**, du fait qu'elles se définissent au départ sous la forme de relations entre contraintes, entre CTGs pour être plus précis. Nous pouvons d'ailleurs signaler à titre de simple remarque que le Graphe de Décision aurait pu être défini de manière équivalente en donnant comme ensemble de variables l'ensemble des CTGs.

4. Le Graphe de Décision

4.1. Définition Formelle du Graphe de Décision

Partant des définitions et propriétés énoncées dans la section précédente, nous pouvons donner la définition formelle du Graphe de Décision.

Définition 4.11 : Soit un STPU $\mathcal{N}$ obéissant aux hypothèses restrictives données. Alors son **Graphe de Décision** associé est un triplet $\mathcal{D} = \{V_D, D_D, \Lambda_D\}$, où
- $V_D = \{x_1, ..., x_d\}$ est l'ensemble des dates des variables de Déclenchement, c'est-à-dire les instants de début de contraintes de Γ.
- $D_D = D_1 \times ... \times D_d$ est le produit cartésien de leurs domaines,
- $\Lambda_D = \{MN_{kk'} / (x_k, x_{k'}) \in V_D^2\}$ est l'ensemble des **meta-contraintes numériques** entre variables de Déclenchement, qui restent à définir.

Il nous reste à montrer que le CSP binaire ainsi défini est équivalent au STPU initial, et qu'il permet de résoudre le problème posé initialement. Tout cela passe bien sûr par la caractérisation préalable des meta-contraintes. Ce terme de meta-contrainte désignant initialement les relations entre CTGs G_k du STPU va par suite s'appliquer aux contraintes entre variables du Graphe de Décision. Pour éviter toute ambiguïté, nous emploierons parfois le terme de "contrainte élémentaire" pour désigner une contrainte entre instants du STPU initial.

Ce sont les meta-contraintes numériques qui nous intéressent, $\mathcal{D}$ étant un graphe numérique. Néanmoins, pour pouvoir les caractériser, il nous a paru avantageux pour la clarté du propos de commencer par étudier les relations symboliques entre deux CTGs, que nous nommerons naturellement **meta-contraintes symboliques**. Notons qu'il s'agit simplement d'un formalisme intermédiaire, qui ne figurera pas explicitement dans notre méthode.

Les meta-contraintes seront a priori des relations entre intervalles "à la Allen", mais tout n'est pas si simple. D'abord, les STPUs définissent un paradigme particulier dans lequel l'ensemble des meta-contraintes symboliques va sensiblement différer de l'ensemble connu des relations de l'algèbre d'intervalle réduite PIA. Ensuite, caractériser la relation numérique entre deux intervalles constitue un exercice nouveau, et a priori peu aisé. En fait, la spécificité des STPUs va ici, au contraire, nous faciliter la tâche ...

4.2. Les Meta-Contraintes Symboliques

Nos hypothèses restrictives nous permettent de conclure que deux CTGs G_1 et G_2 sont reliées par au plus quatre contraintes symboliques entre instants. On peut donc dire qu'une relation symbolique entre G_1 et G_2 n'est autre que la donnée des quatre contraintes symboliques élémentaires séparant les instants de début et de fin des deux CTGs. Nous écrirons donc $MS_{1,2}=\{(d_1,d_2), (d_1,f_2), (f_1,d_2), (f_1,f_2)\}$, pour désigner la meta-contrainte symbolique entre G_1 et G_2.

A priori, nous devrions retrouver les relations de l'algèbre d'intervalle réduite [Vilain89]. En fait, certaines de ces relations ne sont pas admissibles dans notre paradigme contingent. Nous devons par exemple exclure les relations où $f_1=f_2$, puisqu'il est impossible d'être certain de l'occurrence simultanée, en cours d'exécution, de deux événements aléatoires. Formellement, cela signifie que nous avons deux observations sur une même contrainte (la date de l'événement) provenant de deux variables aléatoires différentes, ce qui est par nature incontrôlable:

Propriété 3.3 : si $\exists (G_1,G_2) \in \Gamma^2 / f_1=f_2$, alors $\mathcal{N}$est incontrôlable.

La construction exhaustive qui suit permet de démontrer du même coup la complétude de l'ensemble des meta-contraintes symboliques ainsi déterminé. Le lecteur peu désireux d'entrer dans les détails pourra passer directement à la figure 4.5 qui résume cet ensemble de manière graphique.

♣♣♣

Rappelons que notre réseau symbolique $\mathcal{S}$ se base sur une algèbre de points réduite aux relations $\{=, \leq, \geq, ?\}$ (cf chapitre 2, § 2.3.1). Nous avons donc quatre relations possibles pour tout couple de points. Une meta-contrainte symbolique $MS_{1,2}$ se caractérise par la donnée de quatre couples de points (voir plus haut). Cela nous donne a priori 4^4=256 meta-contraintes, qui peuvent être obtenues en développant un arbre de recherche à 4 niveaux (les quatre couples d'instants) et un coefficient de branchement constant égal à 4 (les quatre contraintes élémentai-

res possibles pour un couple donné). Parmi les feuilles de cet arbre, seules certaines constituent des meta-contraintes symboliques admissibles. Certaines combinaisons étant incohérentes, et d'autres étant redondantes elles seront considérées comme non **admissibles**. Ces différentes notions sont regroupées dans la définition suivante. Nous pouvons de fait nous attendre à obtenir un nombre de feuilles admissibles nettement inférieur à 256. De plus, nous définissons également ci-dessous la notion de combinaison saturée, qui va nous permettre de réduire sensiblement l'arbre de recherche.

Définition 4.12 :

- Une meta-contrainte symbolique $MS_{1,2}$ est **redondante** ssi elle est composée d'au moins deux contraintes élémentaires telles que l'une induit nécessairement l'autre.

- Une meta-contrainte symbolique $MS_{1,2}$ est **incohérente** ssi il existe un cycle incluant l'un des deux couples de points (d_1, f_1) ou (d_2, f_2). En effet, cela obligerait à agréger ces instants, donc en particulier les deux extrémités de la CTG, ce qui revient à restreindre le domaine de cette CTG.

- Une meta-contrainte symbolique *partielle* entre G_1 et G_2 est la donnée d'un sous-ensemble des quatre contraintes élémentaires qui composent $MS_{1,2}$ (c'est-à-dire qu'il s'agit d'un noeud de l'arbre de recherche).

- Une meta-contrainte symbolique **saturée** est une meta-contrainte symbolique partielle telle que les contraintes élémentaires qui la composent induisent celles qui restent à déterminer (c'est-à-dire qu'il n' y a qu'une seule feuille admissible à partir de ce noeud).

Nous allons très vite illustrer ces définitions au travers d'exemples. Nous noterons par la suite chaque meta-contrainte symbolique sous la forme d'un quadruplet de contraintes élémentaires. Nous n'y ferons cependant figurer pour simplifier l'écriture ni les contraintes élémentaires redondantes, ni l'absence de contrainte (relation **?**) entre deux instants. Considérons tout d'abord les relations $MS_{1,2}$ contenant des contraintes élémentaires d'**égalité** entre instants. Nous allons passer en revue les quatre couples de points l'un après l'autre:

1. $f_1 = f_2$ est exclu par définition, car incontrôlable (voir ci-dessus).

2. $f_1 = d_2$: il s'agit d'une relation saturée car $(f_1 = d_2) \Rightarrow (d_1 \leq d_2) \wedge (f_1 \leq f_2) \wedge (d_1 \leq f_2)$. La seule meta-contrainte symbolique admissible à partir de là est donc $\{(f_1 = d_2)\}$.

3. $d_1 = f_2$: il s'agit de la relation symétrique de la précédente, ce qui conduit également à une seule meta-contrainte symbolique admissible:
   $\{(d_1 = f_2)\}$.

4. $d_1 = d_2$ fournit l'implication suivante: $(d_1 = d_2) \Rightarrow (d_1 \leq f_2) \wedge (f_1 \geq d_2)$. Il ne reste donc qu'à développer les trois feuilles possibles en explicitant la contrainte entre f_1 et f_2 (pour laquelle nous rappelons que l'égalité est exclue), ce qui donne trois meta-contraintes symboliques admissibles:
   $\{(d_1 = d_2)\}$ (où $(f_1 \; ? \; f_2)$ est sous-entendue),
   $\{(d_1 = d_2), (f_1 \leq f_2)\}$.
   $\{(d_1 = d_2), (f_1 \geq f_2)\}$.

Cette première étude rapide nous fournit donc déjà cinq meta-contraintes symboliques admissibles. Elle réduit surtout sensiblement l'espace de recherche puisque nous n'avons plus qu'à explorer un arbre dont le coefficient de branchement est descendu à trois, puisqu'il n'est plus nécessaire de considérer les cas des égalités. Ce qui laisse a priori 3^4=81 relations possibles. La figure de la page suivante fournit l'arbre de recherche développé, où une relation complète peut être retrouvée en parcourant un chemin de la racine jusqu'à une feuille. Au niveau des feuilles nous avons différencié les relations obtenues selon qu'elles sont **redondantes**, **incohérentes** ou **saturées**. Un noeud appartenant à l'un de ces trois types n'est pas développé plus avant, pour les raisons invoquées plus haut.

La partie de l'arbre en pointillé est un exemple de développement qui pourrait être effectué à partir d'une relation saturée. Outre le fait qu'elle montre clairement que l'on aboutit dans ce cas-là à une seule feuille admissible, elle permet de visualiser des cas de meta-contraintes symboliques incohérentes, qui sinon n'apparaîtraient pas dans la recherche. Toutes les relations incohérentes se situent en effet sur des branches non développées. Prenons juste deux exemples dans l'arbre pour illustrer les concepts présentés:

- Si l'on choisit $(f_1 \geq d_2)$, puis $(d_1 \geq f_2)$, on tombe sur une meta-contrainte symbolique redondante car $(d_1 \geq f_2) \Rightarrow (f_1 \geq d_2)$. En fait, $\{(d_1 \geq f_2)\}$ est une relation saturée qui figure plus haut dans l'arbre.

- Si l'on choisit $(f_1 \; ? \; d_2)$, puis $(d_1 \geq f_2)$, puis $(f_1 \leq f_2)$, on tombe sur une meta-contrainte symbolique incohérente car $(f_1 \leq f_2) \wedge (d_1 \geq f_2) \Rightarrow (d_1 \geq f_1)$, ce qui, combiné avec la contrainte implicite $(d_1 \leq f_1)$, obligerait à agréger d_1 et f_1.

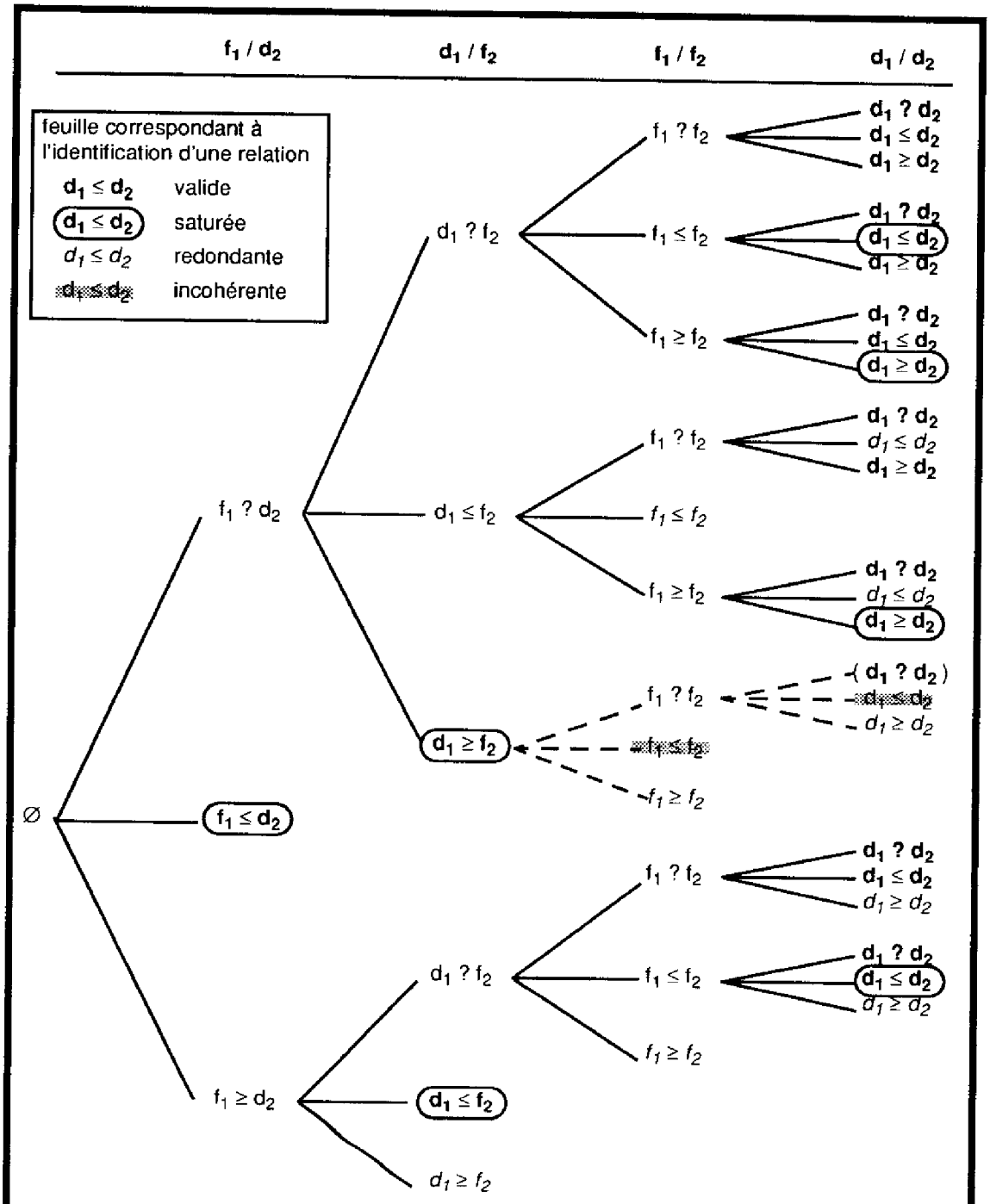


Figure 4.4. Arbre de construction des meta-contraintes symboliques.

♣♣♣

Nous avons donc finalement obtenu 25 meta-contraintes symboliques, qui se décomposent en fait en 14 relations directes et 11 relations inverses. La figure ci-dessous se contente de donner les relations directes, en fournissant en plus de la notation introduite au début du paragraphe, une dénomination par numérotation des meta-contraintes symboliques ainsi déterminées. Nous utiliserons préférentiellement cette notation plus compacte par la suite. Les relations inverses se définissent naturellement de la manière suivante: $MS_{1,2}=Ri_p$ ssi $MS_{2,1}=R_p$. On aura par exemple $Ri_8=\{(d_1 \geq d_2), (d_1 \leq f_2)\}$. Remarquons au passage que $Ri_0=R_0$, $Ri_1=R_1$ et $Ri_{12}=R_{12}$.

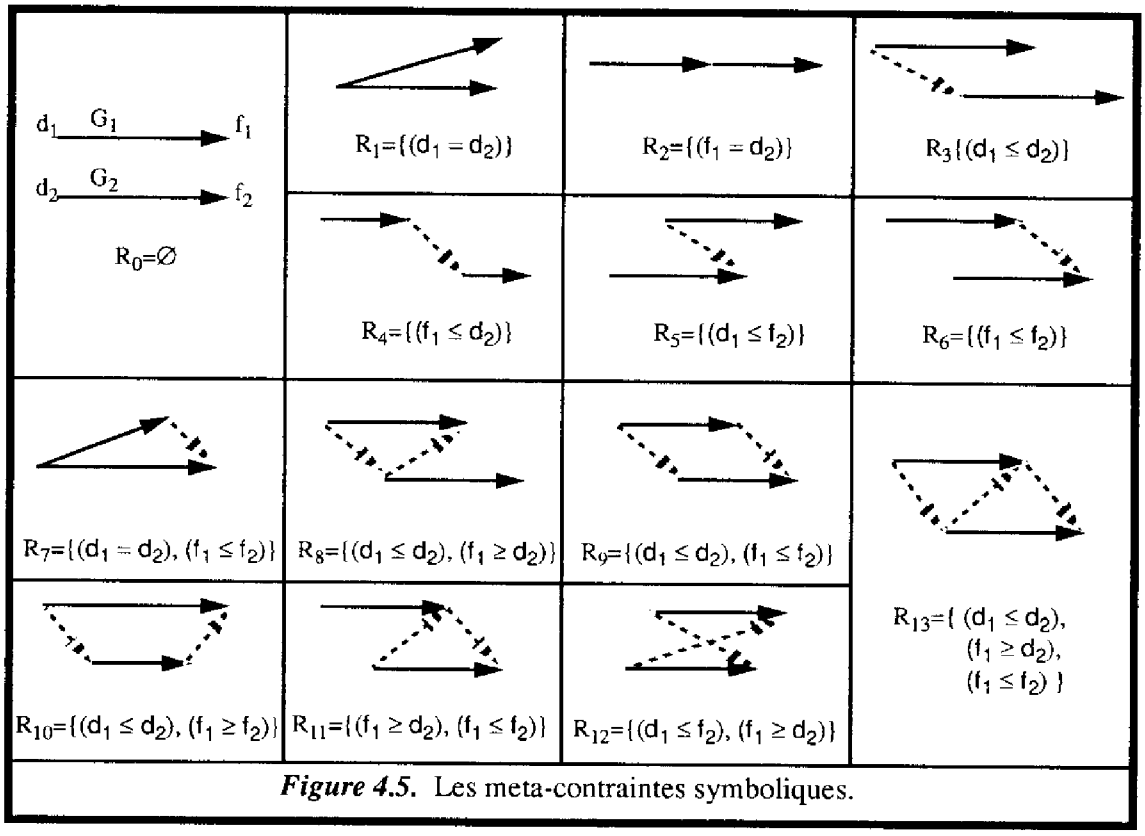


Figure 4.5. Les meta-contraintes symboliques.

Partant de là, nous pourrions démontrer que nous obtenons une structure d'algèbre et donner les tables d'intersection et de composition. Cela ne sera pas fait, pour la simple et bonne raison déjà évoquée que les meta-contraintes symboliques ne sont qu'un formalisme intermédiaire utilisé pour introduire les meta-contraintes numériques. Le Graphe de Décision, qui ne gère explicitement que les meta-contraintes numériques, n'aura pas l'utilité d'effectuer des opérations de type propagation des meta-contraintes symboliques.

4.3. Les Meta-Contraintes Numériques

Il va maintenant s'agir de définir pour chaque couple de CTGs (G_1, G_2) la meta-contrainte numérique $MN_{1,2}$ qui lie d_1 à d_2, représentant intuitivement la marge de temps disponible entre les déclenchements de ces deux CTGs, en respectant la meta-contrainte symbolique $MS_{1,2}$. La détermination de cette meta-contrainte numérique procède de la redéfinition de notre problème au paragraphe 3.3. Nous avons défini la condition *Local-Control*(d_1, d_2, ω_1, ω_2), en montrant que prouver la cohérence d'un réseau $\mathcal{N}_\omega$ revenait à vérifier cette condition pour chaque couple de CTGs (G_1, G_2). Ceci nous permet tout d'abord de proposer une définition de contrôlabilité locale:

Définition 4.13 : Le couple $(G_1, G_2) \in \Gamma^2$ est **localement contrôlable** dans $\mathcal{N}$ ssi $\forall \omega_1 \in G_1$ et $\forall \omega_2 \in G_2$, $\exists\ (d_1, d_2)$ / *Local-Control*(d_1, d_2, ω_1, ω_2).

Comme nous l'avons déjà argumenté, les contraintes en entrée $\llcorner_{d_1d_2}$, $\llcorner_{d_1f_2}$, $\llcorner_{f_1d_2}$ et $\llcorner_{f_1f_2}$ (présentes implicitement dans la condition *Local-Control*(d_1, d_2, ω_1, ω_2), cf § 3.3) expriment soit une stricte égalité, soit une simple précédence, soit l'absence de contraintes. Comme nous l'avons vu au niveau des meta-contraintes symboliques, cela nous donne donc simplement quatre possibilités pour chaque couple de points, qui s'expriment au niveau numérique par les intervalles $]-\infty, +\infty[$ (présent par défaut), $[0, +\infty[$, $]-\infty, 0]$ ou $[0, 0]=\{0\}$. Nous allons chercher dans ce paragraphe à transformer cette condition de contrôlabilité locale, par l'intermédiaire d'un simple travail d'écriture, en **une contrainte binaire entre d_1 et d_2**.

4.3.1. Illustration par l'Exemple

Dans notre exemple, si G_1 est la tâche "Echange-Télé" et G_2 la tâche "Nettoyage", alors nous avons une meta-contrainte symbolique de type "$G_1\ R_{10}\ G_2$". Cherchons à établir la condition de contrôlabilité locale dans ce cas particulier. C'est-à-dire qu'il nous faut traduire dans le formalisme numérique les deux précédences ($d_1 \leq d_2$) et ($f_2 \leq f_1$). Cela nous donne

- $d_2 - d_1 \in [0, +\infty[$
- $d_2 + \omega_2 - (d_1 + \omega_1) \in]-\infty, 0]$

Ceci doit être vérifié quelles que soient les durées observées ω_1 et ω_2, ce qui nous donne:

- $d_1 \leq d_2$
- $\forall \omega_1 \in [inf_1, sup_1]$, $\forall \omega_2 \in [inf_2, sup_2]$, $d_2 + \omega_2 \leq d_1 + \omega_1$

Cette relation se ramène à une simple contrainte entre les dates de d_1 et d_2:

- $\forall \omega_1 \in [inf_1, sup_1]$ et $\forall \omega_2 \in [inf_2, sup_2]$, $0 \leq d_2 - d_1 \leq \omega_1 - \omega_2$

Nous nous sommes donc ramenés à une double inéquation dont les inconnues sont les dates de déclenchement des CTGs. Pour que cette condition soit vérifiée quelles que soient les durées observées, il faut et il suffit que

$- 0 \leq d_2 - d_1 \leq inf_1 - sup_2$.

Nous voici donc en présence d'une contrainte entre les instants d_1 et d_2 du type des contraintes des STPs [Dechter91]. Celle-ci définit la meta-contrainte numérique entre d_1 et d_2, pour laquelle nous utiliserons naturellement les mêmes notations que celles utilisées jusqu'ici dans le cadre des STPs. Cette contrainte et la condition de contrôlabilité locale s'expriment alors de la manière suivante:

- La meta-contrainte numérique entre d_1 et d_2 est $MN_{1,2} = [0, inf_1\text{-}sup_2[$
- La relation entre G_1 et G_2 est localement contrôlable ssi $0 \leq inf_1\text{-}sup_2$

Cette condition de contrôlabilité se ramène à $sup_2 \leq inf_1$, c'est-à-dire qu'il est nécessaire que $G_2 \leq G_1$ (en termes de variables aléatoires), ce qui correspond bien à l'intuition pour une relation de type "*during*".

4.3.2. Application à l'ensemble des meta-contraintes symboliques

Nous allons ci-dessous reprendre succinctement le même raisonnement pour chacune des 13 autres meta-contraintes symboliques directes décrites au paragraphe précédent. Pour les relations inverses, il suffira d'appliquer la contrainte unaire numérique $MN_{2,1} = -MN_{1,2}$.

- $MS_{1,2}=R_0=\emptyset \Rightarrow MN_{1,2}=]-\infty, +\infty[$: toujours contrôlable.

- $MS_{1,2}=R_1=\{(d_1 = d_2)\} \Rightarrow MN_{1,2}=\{0\}$: toujours contrôlable.

- $MS_{1,2}=R_2=\{(f_1 = d_2)\}$: ce cas de figure pose un problème particulier, puisqu'une variable ne peut être à la fois attendue et déclenchée de par la définition de notre partition au paragraphe 2.1. Nous le traiterons à part à la fin de ce paragraphe (cf § 4.5).

- $MS_{1,2}=R_3=\{(d_1 \leq d_2)\} \Rightarrow MN_{1,2}=[0, +\infty[$: toujours contrôlable.

- $MS_{1,2}=R_4=\{(f_1 \leq d_2)\}$ s'écrit
  $\forall \omega_1 \in G_1, d_1 + \omega_1 \leq d_2$
  $\Leftrightarrow MN_{1,2}=[sup_1, +\infty[$: toujours contrôlable.

- $MS_{1,2}=R_5=\{(d_1 \leq f_2)\}$ s'écrit
  $\forall \omega_1 \in G_1, d_1 \leq d_2 + \omega_2$
  $\Leftrightarrow MN_{1,2}=[-inf_2, +\infty[$: toujours contrôlable.

- $MS_{1,2}=R_6=\{(f_1 \leq f_2)\}$ s'écrit
  $\forall \omega_1 \in G_1, d_1 + \omega_1 \leq d_2 + \omega_2$
  $\Leftrightarrow MN_{1,2}=[sup_1\text{-}inf_2, +\infty[$: toujours contrôlable.

- $MS_{1,2}=R_7=\{(d_1 = d_2), (f_1 \leq f_2)\} \equiv R_1 \cap R_6$, ce qui nous donne l'intersection des meta-contraintes numériques correspondantes:
  $MN_{1,2}= \{0\} \cap [sup_1\text{-}inf_2, +\infty[$
  $\Leftrightarrow$ si $sup_1 \geq inf_2$, alors $MN_{1,2}= \{0\}$, sinon $MN_{1,2}$ est incontrôlable.

- $MS_{1,2}=R_8=\{(d_1 \leq d_2), (f_1 \geq d_2)\} \equiv R_3 \cap Ri_5$, ce qui nous donne
  $MN_{1,2}= [0, +\infty[\cap]-\infty, inf_1]$
  $\Leftrightarrow MN_{1,2}= [0, inf_1]$: toujours contrôlable car G_1 CTG forte $\Rightarrow inf_1 \geq 0$.

- $MS_{1,2}=R_9=\{(d_1 \leq d_2), (f_1 \leq f_2)\} \equiv R_3 \cap R_6$, ce qui nous donne
  $MN_{1,2}= [0, +\infty[\cap [sup_1\text{-}inf_2, +\infty[$
  $\Leftrightarrow MN_{1,2}= [\max(0, sup_1\text{-}inf_2), +\infty[$: toujours contrôlable.

- $MS_{1,2}=R_{10}=\{(d_1 \leq d_2), (f_1 \geq f_2)\} \equiv R_3 \cap Ri_6$, ce qui nous donne
  $MN_{1,2}= [0, +\infty[\cap]-\infty, inf_1\text{-}sup_2]$
  $\Leftrightarrow$ si $sup_2 \leq inf_1$, alors $MN_{1,2}= [0, inf_1\text{-}sup_2]$, sinon $MN_{1,2}$ est incontrôlable.

- $MS_{1,2}=R_{11}=\{(d_1 \leq f_2), (f_1 \geq f_2)\} \equiv R_5 \cap Ri_6$, ce qui nous donne
  $MN_{1,2}= [-inf_2, +\infty[\cap]-\infty, inf_1\text{-}sup_2]$
  $\Leftrightarrow$ si $sup_2 - inf_2 \leq inf_1$, alors $MN_{1,2}= [-inf_2, inf_1\text{-}sup_2]$, sinon $MN_{1,2}$ est incontrôlable.

- $MS_{1,2}=R_{12}=\{(d_1 \leq f_2), (f_1 \geq d_2)\} \equiv R_5 \cap Ri_5$, ce qui nous donne
  $MN_{1,2}= [-inf_2, +\infty[\cap]-\infty, inf_1]$
  $\Leftrightarrow$ si $inf_1 + inf_2 \geq 0$, alors $MN_{1,2}= [-inf_2, inf_1]$, sinon $MN_{1,2}$ est incontrôlable.

- $MS_{1,2}=R_{13}=\{(d_1 \leq d_2), (f_1 \geq d_2), (f_1 \leq f_2)\} \equiv R_8 \cap R_6$, ce qui nous donne
  $MN_{1,2}= [0, inf_1] \cap [sup_1\text{-}inf_2, +\infty[$
  $\Leftrightarrow$ si $sup_1 - inf_1 \leq inf_2$, alors $MN_{1,2}= [\max(0, sup_1\text{-}inf_2), inf_1]$, sinon $MN_{1,2}$ est incontrôlable.

Dans la quasi-totalité des cas, nous pouvons exprimer la relation entre G_1 et G_2 par l'intermédiaire d'une contrainte entre d_1 et d_2. Nous avons de plus caractérisé la contrôlabilité de chaque meta-contrainte numérique particulière. Nous constatons notamment que les relations R_7, R_{10}, R_{11}, R_{12} et R_{13} sont susceptibles d'être incontrôlables selon les valeurs présentes dans les contraintes G_1 et G_2. Il reste néanmoins le cas problématique de la meta-contrainte symbolique $R_2=\{(f_1 = d_2)\}$.

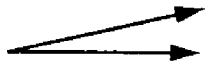
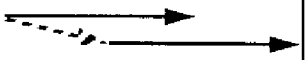
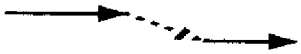
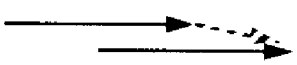
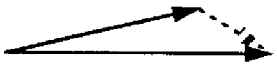
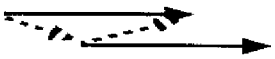
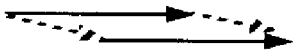
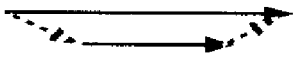
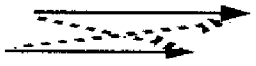
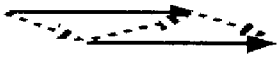
Nom	Représentation de la meta-contrainte symbolique	Relations entre d_1 et d_2	Meta-contrainte numérique $MN_{1,2} \equiv \mathbf{L_{d_1 d_2}}$
R_0	d_1 $D_1=[inf_1,sup_1]$ f_1 d_2 $D_2=[inf_2,sup_2]$ f_2	$\varnothing$	$]-\infty, +\infty[$
R_1		$d_1 = d_2$	$\{0\}$
R_2		$d_1 + D_1 = d_2$	??? ... cf § 4.5 ...
R_3		$d_1 \leq d_2$	$[0, +\infty[$
R_4		$d_1 + D_1 \leq d_2$	$[\sup_1, +\infty[$
R_5		$d_1 \leq d_2 + D_2$	$[-inf_2, +\infty[$
R_6		$d_1 + D_1 \leq d_2 + D_2$	$[\sup_1-inf_2, +\infty[$
R_7		$d_1 = d_2$ $d_1 + D_1 \leq d_2 + D_2$	$[\max(\sup_1-inf_2, 0), 0]$ **INCLB si $\sup_1 > inf_2$**
R_8		$d_1 \leq d_2$ $d_2 \leq d_1 + D_1$	$[0, inf_1]$ **INCLB si $inf_1 < 0$**
R_9		$b_1 + D_1 \leq d_2 + D_2$ $d_1 \leq d_2$	$[\max(\sup_1-inf_2, 0), +\infty[$
R_{10}		$d_1 \leq d_2$ $d_2 + D_2 \leq d_1 + D_1$	$[0, inf_1-sup_2]$ **INCLB si $inf_1 < sup_2$**
R_{11}		$d_1 \leq d_2 + D_2$ $d_2 + D_2 \leq d_1 + D_1$	$[-inf_2, inf_1-sup_2]$ **INCLB si $inf_1 < sup_2 - inf_2$**
R_{12}		$d_1 \leq d_2 + D_2$ $d_2 \leq d_1 + D_1$	$[-inf_2, inf_1]$
R_{13}		$d_1 \leq d_2$ $d_2 \leq d_1 + D_1$ $d_1 + D_1 \leq d_2 + D_2$	$[\max(\sup_1-inf_2, 0), inf_1]$ **INCLB si $\sup_1 - inf_1 < inf_2$**

Tableau 4.1. Meta-contraintes et contrôlabilité locale.

En fait, dans ce cas-là,

- La relation est nécessairement contrôlable, puisqu'il suffit de déclencher G_2 immédiatement après G_1, ce qui peut toujours être fait quelles que soient les valeurs $\omega_1 \in G_1$ et $\omega_2 \in G_2$,

- Il n'y a qu'une seule variable de Déclenchement d_1, d_2 étant confondue avec f_1, et étant donc une variable d'Attente par définition.

Ceci contredit donc l'hypothèse préliminaire selon laquelle une CTG est définie entre une variable de Déclenchement et une variable d'Attente. Pour l'instant, nous choisissons d'étendre nos hypothèses restrictives en excluant ce cas de figure. Il s'agit là d'une "solution de facilité". Nous invoquerons comme excuse le fait que nous ne souhaitons pas alourdir davantage ce paragraphe. C'est pourquoi nous étudierons plus en détail ce cas particulier en fin de section (cf § 4.5).

Pour le moment, nous obtenons donc un principe général applicable aux cas de figure étudiés ci-dessus, exception faite de R_2. Le tableau 4.1 résume l'ensemble des meta-contraintes numériques ainsi déterminées à partir des meta-contraintes symboliques. Y figurent également les propriétés de **contrôlabilité locale** de ces meta-contraintes (condition d'incontrôlabilité notée INCLB).

4.4. Propriétés Générales et Vérification de Contrôlabilité

Nous avons donc défini les meta-contraintes numériques de Λ_D en fonction des contraintes d'entrée dans le STPU initial. De plus, ces meta-contraintes numériques sont par définition des contraintes libres entre instants. En conclusion, le Graphe de Décision $\mathcal{D}$ répond aux caractéristiques d'un STP. On retrouve donc les lois de composition et d'intersection, et la propriété de distributivité de l'une sur l'autre, assurant du même coup la décomposabilité du graphe et l'obtention du réseau minimal par un algorithme de cohérence de chemin. Formellement,

- composition: $MN_{1,3} \leftarrow MN_{1,2} \oplus MN_{2,3} = [inf_{d1d2}+inf_{d2d3}, sup_{d1d2}+sup_{d2d3}]$.
- intersection: $MN_{1,2} \leftarrow MN_{1,2} \cap MN_{1,2}' = [\max(inf_{d1d2}, inf_{d1d2}'), \min(sup_{d1d2}, sup_{d1d2}')]$.

De même, la cohérence de $\mathcal{D}$ se définit alors de la manière traditionnelle suivante:

Le graphe $\mathcal{D}$ est **cohérent** ssi $\forall (d_k, d_{k'}) \in V_D^2 / MN_{kk'}=[inf_{kk'}, sup_{kk'}], inf_{kk'} \leq sup_{kk'}$.

La propriété de contrôlabilité locale dans $\mathcal{N}$ est donc équivalente à la cohérence de la meta-contrainte numérique correspondante dans $\mathcal{D}$:

Propriété 3.4 : $\forall (G_1, G_2) \in \Gamma$, la relation entre G_1 et G_2 est **localement contrôlable** ssi $MN_{1,2}$ est cohérente dans $\mathcal{D}$.

Ce qui dans $\mathcal{N}$ peut également s'écrire, en considérant implicitement d_k comme étant la variable de V_N correspondant à l'instant de début de la CTG G_k:

- $\forall k,k' / (G_k, G_{k'}) \in \Gamma^2, \forall \omega_k \in G_k, \forall \omega_{k'} \in G_{k'}, \exists d_k \in D_k, \exists d_{k'} \in D_{k'} / Local\text{-}Control(d_k, d_{k'}, \omega_k, \omega_{k'})$

C'est-à-dire que nous avons établi une propriété de 2-cohérence dans $\mathcal{D}$. $\mathcal{D}$ définissant un STP, la 2-cohérence implique la cohérence globale. Cette cohérence globale dans $\mathcal{D}$ peut alors s'écrire de manière équivalente par rapport aux variables et contraintes du STPU de départ:

- $\exists \delta_d = \{d_1 \in D_{d1}, ..., d_g \in D_{dg}\}$ tel que
[dans $\mathcal{D}$] $\forall k,k' = 1,...,g, (d_{k'} - d_k) \in MN_{kk'}$.
$\Leftrightarrow$ [dans $\mathcal{N}$] $\forall \omega \in \Omega, \forall k,k' = 1,...,g, Local\text{-}Control(d_k, d_{k'}, \omega_k, \omega_{k'})$

En d'autres termes, l'équivalence entre cohérence locale et cohérence globale dans $\mathcal{D}$ se répercute dans $\mathcal{N}$ en une équivalence entre ce que nous avons défini au paragraphe 4.3 comme étant une condition de contrôlabilité locale, et la propriété de contrôlabilité globale de $\mathcal{N}$. En effet, la formulation ci-dessus nous permet de retrouver la définition de la contrôlabilité de $\mathcal{N}$ établie au § 3.3. D'où l'on peut déduire la propriété fondamentale suivante:

Propriété 3.5 : **Le graphe $\mathcal{D}$ est cohérent ssi le graphe $\mathcal{N}$ est contrôlable.**

De ce fait, l'application d'un algorithme tel que PC-2 dans $\mathcal{D}$ va vérifier la cohérence de ce graphe, et rendre compte d'un réseau minimal, au sens classique du terme.

4.4.1. Illustration par l'Exemple

Nous conservons la description du § 1.1, à un détail près, qui ne sera pas sans conséquences: nous supposons désormais que la voiture de Frank ne peut contenir deux postes de télévision en même temps. Ce qui signifie qu'Albert ne peut déposer l'ancien récepteur dans la voiture, puis prendre le nouveau. Il devrait d'abord poser le récepteur par terre, sortir le nouveau et le poser également par terre, etc. Bref, toute une gymnastique qui n'est pas sans rappeler les multiples problèmes que l'on peut rencontrer dans le monde des cubes, domaine d'application de référence, s'il en est, pour la planification. Dans notre cas, heureusement, Frank et Albert peuvent collaborer pour parvenir à une solution plus simple et plus rapide. Ils décident par exemple que c'est Frank qui amènera le nouveau récepteur (tâche "Apporter-Nouvelle-Télé"), pendant qu'Albert effectuera le trajet inverse avec l'ancien appareil (tâche "Emporter-Ancienne-Télé"). Frank devra avoir commencer cette tâche avant qu'Albert ne vienne déposer l'ancien récepteur.

On a là une illustration d'un problème d'utilisation d'une ressource non partageable [Laborie94] par deux tâches, l'une nécessitant cette ressource en fin d'exécution, alors même

que la seconde tâche est censée libérer la même ressource au début de son exécution. En conclusion, les deux tâches "Emporter-Ancienne-Télé" (contrainte G_1) et "Apporter-Nouvelle-Télé" (contrainte G_2) viennent se substituer à la tâche "Echanger-Télé", et conduisent, avec des contraintes similaires par rapport à la tâche "Nettoyage" (contrainte G_3), au graphe suivant. Deux exemples numériques sont donnés. Le lecteur aura tout de suite compris que nous allons nous employer à démontrer que l'un est contrôlable, et l'autre pas.

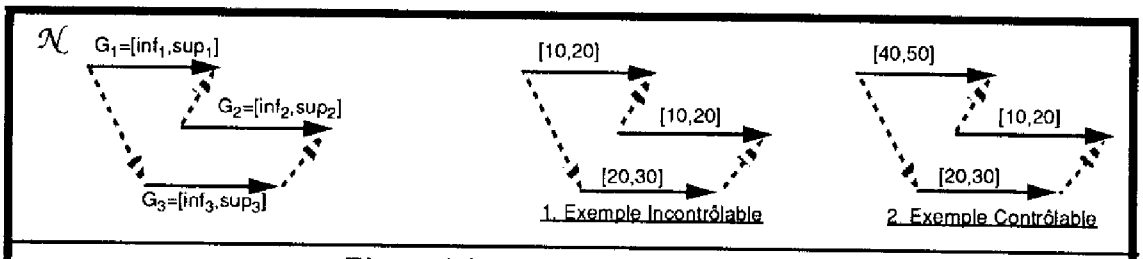


Figure 4.6. Extension de l'exemple.

Nous pouvons analyser les précédences élémentaires liant chaque couple de CTGs (G_1, G_2), (G_1, G_3) et (G_2, G_3), à partir desquelles peuvent être déterminées les meta-contraintes symboliques par simple recours au tableau 4.1. Nous obtenons $G_1\ R_3\ G_3$, $G_2\ R_5\ G_1$, et $G_3\ R_6\ G_2$. A partir de là, le tableau 4.1 toujours nous donne les meta-contraintes numériques, le tout étant représenté dans une figure unique illustrant le graphe $\mathcal{D}$ ainsi que les meta-contraintes symboliques. Nous représentons le Graphe de Décision avant et après une opération de propagation par l'algorithme PC-2.

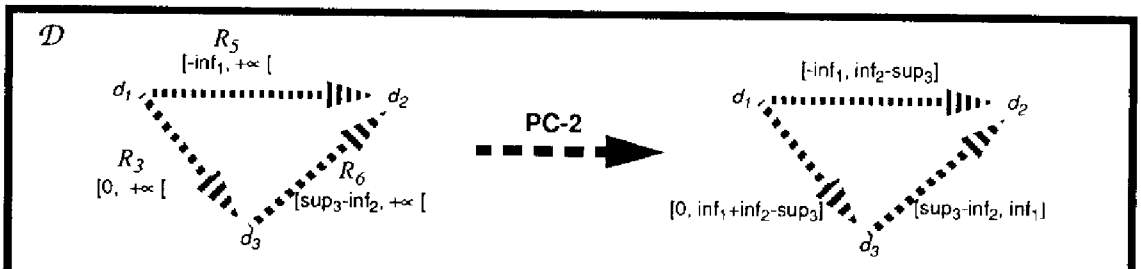


Figure 4.7. Le Graphe de Décision correspondant à l'exemple.

Il ne nous reste plus qu'à transcrire dans ce formalisme les deux exemples numériques de la figure 4.6. Les trois meta-contraintes numériques propagées expriment toutes la même chose, à savoir que ce triplet de CTGs est contrôlable ssi $sup_3 \leq inf_1 + inf_2$. Dans le premier exemple, cette condition se traduit par $30 \leq 20$. Les meta-contraintes résultantes sont donc incohérentes, ce qui signifie que le réseau initial est incontrôlable, et par suite que le plan produit est susceptible d'échouer en cours d'exécution. Dans le deuxième exemple au contraire, représenté ci-dessous, le réseau est contrôlable. Par ailleurs, la modification des meta-contraintes numériques a induit une modification des meta-contraintes symboliques, qui correspond à la détection d'une précédence induite ($d_3 \leq d_2$). Nous savons du même coup qu'Elsa devra avoir com-

mencé sa tâche de nettoyage avant que Frank ne commence à décharger la voiture. Les meta-contraintes symboliques modifiées sont représentées dans la figure suivante. Rappelons-nous néanmoins que les meta-contraintes symboliques ne figurent pas explicitement dans $\mathcal{D}$. La section suivante va s'employer notamment à fournir un mécanisme direct de détection des précédences induites telles qu'ici ($d_3 \leq d_2$).

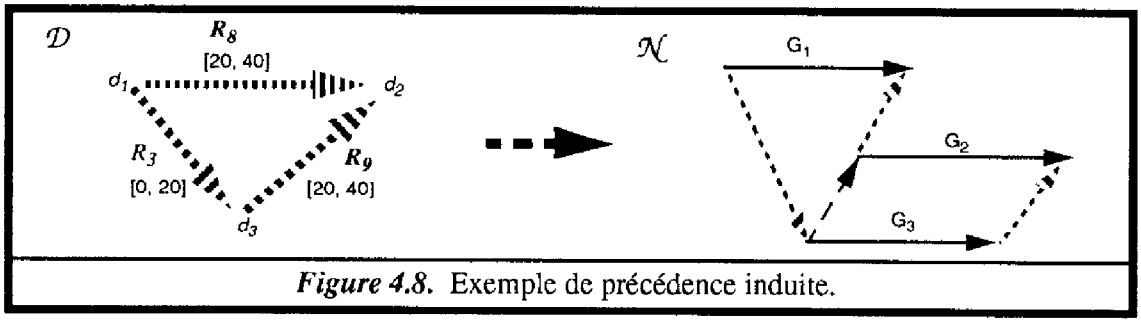


Figure 4.8. Exemple de précédence induite.

Mais avant d'entrer dans le vif du sujet, attardons-nous un instant sur notre cas particulier du § 4.3.2, que nous avions momentanément laissé de côté.

4.5. Un Cas Particulier

Nous revenons ici au cas de la meta-contrainte symbolique $R_2=\{(f_1=d_2)\}$, ainsi que sa relation inverse Ri_2, que nous avons tout d'abord exclues de notre formalisme. Ce paragraphe se veut un premier pas très informel vers une prise en compte satisfaisante de ce type de contrainte. Il s'agira donc de donner des pistes prometteuses, sans avoir pour autant la prétention d'affirmer quoi que ce soit, le travail sur cet aspect n'étant pour l'instant qu'à l'état d'ébauche.

Comme nous l'avons dit rapidement au § 4.3.2, la CTG G_2 contredit l'hypothèse préliminaire selon laquelle une CTG est définie entre une variable de Déclenchement et une variable d'Attente. En fait, si l'on analyse le problème d'un point de vue sémantique, à la variable d'Attente f_2 est associée une variable de Déclenchement. Cette variable de Déclenchement n'est pas d_2 (qui est en fait une variable d'Attente: $d_2=f_1$), mais bel et bien d_1 ! C'est-à-dire que l'on *attend* l'occurrence de f_2 à partir du moment où d_1 est déclenché, puisque G_2 succédera immédiatement et automatiquement à G_1. La CTG G_2 définie entre d_2 et f_2 peut donc être transformée sans perte d'information dans notre réseau en une contrainte $G_3=G_1 \oplus G_2=[inf_1 + inf_2, sup_1 + sup_2]$ définie entre $d_3=d_1$ et $f_3=f_2$. Ceci est représenté sur la figure ci-dessous, où nous avons ajouté un exemple de valuation numérique et une troisième CTG G_0 reliée aux deux premières, ce qui va nous permettre d'illustrer les limites de cette approche.

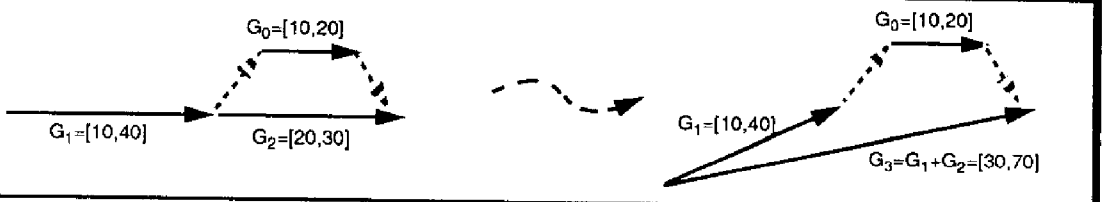


Figure 4.9. Transformation d'une relation R_2 en R_1.

Ceci revient donc à transformer une meta-contrainte symbolique de type R_2 en une meta-contrainte symbolique de type R_1. On voit clairement que cette transformation définit les mêmes contraintes, aussi bien symboliques que numériques, entre d_1, f_1=d_2 et f_2. Par contre, elle contredit le prérequis d'indépendance des variables aléatoires posé au début de ce chapitre (cf § 2.1). En d'autres termes, la durée de G_3 dépend de celle de G_1.

Analysons ceci au travers de l'exemple numérique ci-dessus: le premier réseau est contrôlable, car il suffit de déclencher G_0 en même temps que G_2 pour être sûr de vérifier G_0 *"during"* G_2. Par contre, le deuxième réseau (qui est censé être équivalent) n'est pas contrôlable. En effet, si je considère la situation $\omega=\{\omega_1=40, \omega_3=30, \omega_0=20\}$, je constate que les contraintes de précédence ne sont pas satisfaites. Il n'y a donc **pas équivalence** entre les deux réseaux de la figure. L'explication en est fort simple: ω_1 dépendant de ω_3, l'occurrence simultanée de ω_1=40 et ω_3=30 est en fait impossible.

Plusieurs enseignements sont à dégager de cet exemple, lesquels peuvent nous conduire dans une direction ou une autre. Ces considérations ne sont données que de manière intuitive, le travail sur ces aspects relevant encore actuellement du domaine des prospectives.

1. Nous pouvons chercher un formalisme plus riche nous permettant de prendre en compte la dépendance entre CTGs. Cette solution demande a priori une recherche approfondie à long-terme, qui sort du cadre de ce mémoire (voir les prospectives au chapitre 6, § 2.1)).

2. Nous pouvons conserver la solution consistant à transformer la relation R_2 en R_1, sachant que

   - il faudrait pour cela démontrer un résultat qui semble a priori avéré, à savoir que la nouvelle relation est plus restrictive, c'est-à-dire qu'elle risque seulement de nous amener à rejeter des situations contrôlables; par contre, à l'inverse, si la relation R_1 est contrôlable, alors la relation R_2 d'origine est contrôlable, ce qui constitue une exigence minimale de notre système: rejeter toute situation incontrôlable à l'exécution.

- nous pourrions également démontrer que si l'instant $f_1 = d_2$ n'est relié à aucun autre instant du graphe par un Lien, alors la contrôlabilité du réseau transformé est strictement équivalente à la contrôlabilité du réseau initial. Ceci nous permettrait de définir une famille de problèmes élargie pour laquelle notre modèle est complet.

♣♣♣

Maintenant que le paradigme du Graphe de Décision a été correctement posé, nous allons voir dans la section suivante l'intégration du réseau $\mathcal{D}$ dans notre système IxTeT avec tout ce que cela implique en termes de choix de représentation interne, de traitement des requêtes, ou d'interactions entre $\mathcal{S}$, $\mathcal{N}$et $\mathcal{D}$.

5. Application au Système IxTeT

Il est à noter qu'à l'heure où ce mémoire s'apprête à être mis sous presse, l'intégration réelle dans le gestionnaire de contraintes temporelles du planificateur IxTeT reste encore à faire. Cette section se veut dont une réflexion avancée sur cette intégration, avec la mise en place des algorithmes qui s'avéreront nécessaires. Il y manque néanmoins malheureusement des résultats expérimentaux et un exemple réel d'application au domaine de la planification, éléments qui auraient pu avantageusement étayer notre propos.

Retenons simplement qu'il va nous falloir retrouver des fonctionnalités identiques à celles du chapitre 3, en termes de **mises-à-jour** et de **requêtes**. Nous allons tout d'abord donner quelques éléments de codage du graphe $\mathcal{D}$, après quoi nous établirons les principales interactions nécessaires, à savoir la prise en compte dans $\mathcal{D}$ d'un ajout numérique dans $\mathcal{N}$, et la détection et le report dans $\mathcal{S}$ de précédences induites durant la propagation dans $\mathcal{D}$. Nous verrons aussi dans ce cadre ce qu'il convient de penser de la gestion des requêtes temporelles. Enfin, nous évoquerons également, quoique succinctement, l'apport que peut représenter le Graphe de Décision pour le contrôle de l'exécution du plan produit.

5.1. Représentation Interne du Graphe de Décision: Quelques Eléments de Base

De la même manière que le graphe $\mathcal{N}$était codé à l'intérieur du graphe global $\mathcal{S}$ (cf chapitre 3), il serait avantageux de suivre un principe similaire pour le codage interne du Graphe de Décision. En premier lieu, nous définissons pour les besoins de la cause une fonction *Source*

qui à toute variable d'Attente associe la variable de Déclenchement à laquelle elle est liée. En effet, intuitivement, une variable d'Attente est soit un événement dans le sens où on l'entend au niveau du planificateur, c'est-à-dire un changement instantané de l'état du monde indépendant de l'activité de l'agent, soit il s'agit de l'instant de fin d'une tâche. Dans le premier cas, l'instant initial 0 peut être considéré comme source de l'événement, dans le deuxième cas, l'instant de début de la tâche est fort naturellement à l'origine de l'attente de l'instant de fin.

Plus formellement, une variable d'Attente est toujours un instant de fin d'une CTG, laquelle est du fait de nos hypothèses restrictives une contrainte forte. Nous pouvons donc définir une fonction *Source* qui associe à chaque variable d'Attente les instants de début des CTGs associées. Il pourrait en effet a priori y en avoir plusieurs, mais nous avons vu qu'un instant ne peut être à la fois instant terminal de deux CTGs (par interdiction de la contrainte $(f_1 = f_2)$). Donc la fonction *Source* est une **application de V_A vers V_D** (cf § 3.2.1):

$\forall f_k \in V_A, \exists! d_k \in V_D$ tel que $Source(f_k)=d_k$.

Remarque : De plus, nous pouvons remarquer que cette application est **surjective**, puisque tout instant ajouté l'est en même temps qu'une CTG, donc tout instant de V_D ajouté est un instant début d'une CTG, c'est-à-dire:
$\forall d_k \in V_D, \exists$ au moins un $f_k \in V_A$ tel que $Source(f_k)=d_k$.

Nous pouvons étendre cette application à l'ensemble des instants numériques: à tout instant $d_k \in V_D$, nous associons l'instant d_k lui-même, tout simplement, par l'application identité. Cette extension conduit à la définition de l'application $Source^{\uparrow}$:

Définition 4.14 : $Source^{\uparrow}: V_n \rightarrow V_A$ se définit par:
$\forall f_k \in V_A$, $Source^{\uparrow}(f_k)=d_k \in V_D$ / $G_k \in \Gamma$ est définie entre d_k et f_k.
$\forall d_k \in V_D$, $Source^{\uparrow}(d_k)=d_k$.

Dès lors, nous pouvons disposer pour tout instant d'un attribut *source* dans lequel sera placée son image par l'application $Source^{\uparrow}$. A l'ajout d'une CTG G_k, il suffit alors d'effectuer les affectations $source(d_k) \leftarrow d_k$ et $source(f_k) \leftarrow d_k$. Et à tout moment dans les algorithmes qui suivront, nous pourrons

1. vérifier la nature d'un instant relativement à la dichotomie des variables:
   si $source(i) = i$, alors i est une variable de Déclenchement, sinon il s'agit d'une variable d'Attente.
2. connaître l'instant de début d'une CTG à partir de son instant de fin.

Non seulement l'instant de début d'une CTG est le lieu naturel de l'expression des meta-contraintes numériques $MN_{1,2}$, mais en plus, grâce au paramètre *source*, nous pouvons retrouver cet instant de début à partir de l'instant de fin. Nous avons donc la possibilité, en présence de n'importe quel instant, d'identifier la CTG à laquelle il se rattache, et de retrouver ou d'affecter les meta-contraintes associées à cette CTG.

Nous voyons donc qu'il est possible de mettre en place une **relation directe et univoque entre les CTGs de $\mathcal{N}$et les variables du Graphe de Décision**, ce qui permet de gérer facilement celui-ci à l'intérieur-même du graphe $\mathcal{N}$.

5.2. Processus d'Interaction Elémentaires

5.2.1. Prise en Compte d'un Ajout Numérique dans le Graphe de Décision

Nous avons dit que les meta-contraintes symboliques n'étaient pas gérées explicitement dans notre modèle. Nous avons en fait implicitement montré que les meta-contraintes symboliques sont incluses dans les meta-contraintes numériques, ce qui généralise la vision du symbolique comme un cas particulier du numérique, constatée dans le graphe $\mathcal{G}$, notamment dans [Dechter91] (cf chapitre 2, § 2.4.1).

On doit donc traduire directement un ajout dans $\mathcal{N}$en un ajout au niveau des meta-contraintes numériques dans $\mathcal{D}$. Dans nos hypothèses restrictives, un ajout se limite à une nouvelle CTG forte reliée aux CTGs déjà présentes dans $\mathcal{N}$par l'intermédiaire de simples contraintes symboliques (les Liens). Chaque Lien relie un nouvel instant à un instant déjà présent dans $\mathcal{N}$. Dès lors, il est toujours possible de se ramener à une contrainte entre variables de Déclenchement, grâce notamment au champ *source*.

Soit G_{ij} la nouvelle CTG entre i et j. Nous revenons là à une formulation similaire à celle du chapitre 3. Nous considérons implicitement dans la suite que i est le début de G_{ij} (donc une variable de Déclenchement, donc présente dans $\mathcal{D}$), et j est son instant de fin. j ne peut pas être égal à un instant déjà présent dans $\mathcal{D}$, du fait que l'on a exclu à la fois les situations du type ($f_1 = f_2$), incontrôlables par définition, et ($f_1 = d_2$), qui correspond au cas particulier identifié au § 4.5. Nous allons passer en revue de manière informelle ci-après tous les cas de figure possibles en termes de **traduction d'une contrainte symbolique élémentaire en meta-contrainte numérique** dans $\mathcal{D}$. La figure ci-dessous présente de manière graphique ces cas de figure. Nous nous référons ci-après aux notions de Liens convergents et divergents vues au chapitre 3 (§ 5.3).

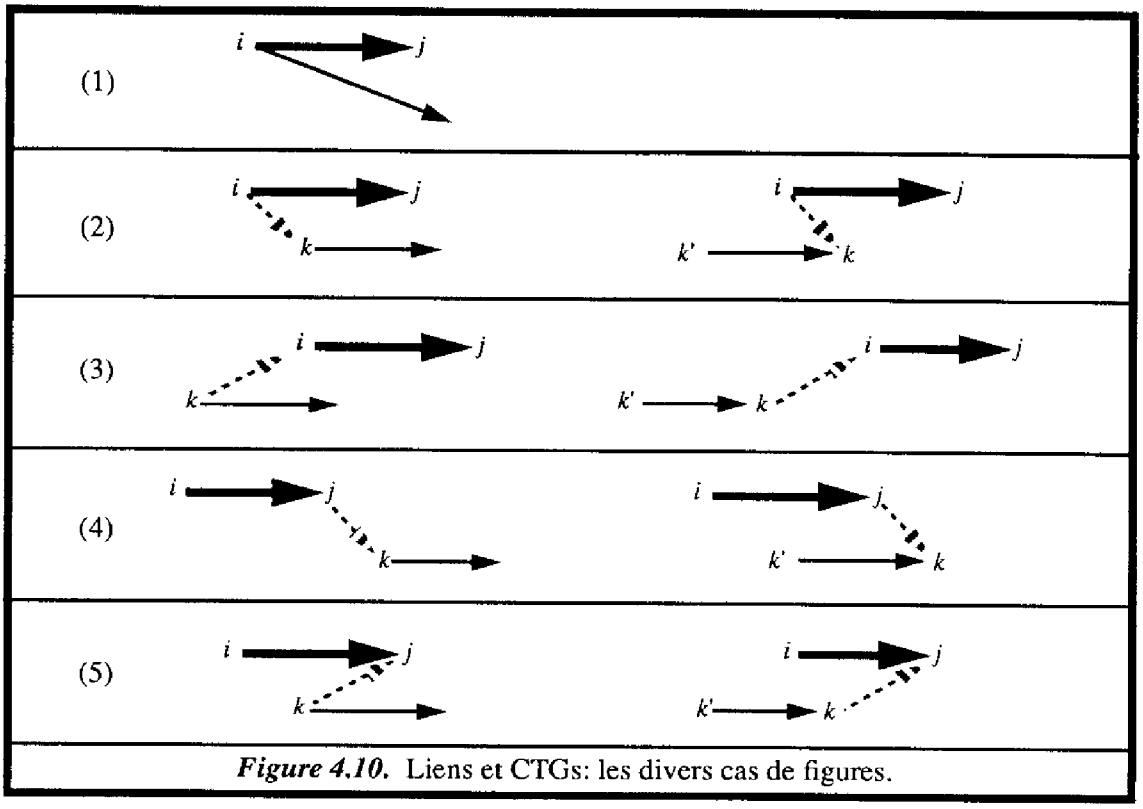


Figure 4.10. Liens et CTGs: les divers cas de figures.

1. Si $\exists\, k \,/\, k = i$, alors $MN_{ik} \leftarrow \{0\}$ (ce qui conduit en fait à une agrégation de k et i).

2. Lien divergent de i: si $\exists\, k \,/\, i \leq k$, alors deux cas peuvent se produire:
   - si $Source^{\uparrow}(k)=k$, alors $MN_{ik} \leftarrow [0, +\infty[$.
   - si $Source^{\uparrow}(k)=k'$, alors il va s'agir définir la meta-contrainte numérique entre i et k'. La précédence définit implicitement une relation R_5 entre G_{ij} et $G_{k'k}$, ce qui nous donne donc $MN_{ik'} \leftarrow [-inf_{k'k}, +\infty[$.

3. Lien convergent vers i: si $\exists\, k \,/\, k \leq i$, alors deux cas peuvent se produire:
   - si $Source^{\uparrow}(k)=k$, alors $MN_{ik} \leftarrow]-\infty, 0]$.
   - si $Source^{\uparrow}(k)=k'$, alors on a une relation Ri_4 entre G_{ij} et $G_{k'k}$, ce qui nous donne donc $MN_{ik'} \leftarrow]-\infty, -sup_{k'k}]$.

4. Lien divergent de j: si $\exists\, k \,/\, j \leq k$, alors deux cas peuvent se produire:
   - si $Source^{\uparrow}(k)=k$, alors on a une relation R_4 entre G_{ij} et une (ou plusieurs) $G_{kk'}$ quelconque(s), ce qui nous donne donc $MN_{ik} \leftarrow [sup_{ij}, +\infty[$.
   - si $Source^{\uparrow}(k)=k'$, alors on a une relation R_6 entre G_{ij} et $G_{k'k}$, ce qui nous donne donc $MN_{ik'} \leftarrow [sup_{ij}\text{-}inf_{k'k}, +\infty[$.

5. Lien convergent vers j: si $\exists\, k \,/\, k \leq j$, alors deux cas peuvent se produire:
   - si $Source^{\uparrow}(k)=k$, alors on a une relation Ri_5 entre G_{ij} et une (ou plusieurs) $G_{kk'}$ quelconque(s), ce qui nous donne donc $MN_{ik} \leftarrow]-\infty, inf_{ij}]$.
   - si $Source^{\uparrow}(k)=k'$, alors on a une relation Ri_6 entre G_{ij} et $G_{k'k}$, ce qui nous donne donc $MN_{ik'} \leftarrow]-\infty, inf_{ij}\text{-}sup_{k'k}]$.

Nous avons fait le tour de tous les cas de Liens possibles, en tenant compte également du cas (1) où i est déjà présent dans $\mathcal{N}$. Cela permet de définir l'interaction de $\mathcal{N}$ vers $\mathcal{D}$. Les algorithmes formels de prise en compte dans $\mathcal{D}$ sont à partir de là relativement aisés à mettre en place. Pour traiter les cas (3) et (5) par exemple, la fonction *AJOUTE-ARCS-CONVERGENTS* (cf chapitre 3, § 5.3) doit être modifiée de la manière suivante:

AJOUTE-ARCS-CONVERGENTS (i, j)
  TantQue $Conv_i \neq \varnothing$ Faire
  $k \leftarrow$ Dépiler($Conv_i$)
  Si $k \leq i$ Alors
    Si $source(k)=k$ Alors
      $MN_{ik} \leftarrow]-\infty, 0]$
      $MAJ \leftarrow MAJ \cup \{(k,i)\}$.
    Sinon
      $k' \leftarrow source(k)$
      $MN_{ik'} \leftarrow]-\infty, -sup_{k'k}]$
      $MAJ \leftarrow MAJ \cup \{(k',i)\}$.
    Retirer k de $Conv_j$.
    Pour tout $l \in Conv_i$ / $inf_{lk} \geq 0$ (l ancêtre de k)
      Retirer l de $Conv_i$ et de $Conv_j$.
      $N_{lk} \leftarrow N_{lk} \cap [0, +\infty[$

  TantQue $Conv_j$ non vide Faire
  $k \leftarrow$ Dépiler($Conv_j$)
  Si $k \leq j$ Alors
    Si $source(k)=k$ Alors
      $MN_{ik} \leftarrow]-\infty, inf_{ij}]$
      $MAJ \leftarrow MAJ \cup \{(k,i)\}$.
    Sinon
      $k' \leftarrow source(k)$
      $MN_{ik'} \leftarrow]-\infty, inf_{ij}\text{-}sup_{k'k}]$
      $MAJ \leftarrow MAJ \cup \{(k',i)\}$.
    Pour tout $l \in Conv_j$ / $inf_{lk} \geq 0$ (l ancêtre de k)
      Retirer l de $Conv_j$.
      $N_{lk} \leftarrow N_{lk} \cap [0, +\infty[$

La modification de l'algorithme *AJOUTE-ARCS-DIVERGENTS* se déduit de même à partir des cas (2) et (4). Notons par ailleurs que le principe ci-dessus s'applique également au cas plus simple de l'**ajout d'une simple précédence** dans $\mathcal{S}$. En effet, en utilisant à la fois les prin-

cipes énoncés au chapitre 3 et ceux évoqués ci-dessus, il est possible de transformer cette précédence en un ensemble de meta-contraintes numériques qui seront alors intersectés avec celles déjà présentes dans $\mathcal{D}$.

5.2.2. Détection des Précédentes Induites

Il reste à considérer l'interaction inverse, à savoir: lorsqu'une meta-contrainte numérique est modifiée, quelles modifications induit-elle au niveau symbolique ? Nous devons donc retrouver les fonctionnalités de détection de précédences induites présentes au chapitre 3.

Tout d'abord, l'algorithme PC-2 induit une restriction du domaine des meta-contraintes numériques, ce qui ne peut que surcontraindre la meta-contrainte symbolique correspondante, c'est-à-dire faire apparaître une nouvelle précédence élémentaire entre deux instants début ou fin des CTGs reliées par la meta-contrainte. Il suffit donc de reprendre les informations contenues dans le tableau 4.1, en raisonnant dans le sens inverse: quelle est la condition pesant sur le domaine de la meta-contrainte numérique pour qu'une précédence soit identifiée ? Nous avons quatre précédences induites possibles (plus les relations inverses), que nous allons étudier tour à tour.

1. $(f_1 \leq d_2)$ ssi dans $\mathcal{D}$,
   $\forall \omega_1 \in [inf_1, sup_1], d_1 + \omega_1 \leq d_2$
   $\Leftrightarrow d_1 + sup_1 \leq d_2$
   $\Leftrightarrow d_2 - d_1 \geq sup_1$
   $\Leftrightarrow inf_{d_1 d_2} \geq sup_1$
   [resp. $(f_2 \leq d_1)$ ssi $sup_{d_1 d_2} \leq -sup_2$]

2. $(d_1 \leq d_2)$ ssi dans $\mathcal{D}$,
   $inf_{d_1 d_2} \geq 0$
   [resp. $(d_2 \leq d_1)$ ssi $sup_{d_1 d_2} \leq 0$]

3. $(f_1 \leq f_2)$ ssi dans $\mathcal{D}$,
   $\forall \omega_1 \in [inf_1, sup_1], \forall \omega_2 \in [inf_2, sup_2], d_1 + \omega_1 \leq d_2 + \omega_2$
   $\Leftrightarrow d_1 + sup_1 \leq d_2 + inf_2$
   $\Leftrightarrow d_2 - d_1 \geq sup_1 - inf_2$
   $\Leftrightarrow inf_{d_1 d_2} \geq sup_1 - inf_2$
   [resp. $(f_2 \leq f_1)$ ssi $sup_{d_1 d_2} \leq inf_1 - sup_2$]

4. $(d_1 \leq f_2)$ ssi dans $\mathcal{D}$,
   $\forall \omega_2 \in [inf_2, sup_2], d_1 \leq d_2 + \omega_2$
   $\Leftrightarrow d_1 \leq d_2 + inf_2$

$\Leftrightarrow d_2 - d_1 \geq -inf_2$
$\Leftrightarrow inf_{d_1d_2} \geq -inf_2$
[resp. $(d_2 \leq f_1)$ ssi $sup_{d_1d_2} \leq inf_1$]

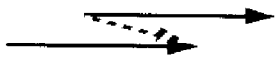
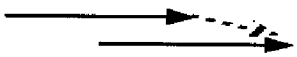
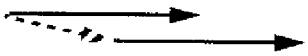
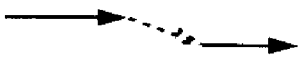
Test sur la meta-contrainte numérique	Précédence Induite
$inf_{d_1d_2} \geq -inf_2$	
$inf_{d_1d_2} \geq sup_1 - inf_2$	
$inf_{d_1d_2} \geq 0$	
$inf_{d_1d_2} \geq sup_1$	

Tableau 4.2. Détection de précédences induites

Le parallèle peut être fait avec les relations R_3 à R_6 illustrées dans le tableau 4.1. Nous avons simplement ici mené le raisonnement dans l'autre sens, comme l'illustre le tableau 4.2 ci-dessus. Il reste les cas des égalités: $(d_1 = d_2)$ sera automatiquement induit dès lors que l'on aura induit $(d_1 \leq d_2)$ & $(d_2 \leq d_1)$. Et $(d_2 = f_1)$ [resp. $(d_1 = f_2)$] n'a aucune chance d'être induit: en effet, la conjonction des conditions correspondant à $(f_1 \leq d_2)$ & $(d_2 \leq f_1)$ conduit à la contradiction $sup_1 \leq inf_{d_1d_2} \leq sup_{d_1d_2} \leq inf_1$.

Nous avons donc tous les cas de figures possibles, qui ont été volontairement ordonnés du plus contraint au moins contraint, c'est-à-dire que

- $(f_1 \leq d_2) \Rightarrow (d_1 \leq d_2)$, qui est donc redondante, par dominance, et ne doit pas être reportée dans $\mathcal{S}$ (cf chapitre 3, § 5.4).
- et de la même manière $(f_1 \leq d_2) \Rightarrow (f_1 \leq f_2) \Rightarrow (d_1 \leq f_2)$.

Nous devrons donc tester à chaque modification de meta-contrainte numérique si cette modification conduit à l'apparition d'une nouvelle précédence, en commençant par tester le cas plus contraint. L'algorithme qui figure ci-dessous devra être intégré dans l'algorithme de propagation PC-2, de la même manière que la détection des précédences induites dans l'algorithme de propagation du Sous-Graphe Numérique (cf chapitre 3, § 5.4), c'est-à-dire en particulier en filtrant l'ensemble *REPORTS* des précédences induites par la relation de dominance. Ce paragraphe a donc permis de définir l'**interaction de $\mathcal{D}$ vers $\mathcal{S}$**.

Si $inf_{d_1d_2} \geq sup_1$ Alors AJOUT-DOMIN(f_1, d_2, *REPORTS*).
Sinon
  Si $sup_{d_1d_2} \leq -sup_2$ Alors AJOUT-DOMIN(f_2, d_1, *REPORTS*).
  Sinon
    Si $inf_{d_1d_2} \geq 0$ Alors AJOUT-DOMIN(d_1, d_2, *REPORTS*).
    Sinon
      Si $sup_{d_1d_2} \leq 0$ Alors AJOUT-DOMIN(d_2, d_1, *REPORTS*).
      Sinon RIEN ← Vrai
    Si $inf_{d_1d_2} \geq sup_1 - inf_2$ Alors AJOUT-DOMIN(f_1, f_2, *REPORTS*).
    Sinon
      Si $sup_{d_1d_2} \leq inf_1 - sup_2$ Alors AJOUT-DOMIN(f_2, f_1, *REPORTS*).
      Sinon RIEN ← Vrai
    Si RIEN Alors
      Si $inf_{d_1d_2} \geq -inf_2$ Alors AJOUT-DOMIN(d_1, f_2, *REPORTS*).
      Sinon
        Si $sup_{d_1d_2} \leq inf_1$ Alors AJOUT-DOMIN(d_2, f_1, *REPORTS*).

5.3. Prise en Compte d'un Ajout à l'Etape de Planification

5.3.1. Processus Global d'Interaction lors d'une Etape d'Ajout

En fonction des principes généraux d'interaction entre $\mathcal{S}$, $\mathcal{N}$et $\mathcal{D}$ présentés ci-dessus, nous pouvons désormais présenter de manière informelle ces interactions dans le cadre d'une étape d'ajout incrémental d'une CTG, effectuée par le planificateur. Nous allons pour cela reprendre rapidement la logique de présentation du chapitre 3, en nous focalisant sur le cas d'ajout le plus complexe, c'est-à-dire l'**ajout d'une CTG étiquetée G_{ij} entre deux instants non numériques**.

La première étape (1) va consister à ajouter les précédences sous-jacentes dans $\mathcal{S}$, qui est mis à jour (2). Il s'agit ensuite de déterminer les Liens, qui sont ajoutés dans $\mathcal{N}$(3) en même temps que la contrainte G_{ij} elle-même (3'). Jusque-là, rien de changé, sauf que nous n'allons pas propager dans $\mathcal{N}$, mais dans $\mathcal{D}$. La mise-à-jour de $\mathcal{D}$ se fait facilement en calculant les nouvelles meta-contraintes numériques à partir des Liens (4) et des CTGs (4'), selon le principe énoncé au paragraphe 5.2.1 ci-dessus. Il ne nous reste désormais plus qu'à propager dans $\mathcal{D}$ (5), ce qui permet d'induire de nouvelles précédences élémentaires reportées (6) et propagées (2) dans $\mathcal{S}$, selon le principe énoncé au paragraphe 5.2.2 ci-dessus. De la même manière, on souhaiterait pouvoir en cours de propagation dans $\mathcal{D}$, induire de la modification d'une meta-contrainte numérique, les modifications dans $\mathcal{N}$essentiellement des contraintes de Déclenchement (7).

Ces étapes peuvent être visualisées dans la figure suivante, où l'on peut observer que les étapes (1) et (2), (3) et (3') proviennent directement du chapitre 3, et que les étapes (4), (4'), (5) et (6) ont été décrites dans les paragraphes précédents. Il reste donc à étudier le cas de l'étape (7).

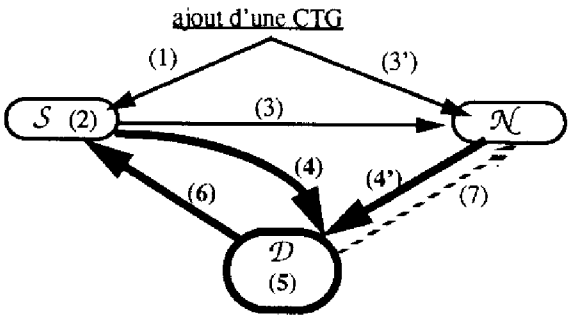


Figure 4.11. Architecture globale du système temporel.

♣♣♣

Concernant l'arc (7) de la figure, il semble naturel de vouloir récupérer dans $\mathcal{N}$ les valuations de chaque CLB, c'est-à-dire de connaître la marge disponible sur chaque contrainte de Déclenchement et chaque contrainte d'Attente. En fait, l'utilité de disposer de ces valuations ne s'impose qu'en fonction de trois considérations distinctes et qui ont déjà été invoquées. Nous les rappelons succinctement ci-après:

1. disposer, par l'intermédiaire d'un graphe complet, des **contraintes symboliques induites**: cette exigence est satisfaite par le Graphe de Décision, et ne peut donc servir d'argument à une explicitation de contraintes dans $\mathcal{N}$;
2. fournir au superviseur d'exécution des **valeurs de contrôle optimales**: il est uniquement nécessaire pour lui de récupérer des valeurs sur les contraintes de Déclenchement, seules contraintes dont l'instanciation soit effectivement de son ressort;
3. disposer de **requêtes efficaces**.

Ces deux dernières exigences ne peuvent être ignorées. Voyons tout d'abord comment obtenir la marge d'une contrainte de Déclenchement. Si l'on se reporte au tableau 4.1, on peut voir que seuls deux cas de figures peuvent se présenter:

- la contrainte de Déclenchement est $L_{d_1 d_2}$: dans ce cas, nous disposons directement de cette contrainte dans $\mathcal{D}$,
- la contrainte de Déclenchement est $L_{f_1 d_2}$: dans ce cas-là, nous pouvons déterminer $L_{f_1 d_2}$ en fonction de $L_{d_1 d_2}$ de la manière suivante:

Propriété 3.6 : $\forall(G_1, G_2)\in\Gamma^2 / G_1=[inf_1, sup_1]$ et $L_{d1d2}=[inf_{d1d2}, sup_{d1d2}]$, on a: $L_{f1d2}=[inf_{d1d2}-inf_1, sup_{d1d2}-sup_2]$.

Nous ne détaillerons pas le raisonnement menant à ce calcul. Retenons simplement que L_{f1d2} se réduit aux durées pouvant être a priori affectées en garantissant la contrôlabilité de l'ensemble, ce qui amène à considérer la CTG G_1 dans une perspective "pessimiste". C'est pourquoi le calcul diffère sensiblement, et va même à l'encontre, de la composition classique utilisée dans les STPs. La principale conséquence est qu'ici l'exécution va **relâcher** certaines contraintes (du fait d'une levée partielle de l'incertitude), et donc éventuellement

(1) rendre contrôlable des contraintes du type L_{f1d2} a priori incontrôlables,

et (2) permettre une affectation "opportuniste" de ces contraintes en cours d'exécution.

Ceci est illustré par la figure suivante, par l'intermédiaire de deux exemples numériques. Dans le premier cas (a), nous voyons que la contrainte L_{f1d2} obtenue est a priori incontrôlable, du fait qu'elle dépend de la valeur prise par la CTG G_1. Il faudra donc attendre l'occurrence de ω_1 pour décider de sa valeur effective. Dans le deuxième exemple (b), nous voyons qu'il serait a priori nécessaire de fixer la valeur de L_{f1d2} à {30}. En fait, là aussi nous sommes dépendant de l'exécution de G_1. $L_{f1d2}=[30, 30]$ signifie en effet qu'il ne nous est pas possible a priori de forcer cette contrainte à durer ni moins de 30, ni plus de 30. Si nous le faisions, nous perdrions la propriété de contrôlabilité de l'ensemble du réseau. Par contre, considérons qu'en cours d'exécution, nous observions $\omega_1=\{10\}$. Dans ce cas, nous pouvons mettre à jour le réseau et nous rendre compte que désormais $L_{f1d2}=[20, 30]$. Donc, 30 est bien évidemment toujours une valeur admissible, mais nous préférerons déclencher au plus tôt en choisissant la valeur {20}.

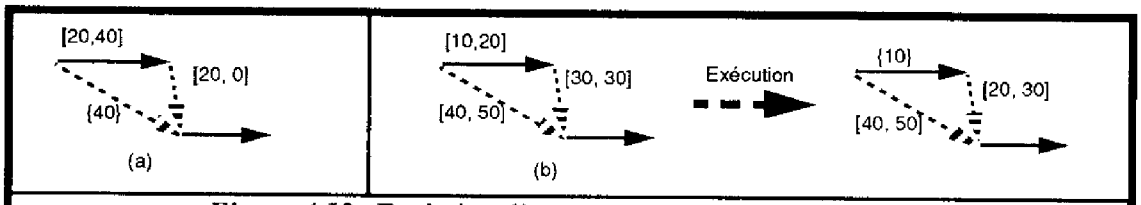


Figure 4.12. Evolution d'une contrainte de déclenchement.

Ce phénomène est essentiellement dû au fait que l'on se contente de vérifier la contrôlabilité du réseau, et non sa forte contrôlabilité (cf § 2.5). En effet, si le réseau était fortement contrôlable, ce problème de dépendance n'apparaîtrait pas.

Bref, dans le cas d'une contrainte de Déclenchement L_{f1d2}, il sera de toutes façons préférable d'attendre d'observer l'évolution de la situation en cours d'exécution pour déterminer la décision optimale. On conclut qu'il n'est pas nécessaire de récupérer la valuation de ces contraintes de Déclenchement dans $\mathcal{N}$. Ces aspects de contrôle en cours d'exécution seront repris et précisés plus loin.

Il nous reste à considérer l'exigence de requêtes performantes évoquée plus haut. Ceci sera fait dans quelques paragraphes. Anticipons cependant quelque peu: nous avons vu de par la propriété ci-dessus que nous pouvions récupérer instantanément la contrainte L_{fid_2} en fonction de $L_{d_1d_2}$. Nous verrons que ceci est une propriété très générale, et que le graphe complet de $\mathcal{D}$ nous permet de disposer de n'importe quelle distance temporelle dans $\mathcal{N}$ en un temps constant (cf § 5.4).

De fait, tous les arguments dont nous disposions pour expliciter le graphe $\mathcal{N}$ en fonction de la propagation dans $\mathcal{D}$ ont volé en éclat les uns après les autres. D'où nous pouvons conclure que l'arc (7) de notre figure 4.10 est superflu (ce pourquoi il figurait en pointillés): ce type d'interaction n'a pas lieu d'être au niveau d'une mise-à-jour. Il sera par contre présent au niveau des requêtes.

5.3.2. Vérification de Contrôlabilité A Priori

Le lecteur se souvient probablement de la possibilité décrite au chapitre 3 (§ 6.3) de vérifier la cohérence du réseau a priori. Nous disposons ici d'une possibilité similaire:

Propriété 3.7 : Soit G_i la CTG ajoutée. Soit $\Gamma_{link} \subset \Gamma$ l'ensemble des CTGs liées à G_i par des Liens. Alors:
L'ajout de G_i dans $\mathcal{D}$ est contrôlable ssi $\forall (G_k, G_{k'}) \in \Gamma_{link}{}^2$, $MN_{ki} \oplus MN_{ik'} \cap MN_{kk'} \neq \emptyset$

C'est-à-dire que nous appliquons un principe similaire à celui de la vérification de cohérence a priori, mais à un niveau au-dessus: pour tout couple (G_i, G_k), on doit ajouter dans $\mathcal{D}$ une meta-contrainte numérique MN_{ik}. Il suffit de vérifier que tout couple de meta-contraintes numériques $(MN_{ik}, MN_{ik'})$ ne rend pas la meta-contrainte $MN_{kk'}$ déjà présente dans $\mathcal{D}$ incontrôlable, c'est-à-dire que l'on pratique une **analyse a priori de triplets de CTGs** directement impliquées par l'ajout. La propriété n'est bien sûr vérifiée que dans la mesure où le graphe $\mathcal{D}$ est complet, ce qui est le cas grâce à la propagation par cohérence de chemin.

5.3.3. Complexités

Une première analyse de complexité peut être menée pour chaque étape. Rappelons tout d'abord que n est la dimension du graphe symbolique $\mathcal{S}$, m est la dimension du graphe numérique $\mathcal{N}$, que nous avons dénommé STPU dans ce chapitre, et p est le degré de parallélisme de $\mathcal{N}$, défini au chapitre 3 (§ 5.6).

(1-2)- Ajout dans $\mathcal{S}$ et propagation: $\mathbf{O(n)}$.

(3)- Détermination des Liens: $\mathbf{O(p.n)}$.

(4)- Détermination des meta-contraintes numériques correspondant aux Liens: le nombre de Liens est de l'ordre de p et chaque transformation se fait en temps constant, ce qui donne du $\mathbf{O(p)}$ en pire cas.
(5)- Propagation: le graphe $\mathcal{D}$ contient environ moitié moins de noeuds que le graphe $\mathcal{N}$, ce qui entraîne une augmentation des performances par un facteur proche de 2, donc l'ordre de grandeur reste le même, c'est-à-dire $O(p.m^2)$. N'oublions pas qu'avant de propager, nous effectuons une opération de vérification de contrôlabilité a priori. Celle-ci nécessite d'observer tous les couples $(G_k, G_{k'})$, ce qui nous donne donc une complexité en $O(p^2)$. Au total, nous récupérons donc du $\mathbf{O(p.(p + m^2))}$.

(6)- Détection et report des précédences induites: la complexité de cette phase, comme dans le chapitre précédent, repose essentiellement sur le filtrage par dominance. On obtient de fait la même complexité en ordre de grandeur que précédemment: $\mathbf{O(p.m^2)}$.

Au total, nous obtenons donc une complexité pour l'ensemble de la phase d'ajout en $\mathbf{O(p.(n + p + m^2))}$, c'est-à-dire très proche de celle de la phase d'ajout dans $\mathcal{N}$ au chapitre précédent, qui rappelons-le était en $O(p.(n + m^2))$. C'est dire que seul le degré de parallélisme p va jouer ici un rôle plus déterminant, c'est-à-dire que les performances du modèle du Graphe de Décision seront **comparables** à celles du sous-graphe numérique dans la mesure où p sera faible, voire borné, c'est-à-dire dans le cas de réseaux séquentiels (cf chapitre 3, notamment le paragraphe 5.6).

Par ailleurs, même si les complexités sont comparables en ordre de grandeur, cela veut simplement dire que l'évolution des performances au fur et à mesure que la taille du graphe augmente est comparable à ce qu'elle était dans le sous-graphe numérique. Par contre, on pourrait a priori s'attendre à des modifications en termes de temps de calcul effectifs.

Si l'on cherche à effectuer une comparaison selon ce critère, deux remarques s'imposent alors:

1. L'ajout d'une structure supplémentaire telle que $\mathcal{D}$ suppose la gestion d'interactions supplémentaires, qui, quoiqu'en temps constant, vont sensiblement diminuer l'efficacité réelle en termes de temps de calcul.
2. Le taille du graphe $\mathcal{D}$ est inférieure à celle de $\mathcal{N}$ le rapport pouvant aller jusqu'à 1/2. Ceci bien sûr divise le temps de propagation d'autant.

Ces deux remarques nous tirent dans deux directions opposées. C'est dire que seule l'expérimentation pourrait nous permettre de comparer efficacement les temps de calcul dans $\mathcal{N}$ et dans $\mathcal{D}$. Comme nous l'avons dit en début de section, de telles mesures expérimentales n'ont malheureusement pu être menées en l'état actuel des travaux. Notre intuition, alimentée notamment par les deux remarques précédentes, nous incite néanmoins à penser que les temps de calcul seront **généralement assez proches**. Ceci sera d'autant plus vrai que p sera faible, c'est-à-dire que le réseau sera fortement séquentiel.

5.3.4. Autres cas de figure

Tout comme dans le chapitre 3, nous pourrions étudier tour à tour les cas d'ajout entre deux instants numériques déjà présents dans $\mathcal{N}$ et les cas d'ajout de simples précédences. Nous avons choisi de ne pas alourdir davantage cette section, une telle étude conduisant à des conclusions strictement similaire à ce qui a été développé au chapitre 3, c'est-à-dire que ces cas de figures sont des cas particuliers simplifiés par rapport à la description des paragraphes précédents.

5.4. Les Requêtes

Tout a été déjà dit ou presque au sujet des requêtes au paragraphe 5.3.1. Nous allons néanmoins ci-après poser les choses de manière un peu plus rigoureuse.

- Pour une requête entre deux instants quelconques de $\mathcal{N}$ nous pouvons aisément retrouver les deux CTGs G_1 et G_2 auxquelles ils sont liés, grâce au champ *source* des instants. La contrainte requise se déduit alors de $L_{d_1 d_2}$ selon un principe similaire à celui qui a été décrit au paragraphe 5.3.1 ci-dessus. Ce qui permet de conclure que la réponse à une requête dans $\mathcal{N}$ sera donnée comme précédemment **en temps constant**.

- Une requête généralisée peut se construire selon le même principe qu'au chapitre 3, l'ensemble des requêtes simples auquel on se ramène étant traité comme indiqué ci-dessus.

En conclusion, nous voyons que les requêtes vont témoigner de complexités similaires à celles du chapitre 3. Tout ceci justifie la représentation en pointillé de l'arc (7) dans la figure 4.11, cette interaction étant gérée au niveau des requêtes et non des mises-à-jour. Cette figure rend compte de fait de l'**architecture globale** de notre nouveau **modèle temporel pour la planification**. Elle constitue un complément à l'architecture présentée au chapitre 3, dans la figure 3.14.

5.5. Prise en Compte au Niveau du Contrôle d'Exécution

Nous ne donnerons ici que quelques éléments de réflexion, de manière très intuitive, le problème de contrôle de l'exécution relevant plus à l'heure qu'il est des prospectives. Le lecteur est néanmoins certainement avide de voir l'utilité de notre modèle en termes de **construction d'une solution** au STPU, sous le contrôle du superviseur d'exécution.

En fait, le plus simple est de ne gérer que le seul graphe $\mathcal{D}$, en ne considérant que les **variables décisionnelles** que sont les instants de début des CTGs: une fois qu'une tâche, c'est-à-dire une CTG G_k est déclenchée, on recherche dans $\mathcal{S}$ les instants $d_{k'}$ immédiatement successeurs de d_k, tels que $d_{k'}$ est une variable de Déclenchement. Ces variables sont celles susceptibles d'être désormais déclenchées à leur tour. Pour chaque couple $(d_k, d_{k'})$, on affecte à $L_{dkdk'}$ la valeur $inf_{dkdk'}$, c'est-à-dire que l'on choisit de **déclencher au plus tôt**. Cela revient à initialiser une variable de "compte-à-rebours", calée sur l'horloge interne du système, qui, lorsqu'elle aura atteint la valeur 0, déclenchera automatiquement la tâche associée. Il ne reste plus alors qu'à mettre à jour $\mathcal{D}$ en propageant ces valeurs $L_{dkdk'}=\{inf_{dkdk'}\}$.

Dans le cas général, une fois G_k déclenchée, nous devons en même temps rechercher dans $\mathcal{S}$ les variable de Déclenchement $d_{k'}$ succédant à f_k, c'est-à-dire les variables susceptibles d'être déclenchées à l'issue de G_k. Nous avons vu que dans le cas général, il était nécessaire d'attendre l'occurrence de $\omega_{k'}$. Dès que cette occurrence est signalée, il est alors possible d'affecter une valeur $inf_{fkdk'}$ en temps constant (cf § 5.3.1). Là aussi un compte-à-rebours sera initialisé pour permettre le déclenchement de $G_{k'}$ à la date voulue, et $\mathcal{D}$ sera mis à jour. La gestion de l'arrivée d'un événement sera effectuée de la même manière.

Se pose alors justement le problème de la mise à jour de $\mathcal{D}$, qui est quand même relativement coûteuse, en $O(m^2)$. On peut d'abord remarquer que les durées des tâches étant de l'ordre de plusieurs minutes dans les applications de planification en robotique, et les mises-à-jour ayant lieu durant ces exécutions, cette complexité n'est a priori pas très problématique. Néanmoins, le cas de figure où plusieurs événements sont observés dans un laps de temps très court est toujours envisageable, et peut poser quelques problèmes. Des techniques de **propagation locale "réactive"**, complétées lorsque le système dispose de plus de temps, peuvent être envisagées. Ces solutions relèvent plus du domaine de la supervision d'exécution proprement dite, et nécessitent de se pencher plus sérieusement sur les méthodes réactives et les processus temps réel. Ces préoccupations sortent du cadre de notre étude, et seront donc laissées aux soins de travaux plus complets sur ces thèmes. Nous arrêtons donc là ce rapide, mais néanmoins nécessaire, coup d'oeil sur l'utilisation de notre modèle en cours d'exécution.

6. Extensions de la Représentation

Tout ce qui précède repose sur les hypothèses restrictives posées au paragraphe 3.1. Il serait néanmoins utile de chercher à lever ces hypothèses de manière à traiter des cas plus généraux. Nous avons commencé à réfléchir sur cette question, mais le travail reste embryonnaire et nécessiterait d'être affiné. Ce qui suit se veut donc l'ébauche informelle d'une étude des extensions possibles. Elle permet néanmoins de mettre à nouveau en avant des situations de **dépendance entre variables aléatoires**, qui ont déjà été abordées au paragraphe 4.5, dans le cadre d'un cas particulier. L'impossibilité a priori de traiter de tels cas constitue visiblement la limitation majeure dont souffre notre paradigme.

6.1. Prise en Compte de Contraintes Etiquetées Libres

Les seules CLBs prises en compte jusqu'à présent sont des précédences. Et si nous cherchions à étendre nos hypothèses aux cas de CLBs quelconques ? Ici, deux cas de figures doivent être distingués, selon qu'il s'agisse d'une contrainte de Déclenchement ou d'une contrainte d'Attente.

6.1.1. Ajout d'une contrainte d'Attente explicite

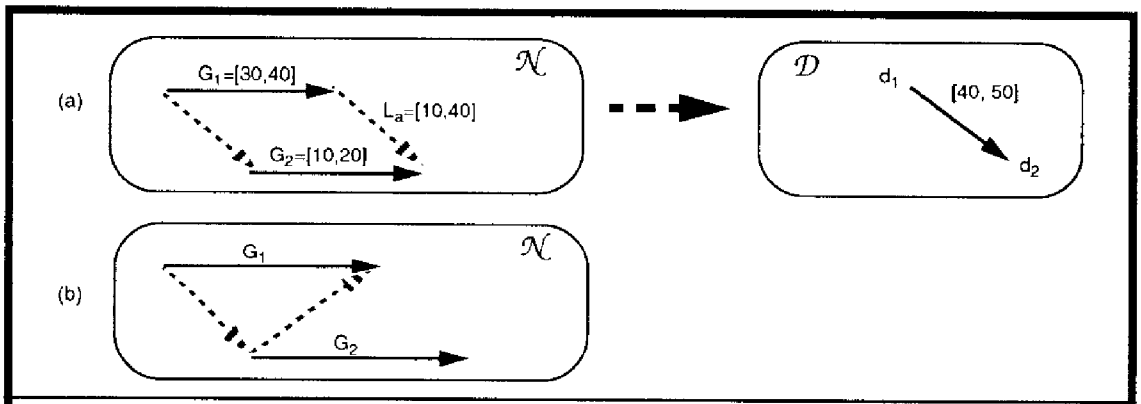


Figure 4.13. Prise en compte d'une contrainte d'Attente explicite.

Considérons les deux exemples de la figure ci-dessus (l'un des deux seulement étant affecté d'une valuation numérique). Ils représentent les deux seuls cas de figure possibles pour une contrainte d'Attente. L'intervalle associé [10, 40], que l'on suppose affecté en entrée à la contrainte d'Attente, signifie dans les deux cas d'un point de vue sémantique que l'on impose à

l'attente d'être au moins égale à 10 unités de temps, et de ne pas excéder 40 unités de temps. La question est alors: quelles décisions peuvent être affectées aux contraintes de Déclenchement du réseau pour que ceci soit vérifié ? C'est-à-dire que d'un point de vue sémantique toujours, cette contrainte d'Attente n'étant pas directement contrôlable, l'intervalle qui lui est associé est un intervalle de durées possibles, tel que l'on a l'habitude de les considérer dans le STPU $\mathcal{N}$.

En d'autres termes, dans la figure précédente, le premier cas (a) présente deux CTGs **indépendantes** (quant à la valeur qu'elles prendront) reliées par une contrainte numérique supplémentaire. Nous pouvons à partir de là calculer la contrainte $L_{d_1d_2}$ qui doit être contrôlable. Nous obtenons

- $inf_{d_1d_2} = inf_a + sup_1 - inf_2$
- $sup_{d_1d_2} = sup_a + inf_1 - sup_2$

ce qui nous donne comme condition de contrôlabilité:

$sup_a - inf_a \geq (sup_1 - inf_1) + (sup_2 - inf_2)$

Dans notre exemple, nous obtenons dans $\mathcal{D}$ une meta-contrainte numérique $MN_{1,2}$=[40, 50]. Dans le deuxième cas (b), la condition s'écrit tout simplement

$sup_a - inf_a \geq (sup_1 - inf_1)$

Ces résultats peuvent être formalisés quelque peu de la manière suivante:

Définition 4.15 : Le **degré de contingence** d'une CTG G_1 est la longueur de l'intervalle représentant le domaine de la contrainte:

$\Delta g_1 = sup_1 - inf_1$

Propriété 3.8 : Le degré de contingence d'une contrainte d'Attente est fonction des degrés de contingence des CTGs qu'elle relie. On distingue alors les deux cas de la figure précédente, pour obtenir:

(a) $\Delta g_a = \Delta g_1 + \Delta g_2$

(b) $\Delta g_a = \Delta g_1$

Nous voyons donc que l'ajout d'une contrainte d'Attente explicite ne pose aucun problème particulier. Elle devra simplement être **prise en compte dans le calcul des meta-contraintes** numériques, de la même manière que dans l'exemple présenté ci-dessus. De plus, dans le cas où cette contrainte est insérée dans le graphe en même temps que les CTGs qu'elle relie, nous pouvons aisément **vérifier a priori la contrôlabilité** d'un tel ajout: la longueur de l'intervalle de la contrainte ajoutée doit être supérieur à son degré de contingence, qui se déduit de la propriété ci-dessus.

6.1.2. Ajout d'une contrainte de Déclenchement explicite

D'un point de vue sémantique, restreindre les valeurs d'une contrainte de Déclenchement revient à imposer un ensemble réduit de décisions pouvant lui être effectivement affectées.

On peut tout de suite écarter un premier cas qui ne pose aucun problème, à savoir celui d'une contrainte de Déclenchement du type $L_{d_1 d_2}$. En effet, celle-ci sera directement placée dans le Graphe de Décision, après avoir vérifié sa contrôlabilité a priori.

Reste le cas d'une contrainte de Déclenchement du type $L_{f_1 d_2}$. Ici, calculer la meta-contrainte numérique $MN_{1,2}=L_{d_1 d_2}$ correspondante ne constitue pas une solution satisfaisante. En effet, considérons l'exemple suivant d'une relation R_4, où nous avons valué a priori la CLB $L_{f_1 d_2}$. D'un point de vue sémantique, restreindre les valeurs de cette contrainte signifie que l'on impose un ensemble réduit de valeurs pouvant lui être affectées.

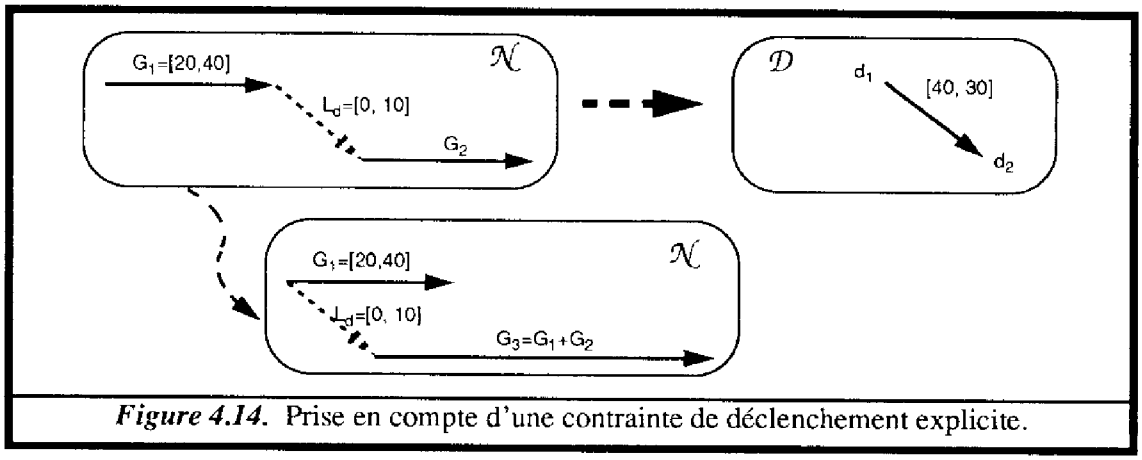


Figure 4.14. Prise en compte d'une contrainte de déclenchement explicite.

Si l'on cherche à calculer $L_{d_1 d_2}$, on obtient **une CLB incontrôlable** ! En effet, l'exemple signifie que l'on doit déclencher G_2 au plus tard 10 unités de temps après la fin de G_1, sur laquelle pèse un degré de contingence égal à 20. Autant dire qu'il nous est impossible de contrôler a priori l'écart entre d_1 et d_2. En d'autres termes, nous nous retrouverons dans une situation exactement similaire au cas particulier évoqué au paragraphe 4.5. Nous pouvons alors proposer le même type de transformation pour aboutir au réseau représenté également dans la figure ci-dessus. C'est-à-dire que nous sommes ramenés de nouveau à une situation faisant intervenir des **variables aléatoires interdépendantes**. La discussion qui peut être enclenchée à partir de là est alors la même que celle qui a prévalu pour le cas particulier précédemment évoqué. C'est-à-dire que nous risquons de **perdre la complétude** du modèle.

6.2. Prise en Compte de Contraintes d'Entrée Faibles

Pour ce qui concerne les contraintes faibles, nous allons devoir traiter successivement l'ajout d'une CLB faible et celui d'une CTG faible. Rappelons tout d'abord qu'une contrainte faible est une contrainte qui n'induit aucune précédence entre les deux instants concernés. Il n'est dès lors plus possible de distinguer aussi clairement que nous l'avons fait jusqu'ici des contraintes de Déclenchement et des contraintes d'Attente. Il va donc falloir distinguer, pour chaque type d'ajout (CLB ou CTG), les trois cas de figures suivants:

(1) une contrainte faible entre deux variables de Déclenchement,
(2) une contrainte faible entre une variable de Déclenchement et une variable d'Attente,
(3) une contrainte faible entre deux variables d'Attente.

6.2.1. Ajout d'une CLB faible

Nous ne détaillerons pas cette partie. Il est en effet aisé de constater que nous sommes ramenés à des cas de figures déjà exposés. Pour résumer, en considérant tour à tour chacun des cas ci-dessus énumérés:

(1) Il suffit de placer la contrainte [-10, 10] par exemple directement dans le Graphe de Décision.

(2) Ce cas est traité comme dans le cas d'une contrainte de Déclenchement (voir plus haut).

(3) Ce cas est traité comme dans le cas d'une contrainte d'Attente (voir plus haut).

6.2.2. Ajout d'une CTG faible

Ici, il convient de détailler quelque peu chacun des cas.

(1) Il est **impossible par définition** de disposer d'une CTG entre deux variables de Déclenchement.

♣♣♣

(2) Il est **impossible par construction** de disposer d'une CTG entre une variable de Déclenchement et une variable d'Attente.

En effet, si f_1 est une variable d'Attente, dont d_1 est la variable de Déclenchement associée (définissant ainsi implicitement une CTG $G_{d_1f_1}$), si d_2 est une variable de Déclenchement, et la contrainte entre f_1 et d_2 est une CTG ($G_{f_1d_2}$), alors $L_{d_1d_2}$ est contingente par addition de deux CTGs: $L_{d_1d_2} \equiv G_{d_1d_2} = G_{d_1f_1} \oplus G_{f_1d_2}$. Ce cas est donc également impossible.

♣♣♣

(3) Dans ce dernier cas, il est nécessaire de distinguer de nouveau deux sous-cas, que nous allons détailler ci-après.

Considérons que nous ajoutons une CTG faible entre f_1 et f_2. Si f_1 et f_2 sont déjà présents dans le réseau, alors par construction nous retrouvons de nouveau $L_{d_1d_2}$ contingente, ce qui exclut donc ce premier sous-cas. L'alternative consiste à disposer d'une CTG G_1 déjà présente, et à ajouter ensuite une CTG G_2 faible entre f_1 et un nouvel instant f_2 (comme l'illustre la figure ci-dessous). Ceci définit implicitement une CTG entre d_1 et f_2. Pour fixer les idées, on peut voir que d'un point de vue sémantique, cela revient par exemple à insérer dans le réseau deux événements f_1 et f_2 devant avoir lieu "*à peu près en même temps*".

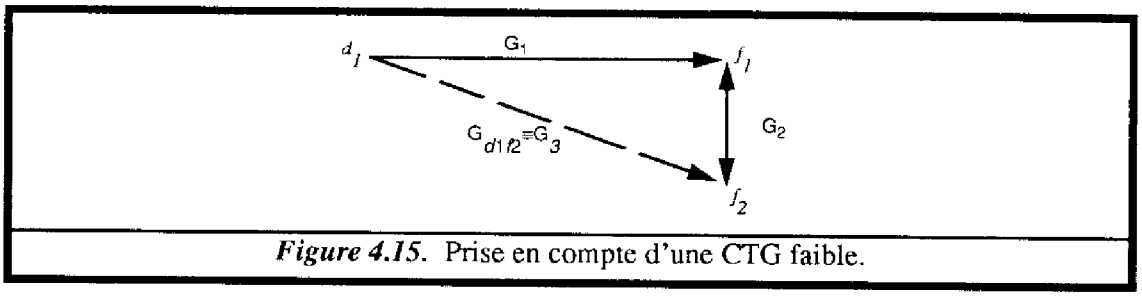


Figure 4.15. Prise en compte d'une CTG faible.

Là encore, il convient de procéder à une transformation du réseau, pour le rendre compatible avec notre modèle du Graphe de Décision. Nous transformons donc G_2 en une CTG $G_3 = G_1 \oplus G_2$ définie entre d_1 et f_2. Le problème est que l'observation de G_3 dépend de celle de G_1. C'est-à-dire que comme précédemment, nous obtenons une situation où coexistent deux **variables aléatoires interdépendantes**, conduisant aux mêmes conclusions que dans la section précédente.

6.3. Conclusion sur les Problèmes de Dépendance

Afin de conclure cette partie, nous pouvons dire pour résumer que notre paradigme de Graphe de Décision s'applique à la plupart des cas de figures rencontrés, si tant est que les deux conditions suivantes sont respectées:

1. Toutes les **CTGs** doivent être **indépendantes**.

2. Toutes les **variables de Déclenchement** doivent pouvoir être instanciées de manière **indépendante**. En effet, dans le cas contraire, on est obligé de se ramener à une situation de CTGs interdépendantes.

Lorsque ces conditions ne sont pas remplies, il est possible d'effectuer un changement de variable pour aboutir à une représentation compatible avec notre modèle, mais au prix d'une forme d'incomplétude. Celle-ci peut être acceptée dans la mesure où elle amène à rejeter des situations contrôlables, mais pas l'inverse, ce qui reste à prouver ! Ces considérations conduisent bien sûr à dégager des prospectives qui seront reprises au chapitre 6.

6.4. Prise en Compte d'Inégalités

Il resterait à étudier la prise en compte des inégalités, qui n'a pas été abordée pour l'instant. Nous nous contenterons ici de fournir quelques pistes, qui nécessiteraient un approfondissement, notamment en termes d'étude de complexité.

1. La prise en compte d'une inégalité dans $\mathcal{D}$ peut être faite de la même manière que dans $\mathcal{N}$. Comme nous l'avons vu au chapitre 3 (§ 8.1), cette prise en compte n'est pas explicite, mais la complétude des algorithmes de propagation est assurée grâce à l'opération d'agrégation au niveau symbolique, à chaque fois que l'on obtient une contrainte réduite au singleton $\{0\}$. Il y aura incohérence ssi on découvre une inégalité dans l'ensemble des instants que l'on est amené à agréger.

   Rappelons par ailleurs que notre méthode basée sur le Graphe de Décision impose implicitement des inégalités du type $(f_1 \neq d_2)$ et $(f_1 \neq f_2)$. Ces inégalités doivent donc être explicitement placées dans I pour pouvoir détecter une incohérence lors d'une opération d'agrégation. De même, nous avons vu que nous devions interdire l'apparition de cycles contenant deux extrémités d_1 et f_1 d'une même CTG G_1. Nous proposons de placer également explicitement des inégalités $(d_1 \neq f_1)$ pour chaque CTG ajoutée.

2. On détectera également de la même manière qu'au chapitre 3 (§ 8.2) une nouvelle inégalité dans $\mathcal{D}$, qui pourra être directement placée dans I.

7. Comparaison avec d'Autres Approches

La prise en compte de contraintes numériques aléatoires reste historiquement liée aux **méthodes PERT** [Kaufman69], où l'on représente un *programme* sous la forme d'un ensemble d'opérations affectées d'une durée aléatoire. La figure suivante illustre le fait que les réseaux PERT ne permettent de représenter que les contraintes suivantes: pour qu'une opération soit déclenchée, il faut que plusieurs opérations données soient terminées; et une fois qu'une opération est terminée, elle permet le déclenchement d'un certain nombre d'opérations. Dans notre paradigme, cela correspond à un ensemble de meta-contraintes symboliques R_4 (cf tableau 4.1), c'est-à-dire en fait des relations *before* de Allen. Comme nous l'avons vu, ces relations sont contrôlables par nature.

Cela nous donne une représentation ou les noeuds sont les opérations, et les contraintes sont de simples précédences. Par contre, les réseaux PERT ne permettent pas de manipuler des contraintes de synchronisation sur la fin d'une opération, et donc des contraintes du type *during* par exemple. La seule contrainte pour laquelle il est nécessaire de vérifier la cohérence de l'ensemble est de ce fait la durée totale du programme. Les réseaux PERT utilisent alors la méthode du chemin critique, c'est-à-dire que l'on cherche le chemin le plus contraint. Il suffit alors de vérifier dans quelle mesure la somme des durées aléatoires sur le chemin critique est inférieure à la durée totale imposée. On peut alors déterminer les marges disponibles entre deux opérations successives, c'est-à-dire le retard qui peut être toléré eu égard à la contrainte de durée totale. En résumé, les méthodes PERT considèrent exclusivement des durées explicites contingentes, et disposent d'une **richesse expressive très limitée** quant aux contraintes.

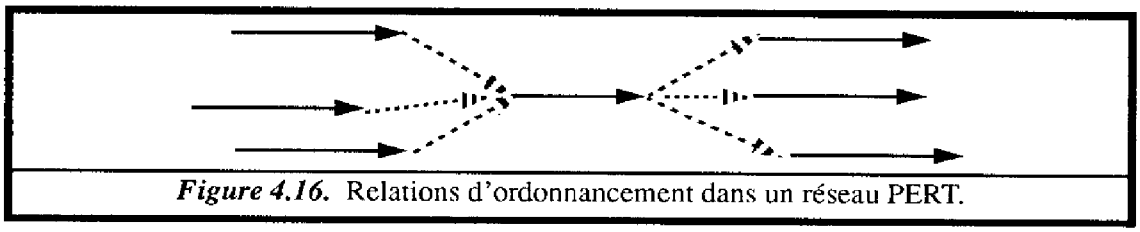


Figure 4.16. Relations d'ordonnancement dans un réseau PERT.

La **dichotomie stricte entre contraintes libres et contingentes** a été très peu abordée. Citons [Dorn94], qui introduit succinctement une telle distinction, mais qui, comme nous l'avons dit, se contente d'interdire la restriction d'une contrainte contingente, sans étudier le problème de la cohérence globale du réseau où cohabitent les deux types de contraintes. Nous avons vu au paragraphe 1.2 que cela n'était pas suffisant.

H. Fargier, D. Dubois et H. Prade [Dubois93, Fargier94] sont les premiers à notre connaissance à avoir défini clairement une partition entre des contraintes qu'ils nomment *contrôlables*

et d'autres qualifiées d'*incontrôlables*. Leur modélisation permet un **traitement de l'incertitude temporelle** plus satisfaisant que le nôtre dans le sens où ils utilisent la logique floue pour pouvoir déterminer le **degré de certitude** pesant sur le réseau global en fonction des incertitudes liées aux contraintes incontrôlables. En ce qui nous concerne, nous nous contentons pour l'instant de déterminer si le plan produit est sûr ou non. Nous reviendrons sur ces aspects dans le dernier chapitre du mémoire (chapitre 6, § 2.3). Leur modèle relève par ailleurs, comme le nôtre, de la famille des CSPs. Par contre, leur domaine d'application étant l'ordonnancement, ils n'ont à manipuler que des notions de **simples précédences** entre opérations à durée incontrôlable, comme dans les réseaux PERT. Ils se ramènent donc à des graphes ne contenant explicitement que les instants de début des opérations, avec une richesse expressive limitée comparativement à notre Graphe de Décision quant aux contraintes manipulées. Cela les conduit à développer des algorithmes plus simples que les nôtres en matière de vérification de cohérence, si ce n'est sur un point: ils gèrent la contrainte de disjonction entre deux opérations (relation [a ∨ b] d'Allen) au niveau d'un arbre de recherche, avec des heuristiques temporelles adaptées.

Partant de là, nous nous devons de citer le travail ultérieur de H. Fargier, J. Lang et T. Schiex [Fargier95], dont nous avons également déjà touché quelques mots. Il constitue une extension des concepts manipulés dans [Fargier94], dans le cadre le plus général des **CSPs discrets**. La partition en entrée porte par contre désormais sur les variables, qui sont contrôlables ou non (on parle dans ce dernier cas de *paramètres*). La contribution repose surtout sur les concepts de **cohérence** d'un tel CSP, et de **complexités** dans le cas général. Ils définissent notamment les concepts de *cohérence*, de *cohérence forte* et de *solution*, dont nous nous sommes inspirés lorsque nous avons développé les concepts de contrôlabilité d'un STPU.

8. Conclusion: Pouvoir Expressif Augmenté, Performances Maintenues

Nous avons présenté dans ce chapitre une technique permettant d'étendre les STPs à la prise en compte de contraintes contingentes (CTGs) dans le cadre du paradigme STPU. Nous avons montré que nous pouvions nous ramener à un STP équivalent ne contenant que les instants de "déclenchement" des CTGs du plan correspondant. Le **Graphe de Décision** est donc comparable aux réseaux issus de la philosophie PERT tels que ceux manipulés par Fargier, Dubois & Prade [Dubois93], qui ne gèrent cependant que des contraintes d'ordonnancement. Notre contribution consiste essentiellement à avoir montré qu'il était possible d'aborder des domaines d'application plus riches, tels que la planification. Il est en effet possible de disposer de toutes les contraintes (ou presque) admises par l'algèbre d'instants entre n'importe quel couple d'ins-

tants du graphe initial, et se ramener malgré tout à un graphe de *variables de Déclenchement* sans perte d'informations. La **richesse expressive** est ainsi sensiblement augmentée par rapport aux autres approches connues.

Par contre, cette extension de la représentation nécessite une étude plus poussée des concepts de cohérence de l'ensemble des contraintes d'entrée. Nous avons défini cette exigence sous le terme de **contrôlabilité** du réseau initial, et montré que cette contrôlabilité se ramenait à un simple concept classique de cohérence dans le Graphe de Décision. Cela nous permet donc d'utiliser toute la puissance des algorithmes de propagation des CSPs classiques. On peut voir notre problème comme un cas particulier des CSPs *mixtes* [Fargier95], dans lequel nous disposons de propriétés particulières nous permettant d'utiliser des **algorithmes polynômiaux**.

Néanmoins, nous avons dû poser un certain nombre d'hypothèses restrictives nous permettant de simplifier le problème initial et de le traiter de manière plus claire. La section traitant des extensions suggère que certaines de ces hypothèse pourraient être aisément levées. Ceci reste à préciser. La conclusion que nous pouvons cependant tirer de cette étude est que les problèmes surviennent dès lors qu'il devient nécessaire de manipuler des **durées aléatoires interdépendantes**.

Ce travail, loin d'être achevé, ouvre un grand nombre de **prospectives** que nous nous devons de développer quelque peu. Mais avant d'en arriver là, nous allons nous offrir une parenthèse, en nous écartant momentanément du domaine d'application de la planification, pour présenter une application d'ordonnancement et d'allocation de ressources. Cette application spécifique, qui a été développée à partir du gestionnaire de contraintes temporelles d'IxTeT, nous a également posé des problèmes intéressants en termes d'incertitudes temporelles numériques.

Chapitre V. Un Problème d'Ordonnancement en Présence d'Incertitudes Temporelles

Comme nous l'avons signalé en introduction, l'application qui va être présentée dans ce chapitre n'est en aucune manière une illustration des chapitres précédents. Elle s'écarte même par de nombreux aspects du domaine de la planification auquel IxTeT est dédié. L'objectif était d'essayer d'utiliser le pouvoir expressif et les fonctionnalités élémentaires du gestionnaire de contraintes temporelles dans le cadre d'une application spécifique se situant dans le domaine de l'ordonnancement et de l'allocation de ressources. Cette application présentait en effet des caractéristiques intéressantes eu égard aux thèmes abordés dans ce mémoire, à savoir la prédominance des informations temporelles numériques, celles-ci étant entachées d'incertitude, et la nécessité d'effectuer fréquemment des requêtes simples en direction du module temporel.

Bien que le développement de cette application ait occupé une part importante du travail mené durant la thèse, nous avons préféré mettre l'accent sur les aspects plus théoriques des deux chapitres précédents. C'est pourquoi nous nous contenterons de présenter les grand axes de notre méthode, sans entrer dans les détails de conception et d'implémentation. Nous nous focaliserons simplement sur les problèmes purement temporels qui nous intéressent plus particulièrement.

Néanmoins, comme il s'agit d'une application complète, nécessitant le développement de stratégies et d'heuristiques appropriées au niveau algorithmique, nous ne pouvions nous passer d'une rapide présentation préliminaire du domaine dans lequel nous nous situons.

1. Ordonnancement et Allocation de Ressources: un Bref Tour d'Horizon

La courte visite du domaine de l'ordonnancement que nous proposons ici s'inspire notamment de l'article de P. Prosser et I. Buchanan [Prosser94]. Le lecteur intéressé par une immersion plus approfondie pourra également se reporter à [French82, Steffen86].

1.1. Les Problèmes d'Ordonnancement en Intelligence Artificielle

La planification, comme nous l'avons vu, raisonne par rapport au but fixé et cherche à déterminer les tâches qui vont permettre de résoudre ce but. L'ordonnancement part au contraire d'un ensemble de tâches connues à l'avance, qu'il s'agit de positionner dans le temps les unes par rapport aux autres. On peut considérer que l'ordonnancement est un aspect particulier du

problème de planification. En fait, ces deux domaines sont sensiblement différents. En planification, la combinatoire provient plus du **choix de la tâche** elle-même que de son positionnement dans le plan partiel. En ordonnancement au contraire, la difficulté repose dans la manière dont on va **placer les tâches** les unes par rapport aux autres, en fonction de deux types essentiels de contraintes:

- Les contraintes de lien causal ou de compatibilité entre tâches (antériorité, synchronisation, non-parallélisme ...) se présentent directement sous la forme de **contraintes temporelles**.
- Les contraintes d'utilisation de ressources (en termes de capacité et de partage), qui seront par la suite évoquées simplement sous le terme de **contraintes de ressource**. Les choix effectués à ce niveau vont à terme se traduire également sous la forme de contraintes temporelles entre tâches.

En fait, planification et ordonnancement tendent actuellement à se rejoindre, le premier domaine s'enrichissant d'une réelle prise en compte des contraintes de ressource (comme nous l'avons vu avec IxTeT au chapitre 2, § 2.2.3) et des problèmes de choix d'ordonnancement qu'elles sous-tendent, le second quant à lui souhaitant désormais gérer des choix entre tâches équivalentes en certains points du programme produit [Laborie95b]. Retenons simplement que sur le problème d'ordonnancement temporel pur vient donc se greffer un problème d'allocation de ressources. Se posent alors les problèmes classiques suivants:

1. **existence** d'une solution (c'est-à-dire cohérence de l'ensemble des contraintes),
2. **détermination** d'une solution, c'est-à-dire un ordonnancement satisfaisant l'ensemble des contraintes évoquées ci-dessus.
3. **caractérisation** des solutions **admissibles**.

Il est clair que la résolution du premier point peut être menée au travers d'une résolution du deuxième ou encore du troisième point [Erschler91]. Notons encore que l'on cherchera généralement à **optimiser** cette solution, ce qui suppose donc la donnée de **critères** d'optimisation, influant sur la **qualité** de la solution produite.

Par ailleurs, quant au type de méthode employé pour déterminer une solution, deux approches opposées ont souvent été confrontées:

- L'approche **statique** (ou **prédictive**) utilise des méthodes d'optimisation combinatoire, à partir de données prévisionnelles et d'objectifs **globaux**. La solution trouvée peut être ensuite exécutée.
- L'approche **dynamique** (ou **réactive**, ou encore **opportuniste**) utilise des heuristiques de décision **locale** pour calculer une solution de manière incrémentale.

L'ordonnancement vu comme un problème d'optimisation est un problème NP-complet [Fox90] dans le cas général, ce qui rend l'approche statique difficile à utiliser dans les cas pratiques. En outre, l'approche dynamique permet de déterminer une solution au fur et à mesure qu'on l'exécute, autorisant ainsi des prises de décision en temps réel. Cela permet non seulement de réduire la combinatoire du problème initial, mais aussi de tenir compte des incertitudes. [Berry92] distingue à ce sujet

- l'incertitude liée à l'**ordonnancement**, c'est-à-dire l'absence de prédiction complète des conséquences d'un choix d'ordonnancement,
- l'incertitude liée à l'**exécution**, c'est-à-dire la prise en compte d'événements imprévus.

L'inconvénient des approches dynamiques est qu'elles ne permettent pas d'assurer l'optimalité de la solution construite. Mais on peut rétorquer, à l'image de [Berry92], qu'il n'existe pas de solution réellement optimale à un problème d'ordonnancement, du fait que les critères d'optimisation (temporels ou liés aux ressources) sont le plus souvent **contradictoires**. Notons pour finir que de nombreux systèmes combinent techniques prédictives et réactives, la première étape fournissant une vue globale de la solution, laquelle est raffinée au fur et à mesure de son exécution [LePape90, Wallace93]. Pour cela, on préférera le plus souvent à des techniques purement réactives des méthodes d'**ordonnancement glissant** [Collinot88, Dean86], où l'on maintient en cours d'exécution un ordonnancement à court-terme.

1.2. Principaux Systèmes et Techniques d'Ordonnancement

Historiquement, ces problèmes ont été initialement abordés par l'intermédiaire de techniques issues de la Recherche Opérationnelle, qui ont vite montré leurs limites en termes d'expressivité. L'Intelligence Artificielle s'est alors penchée sur le problème, renouvelant les techniques employées grâce à une représentation plus riche des connaissances du domaine.

[Steffen86] et [Prosser94] comparent les divers systèmes existant dans le domaine de l'I.A.. M.S. Fox est considéré comme le précurseur en la matière. Son premier système ***ISIS*** en 1983 se base sur une approche par contraintes pour modéliser le problème d'ordonnancement d'un atelier flexible. Il divise le problème par *lots* pour construire progressivement une solution. La faiblesse en matière de prise en compte des conflits de ressources est corrigée dans ***OPIS***, où un processus de raisonnement basé sur les ressources permet d'identifier les "goulets d'étranglement" dont la résolution est prioritaire. A partir de là, C.Le Pape et A.Collinot ont introduit avec ***SOJA***, en 1985, puis ***SONIA*** [Collinot88], en 1988, la prise en compte **explicite** de contraintes temporelles, pour lesquelles des techniques de propagation sont utilisées, et l'utilisation combinée d'**allocation prédictive** des ressource et de **détermination dynamique des**

dates de déclenchement de chaque tâche, pouvant conduire à des **retours-arrière** sur la phase d'allocation prédictive.

Les systèmes ultérieurs ont cherché à limiter la combinatoire du problème par l'intermédiaire de techniques distribuées. Nous n'entrerons pas dans les détails, notre méthode se situant en-dehors de telles approches. Donnons juste quelques pointeurs sur les problèmes que ces systèmes ont eu à traiter:

- Définir l'**architecture**: il s'agit de distribuer le problème à divers agents qui communiquent entre eux. L'architecture la plus couramment utilisée est une architecture **hiérarchique**, qui peut être définie en fonction des ressources, comme dans ***DAS***, ou en fonction d'un niveau de granularité temporelle, comme dans ***DISA*** [Berry92].
- Rechercher la **robustesse**: dans le système ***REDS*** par exemple, on part du principe que la distribution du problème permet d'effectuer la recherche selon plusieurs critères d'optimalité conflictuels en parallèle, de manière à aboutir à un "bon compromis".
- La place de l'**utilisateur**: de plus en plus de systèmes choisissent de placer l'utilisateur dans la boucle, comme *DISA* par exemple où l'utilisateur est chargé de résoudre les conflits pour lesquels le système n'a pas trouvé de solution satisfaisante relativement aux critères de départ. Le système ***ESCALAS*** propose quant à lui plusieurs modes de fonctionnement, du "manuel" à l'"automatique".

Signalons pour finir les approches les plus récentes, qui choisissent de partir d'une solution déjà construite pour chercher à l'optimiser en effectuant des opérations de **"réparation" locale**. Il s'agit d'une part des **algorithmes génétiques**, issus de la communauté I.A., et des techniques de **recuit simulé** et de **"recherche tabou"**, qui proviennent plutôt de la communauté R.O. Ces techniques fournissent généralement de bien meilleures performances. Par contre, elles ne fournissent de solutions réellement satisfaisantes (en termes de complétude et d'optimalité) que dans la mesure où l'ordonnancement de départ est déjà **suffisamment proche de l'optimum** [Prosser94]. Ces techniques sont encore relativement récentes, mais semblent promises à un bel avenir. Nous n'en parlerons pas d'avantage du fait qu'elles se situent largement au-delà du cadre strict de notre étude.

1.3. Stratégies: Recherche d'une Solution et Optimalité

Intéressons-nous plus spécifiquement au problème de recherche d'une solution dans un système centralisé, où cohabitent des contraintes temporelles et des contraintes d'utilisation de ressources. Ce problème peut être vu comme un problème de **décision sous contraintes**. Dans [Berry92] notamment, on peut voir que l'approche de résolution se ramène de ce fait au fonctionnement classique des **CSPs**, à savoir deux étapes successives de décision, qui sont:

1. Le choix de la **variable** que l'on va instancier, c'est-à-dire une **tâche** spécifique.

2. Le choix de la **valeur** que l'on va lui attribuer, à savoir, selon que l'on raisonne par rapport à un critère ou bien l'autre,
   - la **ressource** qui lui est allouée,
   - sa **date de déclenchement**, sous la forme d'un couple {*date_au_plus_tôt, date_au_plus_tard*}, c'est-à-dire d'un intervalle de dates possibles.

Ces deux étapes sont répétées jusqu'à l'obtention d'une solution. En cas de conflit, on effectue un retour-arrière sur des choix antérieurs. S'il n'existe plus de possibilité de retour-arrière, on en déduit qu'il n'existe pas de solution. Afin de limiter ces cas de retours-arrière, on peut utiliser des **heuristiques** de choix (cf chapitre 1) pour guider la recherche. On parle alors de techniques d'**anticipation** (*look-ahead*) [Sadeh91], pour lesquelles on cherche à évaluer par avance les conséquences d'un choix pour optimiser celui-ci. Le type d'heuristique le plus couramment utilisé consiste à choisir en premier les variables les plus critiques, c'est-à-dire celles qui sont susceptibles d'être les plus difficiles à instancier (heuristiques *fail-first*), dans l'esprit de la recherche des goulets d'étranglement dans *OPIS* (voir au paragraphe précédent).

Citons à ce titre la technique originale développée dans le cadre de *DISA* par B. Choueiry [Choueiry94] pour l'allocation des ressources aux tâches. Cette technique, dénommée *Variable Assignment Delay Heuristic*, permet notamment d'isoler l'origine d'un conflit en cas d'échec, lequel est signalé à l'utilisateur qui dispose alors d'informations pertinentes pour prendre une décision, par exemple la relaxation d'une contrainte.

L'utilisation de techniques de type CSP fournit donc, outre la possibilité de relaxer certaines contraintes en cas de problème surcontraint, des techniques de propagation classiques dans des réseaux de contraintes. Cela permet de mettre à jour l'ensemble des données en fonction d'une décision prise, et détecter par la même occasion une incohérence au plus tôt.

Signalons pour finir les travaux menés dans le cadre de la résolution des problèmes "**réellement difficiles**" [Prosser94]. En effet, il apparaît que les problèmes sous-contraints, tout comme les problèmes sur-contraints, sont relativement faciles à résoudre. Les problèmes les plus difficiles se situent à mi-chemin. On propose pour ceux-là des techniques de recherche **coopérative** ou encore de recherche **en temps borné** (*anytime*), qui déterminent une solution dont la qualité sera fonction du temps passé à la déterminer.

2. Un Problème d'Allocation Dynamique de Tâches à un Ensemble de Robots

2.1. Contexte d'Application: le Projet MARTHA

Le groupe Robotique et Intelligence Artificielle du LAAS est impliqué dans un projet ESPRIT III dénommé **MARTHA** (*Mobile Autonomous Robots for Transportation and Handling Applications*). Il s'agit de robotiser les tâches de chargement et de déchargement de containers sur le port de Rotterdam. Des bateaux et des trains (nous utiliserons le terme générique de convoyeur) arrivent et repartent à des dates plus ou moins bien connues. Ils doivent être déchargés de leur contenu, à savoir des containers, qui sont accessibles selon un ordre donné. Chaque container a une destination connue: un bateau ou un train en partance, ou bien une zone de stockage. L'ensemble de ces spécifications constituent la **mission** à conduire. L'objectif est de faire exécuter les diverses tâches de la mission par une **armada de robots** (une cinquantaine a priori).

Le spécifications du projet global sont les suivantes. On dispose d'une station centrale qui envoie leurs buts respectifs aux robots, après quoi la planification et l'exécution des mouvements des robots sont gérées de manière distribuée. Chaque robot détermine son plan local de navigation qu'il insère dans un plan global. Celui-ci est maintenu implicitement par un maintien de plans locaux, obligeant les robots à mener des actions de coopération en communiquant entre eux, par exemple pour la traversée de carrefours, ou pour emprunter un itinéraire sur lequel deux robots ne peuvent se croiser. On cherche à obtenir de cette manière une situation de trafic optimal en cours d'exécution. Tout cela conduit au paradigme général dénommé "*Opération de Fusion de Plans*".

Il s'agit donc d'un problème d'**affectation de missions** au niveau le plus haut (quel robot choisir pour quelle tâche ?) couplé avec une **méthode distribuée multi-robots** pour la réalisation de ces missions. C'est sur ces derniers aspects qu'a pesé la contribution du groupe dans le projet MARTHA. Le lecteur intéressé par ces techniques de navigation multi-robots pourra se reporter à l'article [Aguilar95].

2.2. Description de l'Application

En marge du projet européen, il nous a paru intéressant de regarder ce qu'il était possible de faire au niveau de l'**allocation centralisée de tâches aux robots**, donc à un plus haut niveau

d'abstraction. On ne s'intéresse pas aux problèmes de trafic, ce qui signifie que les contraintes spatiales de l'application se limitent à un ensemble de zones prédéfinies, toutes connectées les unes aux autres par des routes prédéfinies. Ces contraintes spatiales se ramènent donc à des contraintes temporelles, puisqu'une action de déplacement d'une zone à une autre est simplement modélisée par sa **durée** connue de manière **imprécise** (de manière à tenir compte a priori de tous les aléas de la navigation). Les figures ci-dessous illustrent schématiquement le cadre d'application décrit au paragraphe précédent, ainsi que la représentation de l'environnement sous la forme d'un ensemble de zones. Il s'agit d'un "instantané" pris à un moment donné de l'exécution. Les flèches en grisé représentent les déplacements déjà effectués par les robot, et les flèches noires les déplacements à venir.

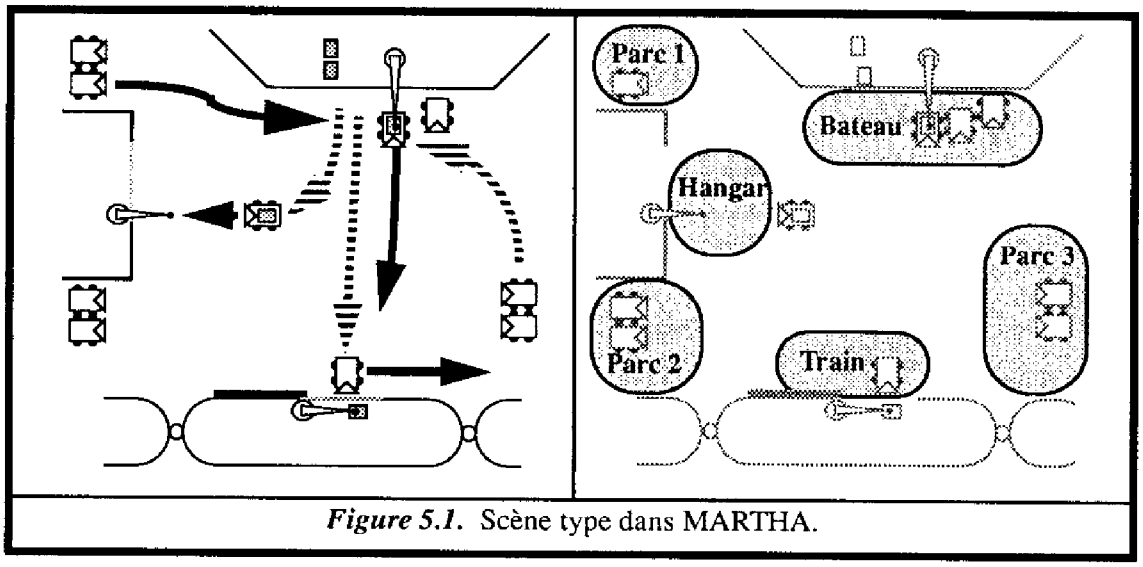


Figure 5.1. Scène type dans MARTHA.

Une tâche va se décomposer en un ensemble de quatre actions élémentaires, qui sont

1. GOTO (*?robot*, *?zone-courante*, *?zone-déchargement*),
2. PICKUP (*?robot*, *?container*, *?zone-déchargement*),
3. GOTO (*?robot*, *?zone-déchargement*, *?zone-chargement*),
4. PUTDOWN (*?robot*, *?container*, *?zone-chargement*).

Par exemple, si un container $Cont_X$ doit être transporté du bateau *Bateau* vers le train *Train*, nous aurons alors à nous préoccuper d'une tâche composée des quatre actions partiellement instanciées suivantes:

1. GOTO (*?robot*, *?zone-courante*, ***Bateau***),
2. PICKUP (*?robot*, $\boldsymbol{Cont_X}$, ***Bateau***),
3. GOTO (*?robot*, ***Bateau***, ***Train***),
4. PUTDOWN (*?robot*, $\boldsymbol{Cont_X}$, ***Train***).

Il sera alors nécessaire de choisir un robot, par exemple $Robot_Y$, pour se charger de cette tâche. Seront alors automatiquement instanciées les variables restantes *?robot* (par $Robot_Y$), et *?zone-courante* (par la zone où se trouve actuellement le robot $Robot_Y$).

Signalons que notre haut niveau d'abstraction nous amène à assimiler un convoyeur à la zone sur laquelle il se trouve. La même variable désignera donc l'un et l'autre. A chaque action sera associée une durée imprécise, sous la forme d'un intervalle de durées possibles [*inf, sup*]. Cette durée variera en fonction des paramètres de l'action (une action GOTO par exemple sera affectée d'une durée qui dépend des deux zones qui doivent être reliées). Signalons au sujet des actions PICKUP et PUTDOWN deux choses essentielles:

- Lorsque le paramètre de zone correspond à un convoyeur, l'action PICKUP [resp. PUTDOWN] doit avoir lieu pendant le délai de présence du convoyeur, donc entre son instant d'arrivée et son instant de départ. Ceci constitue une **fenêtre temporelle** initiale dans laquelle l'action doit se dérouler. Cette fenêtre temporelle s'exprime sous la forme d'un couple d'instants (F_i, F_s) associé à un PICKUP par exemple tel que $F_i \leq$ *début(PICKUP)* et *fin(PICKUP)* $\leq F_s$.

- Sur une même zone, les actions de PICKUP et de PUTDOWN utilisent une ressource non partageable qui est la **grue** utilisée pour effectuer ces actions. Ceci induit donc une contrainte d'exclusion mutuelle entre les PICKUP et PUTDOWN d'une zone donnée. Cela va nous conduire à ordonner les actions de déchargement et de chargement les unes par rapport aux autres, par l'intermédiaire d'ajouts de simples précédences, qui vont induire une mise à jour des fenêtres temporelles (F_i, F_s) (voir plus loin au § 3.3).

Les données initiales du problème sont les suivantes:

- Un ensemble de zones prédéfinies (convoyeurs compris) $\mathcal{Z} = \{Bateau, Train, ...\}$,
- Un ensemble de ρ robots $\mathcal{R} = \{Robot_1, ..., Robot_\rho\}$, tous équivalents, et leurs positions initiales,
- Un ensemble de γ containers numérotés $\mathcal{C} = \{Cont_1, ..., Cont_\gamma\}$, et leurs zones de déstockage. Pour chaque zone, le sous-ensemble de containers correspondant étant doté d'un ordre partiel d'accessibilité, nous adopterons une description sous la forme de "piles" virtuelles.
- Une destination pour chaque container,
- Les dates imprécises d'arrivée et de départ des bateaux et des trains,
- Les durées imprécises de chaque action en fonction des paramètres,

Par contre, nous ne nous intéresserons pas aux problème de croisement, d'embouteillage ou de files d'attente, chaque zone étant supposée de capacité suffisante pour gérer localement ces contraintes. Signalons pour terminer quelques spécifications et précisions supplémentaires:

- Il y a nettement moins de robots que de containers à déplacer (les spécifications de départ donnent 50 robots pour plusieurs centaines de containers par mission quotidienne), ce qui signifie qu'un robot devra se charger successivement de plusieurs tâches de transport de containers.

- Les containers sont ordonnés (sous forme de piles) dans leurs zones de déchargement. On n'impose par contre pas d'empilement prédéfini au niveau de leur destination.

- Les instants correspondant à l'arrivée et au départ d'un convoyeur définissent implicitement un intervalle de temps sur lequel une proposition IN_AREA (*?zone-convoyeur*) est vraie.

- Nous considérerons dans un premier temps pour simplifier le problème que sur une zone donnée, un seul type d'action pourra être mené, soit des déchargements, soit des chargements. Ceci nous permet de ne pas avoir à considérer les cas de PICKUP et de PUTDOWN devant s'entremêler sur une même zone. Ces cas sont traités néanmoins dans [Vidal95], et seront succinctement évoqués dans ce mémoire à la fin de ce chapitre.

2.3. Représentation Temporelle de Base

Tout ce qui précède nous amène à adopter une représentation temporelle de type STP [Dechter91]. Nous utiliserons donc une partie du gestionnaire de contraintes temporelles du planificateur IxTeT (cf chapitre 2), à savoir la structure du **graphe d'instants** disposant d'un instant initial 0, et les algorithmes élémentaires de gestion aussi bien des **contraintes symboliques** que des **contraintes numériques**. A chaque container, dont on connaît la zone de déchargement et la zone de destination, est associé implicitement une tâche prédéfinie, à laquelle est associée un graphe élémentaire que nous appellerons **composante-tâche** (voir plus loin au § 4.2), et dont la représentation est donnée dans la figure suivante.

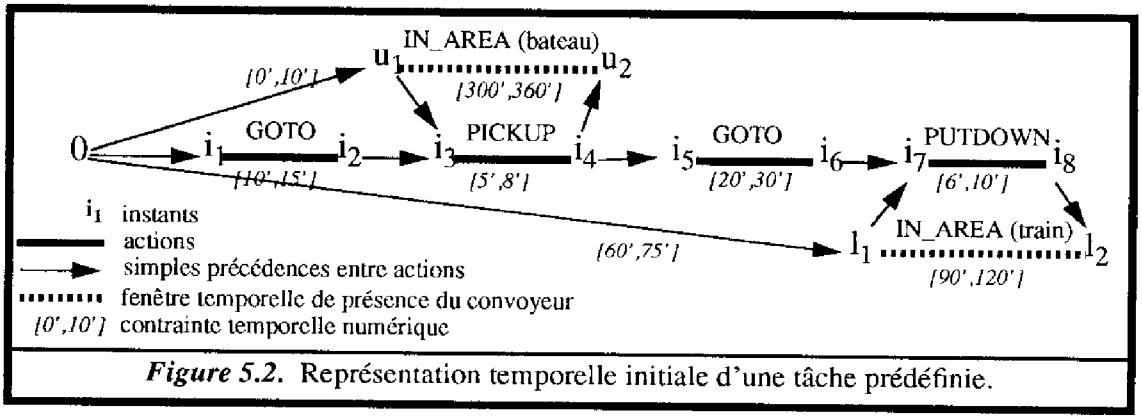


Figure 5.2. Représentation temporelle initiale d'une tâche prédéfinie.

A titre d'illustration, nous avons fixé des valuations numériques et des zones de déchargement et de chargement quelconques. Ces tâches peuvent être générées pour chaque container. On constate alors que dans chaque tâche ainsi développée, il reste à instancier le robot devant l'effectuer et le paramètre *?zone-courante* du premier GOTO (voir paragraphe précédent). On peut remarquer que ce choix va fixer la durée imprécise associée à ce GOTO, laquelle est indéterminée au départ. Le problème comme nous allons le voir se résume donc au **choix d'un robot pour chaque tâche**.

Pour revenir à la représentation temporelle, on voit donc qu'il est possible de générer dans une phase de **prétraitement** un graphe constitué des composantes-tâches de l'ensemble des containers, positionnées a priori en parallèle. Le choix d'un robot pour chaque tâche constitue alors une opération d'**allocation de ressources**, qui va entraîner l'**ajout de contraintes temporelles** dans le graphe, conduisant ainsi progressivement à un **ordonnancement partiel** qui constituera la **solution** à notre problème. Mais avant de préciser tout cela, donnons-nous un petit exemple illustratif.

2.4. Un Petit Exemple Illustratif

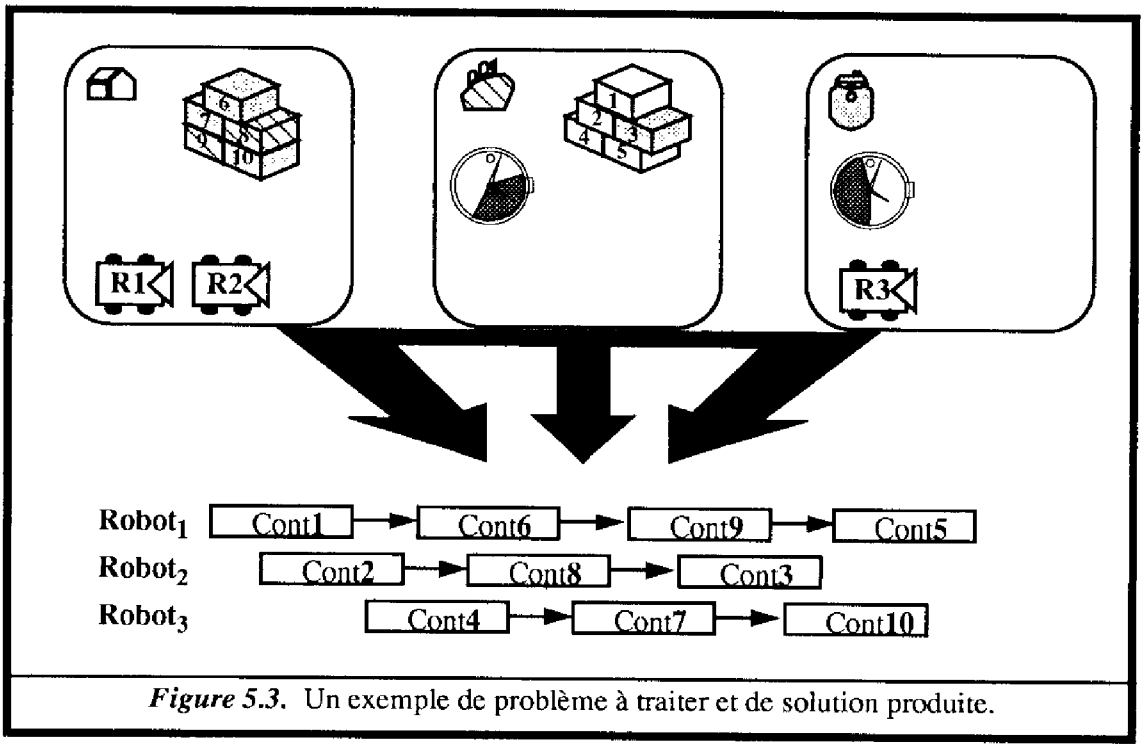


Figure 5.3. Un exemple de problème à traiter et de solution produite.

Ramenons-nous simplement à trois zones *Bateau*, *Train* et *Hangar*, et trois robots $Robot_1$, $Robot_2$ et $Robot_3$. La figure ci-dessus représente ces trois zones avec les robots situés dans leurs positions initiales. Les périodes de présences attendues du bateau et du train sont schématisées par l'intermédiaire de zones noircies sur des horloges. Par exemple, le bateau arrive dans dix minutes et repart dans tente-cinq minutes. Le lecteur ne doit pas oublier que ces contraintes sont entachées d'imprécisions, lesquelles n'ont pas été représentées pour ne pas surcharger la figure. Nous avons également dessiné les "piles" de containers sur le bateau et dans le hangar, les niveaux de gris indiquant la destination des containers, par exemple le train pour les containers $Cont_6$, $Cont_7$ et $Cont_{10}$ situés dans le hangar et $Cont_3$ situé sur le bateau.

A partir de là, il va s'agir de distribuer les tâches de transport de container aux robots, en cherchant à répartir ces tâches le mieux possible, tout en évitant les temps morts, ceci de manière à obtenir une solution qui soit la plus susceptible d'être exécutée dans les délais imposés. La figure illustre la solution obtenue selon ces critères dans le cas de l'exemple.

2.5. Caractérisation du Problème et Vision d'Ensemble de la Méthode

Nous disposons en fait d'un ensemble prédéfini de tâches, pour lesquelles plusieurs types d'opérations doivent être menées:

1. Choisir un robot pour effectuer la tâche.
   Il s'agit en fait d'un problème d'allocation de ressources, la ressource étant ici le robot. C'est pourquoi il est nécessaire d'affiner notre formulation en fonction du type de problème ainsi identifié: il ne s'agit pas d'allouer une tâche à un robot, mais au contraire d'**allouer un robot (la ressource) à une tâche prédéfinie**.

2. Ordonner les tâches successives pour un même robot.
   C'est-à-dire que dans le cas où on réalloue un robot à une nouvelle tâche, il faut situer cette tâche relativement aux précédentes, par un simple ajout de contraintes temporelles. Ces ordonnancements successifs seront faits, comme nous allons le voir, de manière chronologique, du fait qu'il ne nous est pas possible d'envisager de retours-arrière sur les choix effectués.

3. Ordonner les actions PICKUP [resp. PUTDOWN] sur une même zone.
   C'est-à-dire que la solution finale doit fournir un ordonnancement complet de ces actions sur chaque zone.

On voit donc qu'il s'agit là d'un problème **conjoint** [LePape94] d'ordonnancement et d'allocation de ressources. Remarquons que le problème d'allocation de ressources (opération 1 ci-dessus) est a priori assez simple, comparativement par exemple aux problèmes abordés dans [Berry92], puisqu'ici toutes les ressources, non partageables par définition, sont équivalentes et peuvent être identiquement allouées à n'importe quelle tâche. C'est cette allocation de ressources qui induit l'ordonnancement par ajout de contraintes temporelles dans un graphe d'instants géré séparément (opération 2 ci-dessus), ce qui peut être comparé aux approches de [Boddy93], qui utilise le TMM de Dean et McDermott (cf chapitre 1, § 4.4) et de [Erschler91], qui utilise des graphes "potentiels-bornes" qui sont similaires aux STPs.

En plus de la ressource *robot*, il convient également de considérer le cas de la grue, ressource non partageable par définition. Comme nous l'avons suggéré plus haut, cette ressource n'a pas besoin d'être gérée explicitement. On se ramène en fait à un problème d'ordonnancement complet des actions de PICKUP et PUTDOWN sur une même zone (opération 3 ci-dessus). Ce problème va s'avérer plus épineux qu'il n'y paraît. Il va s'agir d'adopter une stratégie d'**engagement minimal**, c'est-à-dire ne pas forcer un ordonnancement a priori tant que cela n'est pas nécessaire, de manière à limiter les retours-arrière.

♣♣♣

Ensuite se pose le problème de la **propagation des contraintes temporelles** à l'ensemble du graphe. La question qui se pose est alors la suivante: quel type de propagation, pour quel type de complexité ? Enfin, signalons en guise d'anticipation sur ce qui va suivre que le choix du robot notamment va se poser en termes de critères purement temporels. L'**imprécision des contraintes temporelles numériques** va alors jouer un rôle clé. Au fur et à mesure que le nombre de tâches associées à un robot augmente, les imprécisions temporelles augmentent également en proportion. Il arrive un moment où il n'est plus possible d'effectuer un choix dans lequel on puisse avoir suffisamment confiance, c'est-à-dire que ce choix a des chances non négligeables d'être sous-optimal au moment de l'exécution.

C'est pourquoi, comme nous allons le voir, nous avons choisi d'adopter une méthode d'ordonnancement en "**horizon glissant**" (cf § 1.1), c'est-à-dire que nous réallouons les robots à de nouvelles tâches au fur et à mesure de l'exécution de celles-ci. Ceci permet, comme nous l'avons vu au premier paragraphe, de déterminer une solution qui tienne compte des **incertitudes**. Dans notre application, seules les incertitudes d'ordonnancement sont prises en comptes. Il s'agit d'incertitudes temporelles pesant sur les durées des actions (qui sont contingentes par définition, si l'on se réfère aux définitions du chapitre 4). Ces incertitudes sont levées au fur et à mesure de l'exécution de la mission en cours. Ce choix amène à définir une **architecture à trois niveaux**:

1. Une **boucle de décision** qui alloue de manière itérative un robot à chaque tâche, avec des heuristiques qu'il va falloir définir,

2. Un **module de gestion du graphe temporel**, dont il va falloir définir les processus de requête et de mise-à-jour.

3. un **module d'exécution** des tâches, qui gère à la fois le déclenchement des actions et la réception des événements correspondant à la fin de celles-ci, ou aux arrivées et départ des convoyeurs.

Cette méthode, pour laquelle allocation de ressources, ordonnancement et exécution doivent avoir lieu en parallèle, impose de fait des exigences accrues en matière d'efficacité, c'est-à-dire que

- D'une part, la phase d'allocation de ressources devra pouvoir être menée **sans retours arrière**, aussi bien sur les choix d'allocation de robots que sur les choix d'ordonnancement des PICKUP et des PUTDOWN. Ceci justifie donc la remarque faite au début de ce paragraphe sur l'allocation chronologique des tâches pour un même robot.

- D'autre part, la propagation des contraintes temporelles devient un enjeu décisif, dont les performances risquent fort de déterminer l'**efficacité** du système global.

Nous allons étudier dans la suite du chapitre tour à tour chacun des trois modules que nous venons de mettre en avant.

3. La Boucle de Décision

Cette section a pour objet de définir la boucle d'allocation itérative de robots aux tâches prédéfinies de transport de containers. Elle ne sera présentée que de manière assez superficielle, ne représentant pas une contribution majeure dans le cadre de cette application. Nous renvoyons le lecteur souhaitant avoir des précisions supplémentaires à un article plus fourni sur ce thème [Vidal95]. A chaque étape de la boucle, nous disposons d'une situation définie par

1. Le graphe courant, dans lequel un certain nombre de tâches sont instanciées et ordonnées du fait de l'allocation d'un robot, et d'autres tâches restent non instanciées et non ordonnées.

2. La position courante de chaque robot et son **instant de disponibilité**: il s'agit

- soit de l'instant initial 0 si le robot n'a pas encore été alloué,
- soit de l'instant correspondant à la fin de la dernière tâche à laquelle il a été alloué (c'est-à-dire la fin du dernier PUTDOWN qu'il a effectué).

3. Les piles courantes de containers dans chaque zone, pour lesquelles on doit pouvoir identifier ceux qui sont en sommet de pile, que nous appellerons les **containers accessibles**.

Remarquons que le point 3 représente implicitement l'ensemble des **tâches devant être instanciées**. C'est pourquoi nous confondrons par la suite le container et la tâche de transport de celui-ci, en parlant selon les cas d'instanciation de la tâche, ou bien, par abus de langage, d'instanciation du container. Le point 2 représente quant à lui la **disponibilité des ressources** pouvant être alloués. On a donc là un problème classique d'allocation de ressources, nécessitant les deux étapes successives présentées au paragraphe 1.1:

- Le choix de la **variable**: il va s'agir ici de choisir la tâche à instancier, c'est-à-dire le **container** à transporter.
- Le choix de la **valeur**: il s'agit tout simplement du **robot** à allouer au container.

Nous allons définir les heuristiques choisies pour effectuer ces choix. Il va s'agir ensuite d'étudier les opérations d'ordonnancement temporel qui en découlent, aussi bien sur les tâches pour un robot donné, que sur les actions de PICKUP et PUTDOWN sur une zone donnée. Mais avant cela, arrêtons-nous un instant sur la gestion des piles de containers.

3.1. Représentation et Gestion des Piles de Containers

Une pile de containers correspond à un **ordre partiel** de saisie de ces containers. Ceci nous a suggéré l'idée d'utiliser, en la simplifiant, la structure symbolique du graphe temporel d'IxTeT pour gérer les piles de containers. Les piles de containers de l'exemple illustratif peuvent par exemple être manipulées sous la forme du graphe symbolique suivant, muni d'un point initial 0 ajouté de manière artificielle.

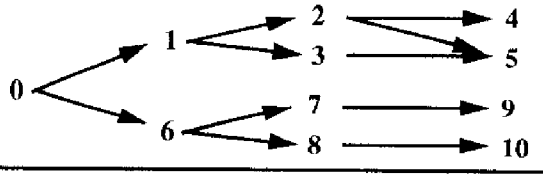


Figure 5.4. Représentation graphique des piles de containers de l'exemple.

Nous pouvons dès lors utiliser des algorithmes symboliques simplifiés pour

- déterminer les containers **accessible**: il s'agit des points du graphe de rang 1, c'est-à-dire les successeurs directs du point 0.
- **mettre à jour** facilement et efficacement (en temps linéaire par rapport au nombre de containers γ) ces structures à chaque fois que l'on a "instancié" un container, celui-ci étant alors **retiré** du graphe.

Cette technique, aisée à mettre en place à partir des structures et algorithmes temporels symboliques présents dans IxTeT, nous permet de maintenir de manière simple et efficace l'ensemble des containers accessibles sur l'ensemble des zones. C'est parmi ceux-ci qu'il va s'agir de choisir le prochain à instancier.

3.2. Stratégies de Choix

Les heuristiques présentées ci-après ont pour but d'aider à la détermination sans retour-arrière d'une solution "optimale". Nous avons choisi de privilégier un **critère temporel d'optimalité**, comme dans [LePape90]: nous cherchons à minimiser le risque de ne pas avoir déchargé [resp, chargé] tous les containers dans une zone donnée avant le départ du convoyeur concerné. Ceci nous a conduit à choisir les heuristiques qui vont être évoquées succinctement ci-après.

Mais au préalable, il nous a fallu mettre en place un algorithme de **vérification partielle de cohérence a priori**. Nous nous contentons de vérifier si la somme des durées des actions de déchargement [resp. chargement] dans une zone donnée est nécessairement inférieure à la durée de stationnement du convoyeur dans la zone. Si tel n'est pas le cas, alors la mission n'est pas réalisable. Par contre, si la réponse est positive, cela n'induit pas nécessairement la réussite de la mission. En effet, les actions de déchargement ou chargement ne vont pas nécessairement se succéder immédiatement les unes aux autres. Des délais d'acheminement d'un robot sur la zone peuvent induire une période d'inactivité de la grue mettant en danger l'achèvement de la mission avant le départ du convoyeur. Notre processus n'est donc qu'un prétraitement visant à vérifier simplement la **faisabilité** de la mission (c'est-à-dire la possibilité, et non la certitude, qu'elle réussisse).

3.2.1. Choix du Container

En fonction de ce qui a été dit au-dessus, nous nous devons de choisir le container **le plus urgent**, dans une stratégie de type *fail-first* (cf 1.1). Celui-ci se détermine de la manière sui-

vante. Le container le plus urgent est celui pour lequel nous disposons d'un laps de temps minimal pour le décharger de sa zone. Nous allons donc passer en revue tous les PICKUP des containers accessibles. Ceux-ci sont contraints par leurs fenêtres respectives. Celui dont la date au plus tard est minimale fournit le container prioritaire.

A partir de là, il est clair que deux containers déchargés sur une même zone sont équivalents. Nous définissons alors, afin de départager les éventuels ex-aequo, un deuxième critère concernant les zones de déchargement, donc les actions de PUTDOWN. Il va s'agir cette fois-ci, au contraire, de choisir le container pour lequel on n'aura pas à attendre dans la zone de déchargement, de manière à ne pas immobiliser inutilement un robot. Il s'agit donc de minimiser la date au plus tôt de début du PUTDOWN.

Ces heuristiques sont on ne peut plus simples, mais se sont avérées largement suffisantes pour fournir des solutions satisfaisantes en termes d'optimalité dans les cas de figures que nous avons eus à traiter (voir les résultats expérimentaux à la fin de ce chapitre). La structure **modulaire** de notre architecture nous permet de les modifier à volonté, autorisant ainsi des améliorations ultérieures dans le but d'aborder des applications plus complexes, ou des critères d'optimalité affinés.

En termes de **complexité**, cette phase est simplement **linéaire** par rapport au nombre de containers accessibles. Plus la structure d'empilement des containers sera importante, plus l'heuristique sera donc efficace.

3.2.2. Choix du Robot

Le choix du robot le plus approprié est immédiat. Il s'agit du robot **le plus prompt**, c'est-à-dire celui qui sera le plus rapidement sur la zone de déchargement. Il suffit donc de minimiser pour l'ensemble des robots la date d'arrivée estimée sur la zone, qui est la somme de la date de l'instant de disponibilité du robot et de la durée de l'action de déplacement l'amenant de la zone où il se trouve vers la zone de déchargement. La **complexité** est ici **linéaire** par rapport au nombre de robots.

3.3. Stratégies d'Ordonnancement

Une fois que l'étape d'allocation de ressources est achevée, trois opérations d'ordonnancement doivent être effectuées:

1. Ordonnancement des tâches pour le robot alloué,

2. Ordonnancement du PICKUP avec les autres PICKUP dans la même zone.
3. Ordonnancement du PUTDOWN avec les autres PUTDOWN dans la même zone.

Le premier ne pose pas de problème particulier. Il s'agit simplement d'ajouter une contrainte de précédence entre l'instant de disponibilité du robot et la nouvelle tâche à laquelle on vient de l'allouer. L'**instant de disponibilité** du robot est alors mis à jour: il s'agit de la fin de la tâche nouvellement insérée. Les deux autres points doivent être étudiées un peu plus en détail.

3.3.1. Ordonnancement des Actions de Déchargement

Rappelons que le critère de choix du container amène à instancier en premier le container qui doit être saisi en premier. Une action de type PICKUP qui vient d'être instanciée doit donc nécessairement succéder à celles qui ont déjà été instanciées sur la même zone. On choisira donc d'ajouter une contrainte temporelle forçant le PICKUP de la tâche en cours d'instanciation à succéder au précédent. Le fenêtres temporelles des deux PICKUP ainsi ordonnés sont alors mises à jour, comme l'illustre la figure ci-dessous.

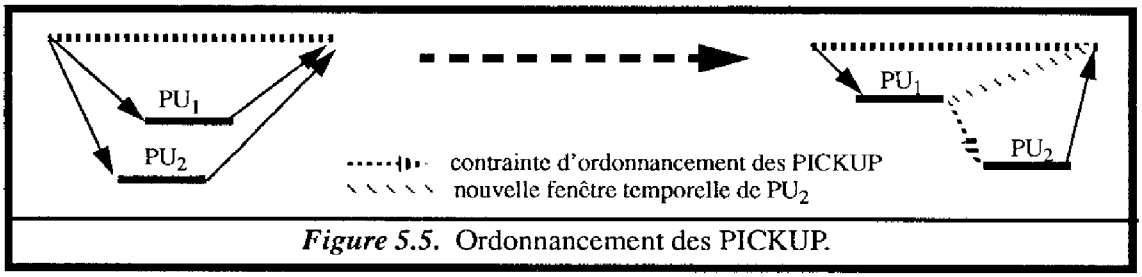


Figure 5.5. Ordonnancement des PICKUP.

3.3.2. Ordonnancement des Actions de Chargement

Ici, le problème est un peu plus complexe. Si nous forçons le PUTDOWN de la tâche en cours à succéder aux autres PUTDOWN précédemment instanciés, nous risquons de nous écarter sensiblement de la solution optimale. Considérons pour nous en convaincre l'exemple de la figure ci-dessous.

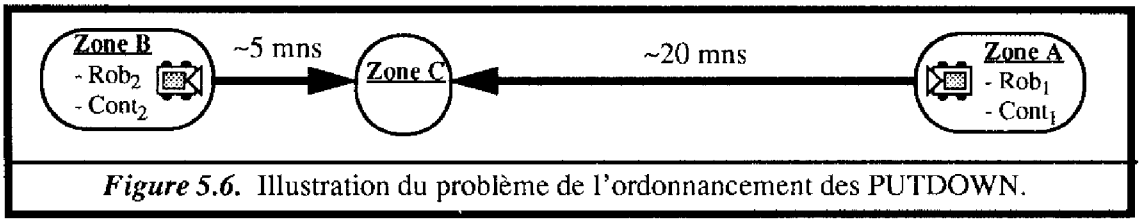


Figure 5.6. Illustration du problème de l'ordonnancement des PUTDOWN.

Le robot $Robot_1$, en zone *ZoneA*, est alloué au container $Cont_1$ (le plus prioritaire), qui doit être transporté de *ZoneA* à *ZoneC*, ce qui prend environ 20 minutes. Après quoi le robot $Robot_2$, en zone *ZoneB*, est alloué au container $Cont_2$, qui doit être transporté de *ZoneB* à *ZoneC*, ce qui prend environ 5 minutes. Les deux tâches seront déclenchées en même temps, la date de disponibilité des deux robots étant la même. On constate alors que $Robot_2$ arrive dans *ZoneC* **avant** $Robot_1$, bien qu'il ait été alloué à une étape ultérieure. Il n'est donc pas souhaitable de forcer PUTDOWN ($Robot_2$, $Cont_2$, *ZoneC*) à avoir lieu **après** PUTDOWN ($Robot_1$, $Cont_1$, *ZoneC*).

Nous devrions au contraire insérer PUTDOWN ($Robot_2$, $Cont_2$, *ZoneC*) avant PUTDOWN ($Robot_1$, $Cont_1$, *ZoneC*). En toute généralité, ceci peut décaler l'action PUTDOWN ($Robot_1$, $Cont_1$, *ZoneC*), et donc changer la date de disponibilité de $Robot_1$, et donc remettre en cause les tâches suivantes de celui-ci. Dans une perspective de recherche de solution sans retour-arrière, de telles remises en cause du graphe courant sont difficilement tolérables.

Nous pouvons déjà observer qu'il est possible de **conserver les PUTDOWN en parallèle tant que les robots correspondants n'ont pas été réalloués à de nouvelles tâches**. C'est en effet au moment où j'alloue le robot à une nouvelle tâche que je dois connaître, et ensuite modifier, sa date de disponibilité. Je devrai donc à ce moment-là fixer la position de son dernier PUTDOWN si je veux être sûr de ne pas avoir à effectuer de retour-arrière sur ce choix.

Il est heureusement possible de démontrer la propriété ci-dessous (qui constitue une condition suffisante), pour laquelle nous nous donnons les notations suivantes:

- $i \geq j$ signifie que l'étape d'itération i de la boucle d'allocation est postérieure à l'étape j.
- PUTDOWN^{z,i} est le PUTDOWN instancié dans la zone z à l'itération i.
- la date de début de PUTDOWN^{z,i} est l'intervalle $[inf^{z,i}, sup^{z,i}]$.
- on définit alors pour une zone z donnée:

"PUTDOWN$^{z,i'}$ se situe avant PUTDOWN$^{z,i''}$ ssi $inf^{z,i'} < inf^{z,i''}$.

Propriété 5.1 : Soit un robot $Robot_1$ en cours de réallocation au container $Cont_x$, à l'étape j. Soit PUTDOWN^{z,i} le dernier PUTDOWN effectué par ce robot. Alors:
- il n'existera pas d'étape $i' \geq j$ telle que "PUTDOWN$^{z,i'}$ se situe avant PUTDOWN^{z,i}".

Preuve:

Raisonnons par l'absurde. Soit $Robot_2$ le robot alloué à PUTDOWN$^{z,i'}$. PUTDOWN^{z,i} et PUTDOWN$^{z,i'}$ ont par définition la même durée. Donc, si PUTDOWN$^{z,i'}$ se situe avant PUTDOWN^{z,i}, cela veut dire que $Robot_2$ termine sa tâche avant $Robot_1$ sur la zone z (les deux PUTDOWN ayant par définition la même durée imprécise). Cela signifie que si nous l'avions choisi à l'étape j pour se charger de $Cont_x$, il aurait pu venir directement en

zone z et enchaîner sur la tâche de transport de $Cont_x$ plus rapidement que $Robot_1$. En d'autres termes, $Robot_2$ aurait dû être choisi à la place de $Robot_1$ à l'étape j.

CQFD.

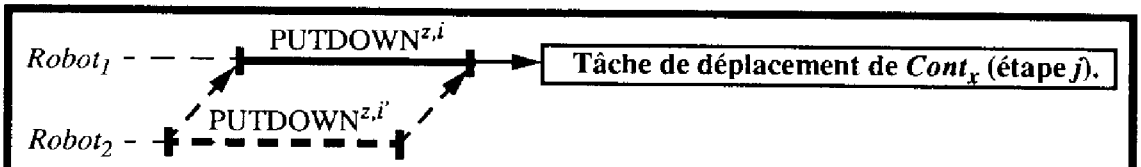


Figure 5.7. Propriété d'optimalité d'un ordonnancement différé de PUTDOWN.

La figure ci-dessus illustre la preuve de la propriété, laquelle peut se résumer ainsi: à partir du moment où l'on réalloue un robot, on est sûr qu'aucun autre PUTDOWN$^{z,i'}$ ne viendra s'insérer avant le dernier PUTDOWN^{z,i} qu'il avait effectué à l'étape i. On peut donc fixer le positionnement temporel de PUTDOWN^{z,i} à ce moment-là, en étant sûr de ne pas avoir à revenir sur ce choix.

La figure ci-dessous illustre la stratégie d'ordonnancement des PUTDOWN qui en découle. **On conserve les PUTDOWN en parallèle jusqu'à ce qu'un robot soit réalloué**. A ce moment-là, on impose à son dernier PUTDOWN de précéder les autres: dans la figure ci-dessous, l'ordonnancement forcé de PD_1 conduit à repousser les PUTDOWN PD_2 et PD_3, et donc à modifier les dates de disponibilité des robots correspondants, lesquels, par application de la propriété 5.1, n'ont pas encore été réalloués. Ce principe amène à une **mise-à-jour des fenêtres temporelles** exactement comme dans le cas des PICKUP.

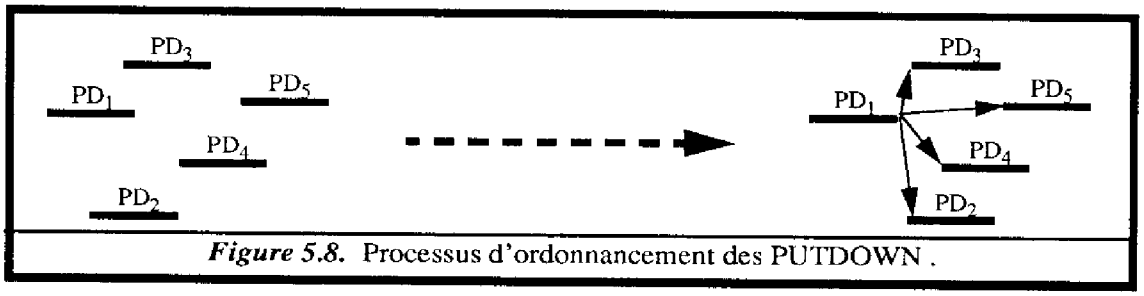


Figure 5.8. Processus d'ordonnancement des PUTDOWN .

3.4. Processus Complet d'une Etape d'Allocation

Nous avons donc mis en avant une stratégie de moindre engagement, afin d'assurer un ordonnancement des opérations de déchargement et de chargement sans retour-arrière permettant la détermination d'une solution répondant à nos critères d'optimalité.

- Les PICKUP sont ordonnés immédiatement,
- Les PUTDOWN sont ordonnés dès que le robot qui leur est associé est réalloué.

Signalons que dans [Vidal95], nous suggérons que nous pouvons également, grâce à des heuristiques appropriés, traiter les cas où PICKUP et PUTDOWN doivent être effectués dans une même zone. En guise de conclusion à cette partie, nous donnons ci-après l'algorithme général d'une étape d'allocation.

1. Choix d'un container $\boldsymbol{Cont_X}$ parmi les containers accessibles.
2. Choix du robot $\boldsymbol{Robot_Y}$ pour transporter $\boldsymbol{Cont_X}$.
3. Ordonnancement de la tâche de transport de $\boldsymbol{Cont_X}$ par rapport à l'instant de disponibilité de $\boldsymbol{Robot_Y}$.
4. Ordonnancement strict du dernier PUTDOWN effectué par $\boldsymbol{Robot_Y}$.
5. Ordonnancement strict du PICKUP effectué par $\boldsymbol{Robot_Y}$ pour le container $\boldsymbol{Cont_X}$.
6. Mise à jour de la tâche du robot $\boldsymbol{Robot_Y}$.

L'étape 6 consiste simplement à mettre à jour la **table des variables** en instanciant les variables des actions de la tâche en cours (cf § 2.2). Les étapes 1 et 2 concernent les **choix heuristiques** liés à l'allocation de ressources, alors que les étapes 3 et 5 concernent les **ajouts de contraintes temporelles** qui en découlent. Ceci illustre parfaitement la séparation entre ces deux aspects complémentaires du problème.

3.5. Limites de Pertinence des Critères de Choix

Revenons sur le choix du robot. Nous avons vu qu'il s'agit de calculer pour chaque robot ce que serait sa date d'arrivée sur la zone de déchargement. Du fait des données numériques imprécises en entrée, ces dates se présentent sous la forme d'intervalles *[date_au_plus_tôt, date_au_plus_tard]*. Les divers intervalles calculés sont donc susceptibles de **se recouvrir** temporellement.

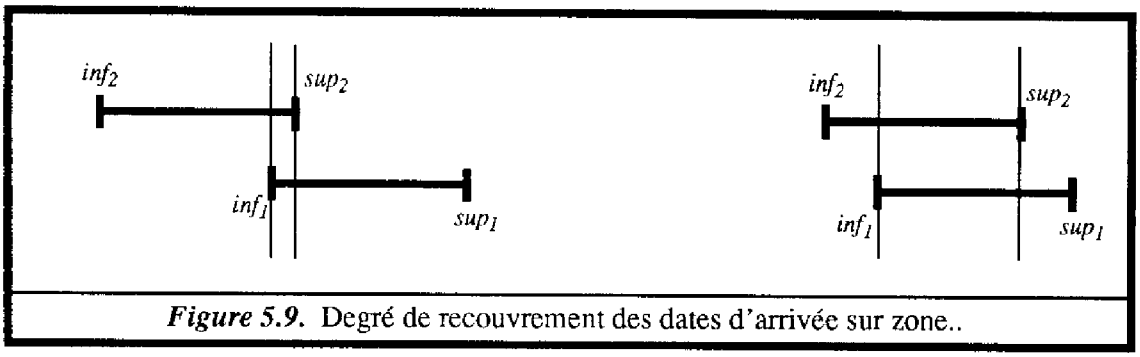


Figure 5.9. Degré de recouvrement des dates d'arrivée sur zone..

Considérons deux robots $Robot_1$ et $Robot_2$ par exemple, dont les dates d'arrivée estimées sont $d_1=[inf_1, sup_1]$ et $d_2=[inf_2, sup_2]$. Alors nous avons

- si $sup_1 \leq inf_2$, alors il est **certain** que $Robot_1$ arrivera le premier,
- si $sup_2 \leq inf_1$, alors il est **certain** que $Robot_2$ arrivera le premier,
- sinon il y a recouvrement de $[inf_1, sup_1]$ et $[inf_2, sup_2]$, c'est-à-dire qu'il est **possible** que $Robot_1$ **ou** $Robot_2$ arrive le premier.

C'est-à-dire que dans le troisième cas, l'incertitude ne nous permet pas de déterminer l'allocation optimale. Nous avons choisi de moduler quelque peu cette affirmation en acceptant un "faible" recouvrement des intervalles. La figure ci-dessus représente deux cas de recouvrement. Nous voyons que dans le premier cas, nous pouvons nous permettre d'effectuer le choix de $Robot_1$, qui a les plus grandes chances d'arriver effectivement le premier, alors que dans le deuxième cas, tout choix serait arbitraire et ne nous permettrait pas de garantir un quelconque niveau d'optimalité de la solution engendrée.

On définit donc le **degré de recouvrement** de deux dates imprécises d_1 et d_2 comme étant
(1) $\Delta_R = (sup_1\text{-}inf_2) / (sup_2\text{-}inf_1)$ si $inf_1 \leq inf_2$ (voir figure ci-dessus),
(2) $\Delta_R = (sup_2\text{-}inf_1) / (sup_1\text{-}inf_2)$ si $inf_2 \leq inf_1$.

Il suffit alors de définir pour une mission donnée un **seuil de recouvrement** σ_R. Dès lors,
- si $\Delta_R \leq \sigma_R$, alors on choisit le robot qui minimise (inf_1, inf_2),
sinon on doit stopper le processus d'allocation de ressources du fait des incertitudes.

Notons que le seuil de recouvrement définit pour une mission donnée la **qualité** de la solution obtenue (c'est-à-dire le **degré de satisfaction des critères d'optimalité**). Un seuil $\sigma_R=0$ signifie que l'on ne recherche que les solutions optimales, un seuil $\sigma_R=20\%$ signifie que chaque choix effectué a 20% de chance de se révéler sous-optimal lors de l'exécution. Nous reviendrons sur le problème du choix d'un seuil de recouvrement pour une application donnée en section 5.

Partant de là, nous voyons apparaître notre méthode d'allocation de ressources en horizon glissant, qui sera précisée en section 5. En effet, plus on attribue de tâches à un robot, plus l'estimation de sa date de disponibilité doit faire la somme d'un nombre important de durées imprécises. Donc plus on alloue loin en avant dans le temps, plus l'imprécision des dates de disponibilité des robots augmente, et plus ces dates sont donc susceptibles de se recouvrir. Le seuil de recouvrement, une fois fixé, caractérise donc le degré de confiance (ou de fiabilité) minimal que l'on exige lors d'un choix. Il ressort de ce qui précède qu'il arrive nécessairement un moment où il n'est plus possible d'effectuer un **choix suffisamment fiable** entre deux robots.

C'est à ce moment-là que l'on choisit de stopper la phase d'allocation de ressources. L'exécution du plan suit alors son cours pendant qu'un "espion" surveille l'évolution des deux dates de disponibilité en conflit, de manière à pouvoir relancer l'allocation de ressources dès que l'exécution a levé l'incertitude. Nous reprendrons cela en section 5.

4. La Gestion du Graphe Temporel

Il va s'agir maintenant de décrire les techniques employées en matière de gestion des contraintes temporelles numériques, notamment en termes de **propagation** de ces contraintes dans le graphe. Nous allons commencer par éclairer cette section d'un commentaire préalable sur l'importance de cette phase.

4.1. La Propagation des Contraintes Temporelles: un Enjeu Décisif

Nous l'avons déjà dit, la détermination d'une solution passe par l'allocation des ressources mais aussi par l'allocation de dates de déclenchement pour chaque tâche (ordonnancement temporel proprement dit). Le premier point a été décrit dans la section précédente. Le second passe par l'ajout de contraintes temporelles qui nécessite la mise en place de mécanismes de propagation.

La question de la propagation, comme dans les chapitre précédents, se pose en termes de **compromis** entre **efficacité et complétude**, mais aussi entre **requêtes et mises-à-jour**. Nous allons étudier ces aspects tour à tour.

- Remarquons tout d'abord que notre graphe initial est composé exclusivement d'instants numériques. C'est-à-dire que nous ne pourrons pas utiliser les techniques de réduction à un sous-graphe du chapitre 3. Ceci est lié à la spécificité des problèmes d'ordonnancement qui, contrairement à leurs homologues issus du domaine de la planification, témoignent d'une **prédominance des contraintes temporelles numériques** [Erschler91].

- Remarquons également que la **dimension** de notre graphe est loin d'être négligeable. Nous associons à un container quatre actions, soit huit instants. Il faut multiplier cela par le nombre de containers, et ajouter les instants d'arrivée et de départ des convoyeurs. Pour 200 containers, cela nous donne par exemple des graphes de plus de 1600 instants !

- L'exigence en matière de **complétude** de la propagation reste intacte, la vérification de faisabilité n'interdisant pas l'apparition d'une incohérence lors d'une étape d'allocation, du fait de l'accumulation imprévue de temps morts entre deux PICKUP par exemple (cf § 3.2). Cette incohérence doit pouvoir être détectée au plus tôt, afin de chercher à anticiper l'échec en relaxant des contraintes (ce qui reviendrait par exemple dans notre application à négocier un départ différé d'un convoyeur).

- L'exigence en termes d'**efficacité** est particulièrement importante, puisqu'allocation et exécution ont lieu en parallèle.

- Les requêtes temporelles ont une importance primordiale puisque les critères de choix sont temporels, et nécessitent donc une **interrogation constante du graphe**. Elles sont par contre relativement **simples**: il va s'agir simplement de chercher des dates (disponibilité du robot et fenêtres temporelles des PICKUP et des PUTDOWNS, comme cela va être détaillé dans ce qui suit).

Nous pouvons conclure de ce qui précède les choses suivantes: l'algorithme de propagation doit être très **performant** (si possible en temps quasi-constant, du fait du fonctionnement en ligne de l'algorithme d'allocation). Il doit par ailleurs **vérifier la cohérence** et rendre compte des **dates minimales** dans le graphe. Ceci constitue une exigence restreinte par rapport aux chapitres précédents, les seules contraintes minimales que nous souhaitons récupérer étant les dates (contraintes de type C_{0i}). A partir de ces spécifications, nous allons développer notre méthode dans le paragraphe suivant.

4.2. Une Méthode de Décomposition du Graphe Temporel

Nous allons voir qu'il est possible de profiter de la structure particulière du graphe pour décomposer celui-ci en un ensemble de composantes dans lesquelles la propagation pourra être confinée. Ceci nous permettra d'obtenir un gain plus que substantiel en termes de performances, tout en satisfaisant nos exigences en matière de complétude. Nous allons tout d'abord commencer par définir cette décomposition, après quoi nous analyserons ses propriétés.

4.2.1. Décomposition en Composantes-Tâches

A chaque container est associé une tâche prédéfinie, et donc un ensemble d'instants et de contraintes particuliers du graphe, comme nous l'avons vu au paragraphe 2.3. La définition suivante offre un cadre rigoureux à ce concept.

Définition 5.1 : Soit un graphe $\mathcal{G} = \{V, D, C\}$ de type STP (cf chapitre 1). A tout container $Cont_x$, on associe une **composante-tâche** $\Theta_x = \{V_x, D_x, C_x\}$ où

- V_x est l'ensemble des instants suivants:
  - l'instant initial 0,
  - les 8 instants correspondant aux débuts et fins des quatre actions dévolues au container $Cont_x$,
  - les 4 instants correspondant aux **fenêtres temporelles** du PICKUP et du PUTDOWN,
  - dans le cas où la tâche est instanciée, l'instant de **disponibilité** précédent du robot alloué.
- D_x représente les domaines de ces instants, c'est-à-dire les intervalles de dates possibles,
- C_x représente l'ensemble des **contraintes d'entrée** pesant sur ces instants.

La figure suivante illustre cette décomposition. On présente quatre composantes-tâches reliées entre elles par des contraintes temporelles

- d'ordonnancement de tâches pour un même robot,
- et d'ordonnancement des PICKUP et PUTDOWN pour une même zone.

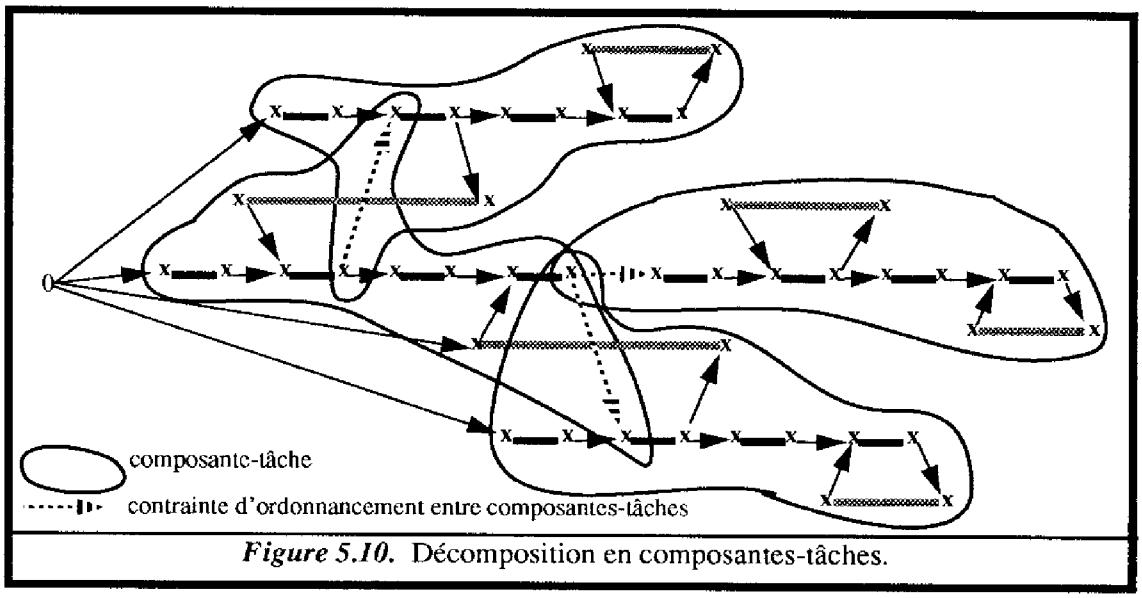


Figure 5.10. Décomposition en composantes-tâches.

Pour des raisons évidentes de lisibilité de la figure, nous avons choisi de ne pas inclure l'instant 0 dans ces composantes-tâches. De même, des précédences entre 0 et certains instants du graphe ont été omises pour les mêmes raisons. La composante-tâche représentée en grisé peut être vue comme ayant les trois autres comme *voisines*, comme nous allons le voir dans le paragraphe suivant. Donnons-nous auparavant une autre définition qui va nous être utile.

Définition 5.2 : L'ensemble $\Theta = \{\Theta_x / x=1,...,\gamma\}$ est une décomposition du graphe $\mathcal{G}$. Elle sera dite **totale** ssi
- $\forall i \in V, \exists x / i \in V_x$ (partition complète sur les instants),
- $\forall N_{ij} \in C, \exists x / N_{ij} \in C_x$ (partition complète sur les contraintes d'entrée).

4.3. Propriétés de cette Décomposition

Propriété 5.2 : La décomposition en composantes-tâches est une décomposition totale.

La preuve provient directement de la définition même des composantes-tâches. Elle est immédiate en ce qui concerne les instants. Une contrainte d'entrée quant à elle relie
- soit deux instants liés aux actions des containers,
- soit un PICKUP ou un PUTDOWN avec les instants définissant sa fenêtre temporelle,
- soit la fin d'un PUTDOWN et le début du GOTO de la tâche suivante, ce qui apparaît dans la composante-tâche de cette tâche grâce à la présence de l'instant de disponibilité précédent.

Définition 5.3 : $\forall\ \Theta_x$ et $\Theta_{x'}$ composantes-tâches de Θ, Θ_x et $\Theta_{x'}$ sont **voisins** ssi $\exists$ une contrainte d'entrée $N_{ij} \in C / N_{ij} \in C_x$ et $N_{ij} \in C_{x'}$.

Nous définissons alors l'algorithme de propagation suivant: soit N_{ij} une contrainte ajoutée ou modifiée. Alors $\forall\ \Theta_x$ tel que $(i, j) \in V_x^2$, on exécute la fonction

- *Propager-Comp-Tâche*(N_{ij}, Θ_x):
  1. effectuer une propagation de N_{ij} dans Θ_x par cohérence de chemin,
  2. $\forall\ N_{kk'}$ modifiée dans Θ_x ET $\forall\ \Theta_{x'}$ voisin de Θ_x tel que $N_{kk'} \in C_{x'}$: exécuter *Propager-Comp-Tâche*($N_{kk'}$, $\Theta_{x'}$).

Il s'agit donc d'une fonction récursive effectuant des mises-à-jour par cohérence de chemin dans chaque composante-tâche, propageant les modifications d'une composante-tâche à l'autre par l'intermédiaire de la relation de voisinage, et ceci jusqu'à ce qu'aucune contrainte ne soit plus modifiée. Nous obtenons alors les propriétés de complétude suivantes. Nous ne donnons ensuite que les grandes lignes de la démonstration.

Propriété 5.3 : L'algorithme incrémental *Propager-Comp-Tâche*
(a) est **complet** vis-à-vis de la cohérence globale du graphe,
(b) et fournit les **dates minimales** du graphe global.

Preuves:

(b) Raisonnons par l'absurde. La date de l'instant i peut être vue du fait de notre algorithme comme une contrainte N_{0i}. Supposons qu'elle ne soit pas minimale. Cela signifie qu'il existe un chemin de contraintes d'entrée $\{0, k_1, ..., k_n, i\}$ tel que $(N_{0k1} \oplus N_{k1k2} \oplus ... \oplus N_{kni}) \subset N_{0i}$. Or, $\exists \Theta_x / (k_1, k_2) \in V_x^2$ (de part la propriété de partition complète des contraintes d'entrée). Par définition des composantes-tâches, $0 \in V_x$. Donc, du fait du maintien de cohérence dans Θ_x, nous avons nécessairement $N_{0k2} \subseteq N_{0k1} \oplus N_{k1k2}$. Si l'on répète le raisonnement pour chaque contrainte d'entrée composant le chemin, on aboutit finalement à $N_{0i} \subseteq (N_{0k1} \oplus N_{k1k2} \oplus ... \oplus N_{kni})$, ce qui contredit l'hypothèse initiale.

CQFD.

(a) Nous avons vu que dans les réseaux PERT (cf chapitre 4), la seule contrainte à satisfaire était la durée totale du programme. De manière similaire, nous n'avons ici qu'un ensemble clairement identifié de contraintes à satisfaire. Il s'agit de vérifier que la succession des PICKUP [resp. PUTDOWN] sur une même zone se déroule nécessairement pendant le délai de présence du convoyeur. Du fait de l'utilisation des fenêtres temporelles, ceci est équivalent à la satisfaction d'un ensemble de contraintes locales, à savoir la nécessité pour chaque PICKUP [resp. PUTDOWN] de se dérouler entre les deux instants de sa fenêtre temporelle, ce qui est vérifié par l'algorithme de cohérence de chemin dans chaque composante-tâche.

CQFD.

En termes de **performances**, chaque composante-tâche étant composée de 14 instants au maximum, la complexité de *Propager-Comp-Tâche* est de $O(\gamma.14^3) \equiv O(\gamma)$ en pire cas (où γ est le nombre total de containers), ce qui est largement inférieur aux complexités des algorithmes de filtrage appliqués au graphe global dans les chapitres précédents. La section suivante va nous permettre de voir que les performances sont en fait encore supérieures. Mais avant cela, signalons à titre de remarque supplémentaire que le filtrage par cohérence de chemin dans chaque composante-tâche permet par ailleurs de récupérer

- les contraintes minimales entre les instants de début et de fin des actions de déchargement et de chargement et leurs fenêtre temporelles, fournissant ainsi une information de type "marge" disponible pour l'exécution de ces actions,

- les contraintes entre le début du premier GOTO et la fin du PUTDOWN pour une tâche donnée, celle-ci pouvant être récupérée directement par les requêtes effectuées par l'algorithme de choix du robot (cf § 3.2.2).

5. La Boucle de Contrôle Globale

5.1. Exécution en Horizon Glissant et Architecture Globale

Comme nous l'avons vu, lorsque la boucle de décision ne peut plus choisir un robot avec suffisamment de pertinence, le relais est passé à un module de simulation d'exécution (cf § 3.5), qui gère le déclenchement des tâches et l'arrivée des événements correspondant aux fins de tâches et aux arrivées et départ des convoyeurs. Nous ne détaillerons pas cette phase. Signalons simplement qu'au fur et à mesure que des décisions sont prises, ou que des observations surviennent, ces informations sont propagées par l'intermédiaire d'un algorithme de **cohérence d'arc dirigée** [Vidal95], dont la complexité est en $O(m.\rho)$, où m est le nombre d'instants composant la partie du graphe à exécuter, et ρ est le nombre de robots. Cet algorithme va donc progressivement lever l'incertitude pesant sur le choix d'allocation de ressources, jusqu'à ce que ce choix puisse être à nouveau fait avec une fiabilité suffisante, c'est-à-dire jusqu'à ce que $\Delta_R \leq \sigma_R$ (cf § 3.5).

Notre système de simulation fonctionne pour l'instant en alternant ordonnancement et exécution. Une extension naturelle, qui reste à faire, consisterait à disposer d'un véritable parallélisme entre les deux processus.

On peut remarquer que le module d'ordonnancement et d'allocation de ressources travaille en anticipation (principe d'horizon glissant) par rapport au module d'exécution. Par contre, les propagations s'effectuent sur la même partie du graphe, qui correspond aux composantes-tâches qui ont été **instanciées mais non encore exécutées.**

5.2. Caractérisation des Solutions

En fait, on constate que deux critères sont en compétition lorsque l'on cherche à caractériser ce qu'est une "bonne" solution.

- La **qualité** de la solution correspond à la probabilité plus ou moins grande que la solution effective produite à l'exécution sera optimale par rapport aux critères temporels évoqués précédemment. Ceci dépend bien sûr du seuil de recouvrement σ_R: plus celui-ci est faible, plus on a de chance que le robot choisi sera effectivement le meilleur au moment de l'exécution.

- La **sécurité** de la solution correspond à la certitude d'avoir toujours pour chaque robot une anticipation "confortable", c'est-à-dire au moins une tâche attribuée en avant par rapport à l'exécution courante. Ceci dépend également du seuil de recouvrement σ_R: plus celui-ci est important, moins on a de chance de bloquer systématiquement sur le choix d'un robot et de ne disposer éventuellement d'aucune tâche attribuée pour un robot à un moment donné durant l'exécution, ce qui conduirait à un blocage du système global.

Tout dépend également du **degré d'imprécision** des données temporelles en entrée. Si les durées sont assez précises, on pourra imposer un seuil de recouvrement σ_R faible, voire égal à 0, garantissant la sécurité de la solution, tout en garantissant une bonne qualité de celle-ci. Par contre, si ce degré d'imprécision est important, alors il faudra trouver le meilleur compromis, c'est-à-dire garantir une sécurité minimale, de manière à disposer d'une qualité la plus satisfaisante possible.

Ceci est illustré par le tableau ci-dessous, où l'on peut voir le nombre de tâches attribuées en avant à chaque robot en fonction du seuil de recouvrement et du degré d'imprécision des contraintes d'entrée. Finalement, nous voyons qu'il convient de **fixer le seuil de recouvrement en fonction des imprécisions d'entrée**, de manière à maintenir une anticipation "minimale" d'environ une à deux tâches en avant pour chaque robot, ce qui constitue un compromis suffisant en termes de sécurité, et maximise la qualité de la solution obtenue.

De plus, cela amène à borner la partie du graphe sur laquelle ont lieu les propagations. Le nombre de tâches instanciées non encore exécutées étant constant pour un robot donné, la complexité ne dépend plus alors que du nombre de robots ρ. Nous obtenons donc des complexités en pire cas de $O(\rho)$ pour les propagations de la phase d'ordonnancement et d'allocation de ressources, et en $O(\rho^2)$ pour la phase d'exécution.

	imprécisions de l'ordre de 2 à 5 mns	imprécisions de l'ordre de 5 à 10 mns
seuil de 10%	3 tâches environ par robot	**1 à 2 tâches par robot**
seuil de 5%	**1 à 2 tâches par robot**	1, voire 0 (!) tâche par robot

Tableau 5.1. Caractérisation du degré d'anticipation.

En conclusion, nous obtiendrons finalement une solution dont **la qualité dépend du degré d'imprécision des données d'entrée**, ce qui semble assez naturel. Notons pour finir au niveau des prospectives que le recours à un seuil de recouvrement, fixé de manière plus ou moins empirique, constitue une première approche du problème, qu'il conviendrait d'améliorer par la

construction d'une **fonction coût**, paramétrée en fonction du degré d'imprécision temporelle, et évaluée à chaque comparaison de deux robots.

5.3. Résultats Expérimentaux

Une analyse expérimentale peut être trouvée dans [Vidal95], qui témoigne du gain déterminant de notre méthode vis-à-vis des techniques de filtrage classiques. Nous obtenons pour des exemples de petite taille (2 ou 3 robot et une dizaine de containers) des temps de 10 à 15 secondes sur une Sparc2 pour la boucle de décision complète, et de l'ordre de la milliseconde pour la phase d'exécution entre deux phases d'ordonnancement. A titre de comparaison, les techniques classiques rendent compte de temps de propagation allant jusqu'à 10 minutes.

Lorsque nous passons à des exemples avec 50 robots et 200 containers, le temps de la phase de prétraitement hors-ligne (expansion du graphe initial, propagations initiales, vérification de faisabilité) augmente de manière très sensible, alors que les algorithmes en ligne témoignent de temps de calcul quasi-identiques à ceux des petits exemples. Les tests fournissent donc une complexité expérimentale de la boucle de contrôle globale quasi-constante.

Nous avons aussi pu vérifier que la qualité de la solution produite était inversement proportionnelle au seuil de recouvrement fixé au départ.

5.4. Architecture Globale

Pour terminer, nous donnons ci-après l'architecture globale de notre système, où nous avons fait figurer les modules de gestion annexes (piles de container et table des variables).

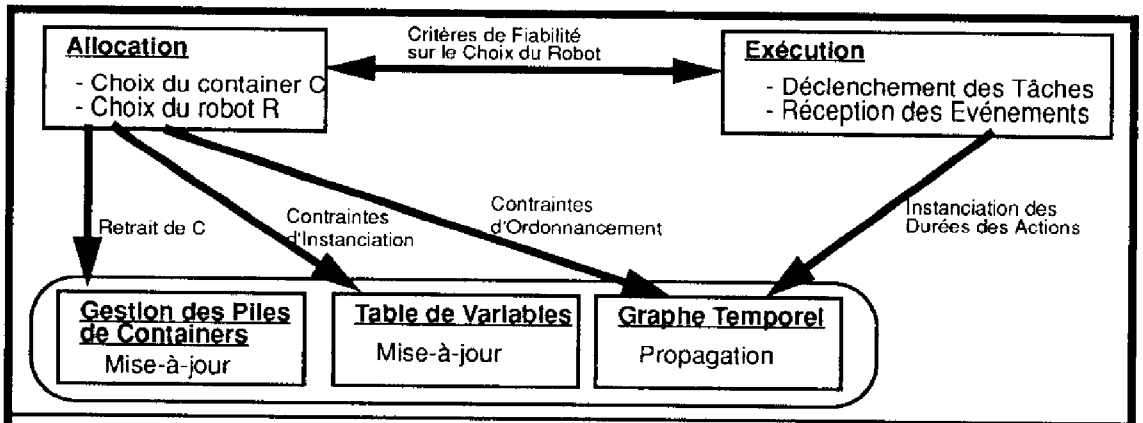


Figure 5.11. Architecture globale de l'application

6. Conclusion: Techniques Dépendant de l'Application pour une Efficacité Maximale

L'application qui a été présentée dans ce chapitre est un exemple d'utilisation de techniques de propagation temporelle utilisant la **structure spécifique** du graphe **dépendant de l'application**. Exploiter les caractéristiques de l'application nous permet de maintenir nos exigences à la fois en termes de **complétude** et d'**efficacité** dans un graphe temporel **de grande taille** et constitué de **contraintes essentiellement numériques**.

Il s'agit aussi d'un bon exemple de ce qui peut être fait dans des applications d'ordonnancement et d'allocation de ressources où les contraintes temporelles peuvent être plus riches en termes d'**expressivité** que dans un réseau PERT par exemple: la gestion explicite des fenêtres temporelles permet en effet de tenir compte facilement de contraintes de type *during*.

Notre contribution se situe également en termes de caractérisation du **degré d'optimalité** d'une solution en fonction du **degré d'incertitude** présent au départ. Par ailleurs, on peut noter que notre système est **incomplet** dans le sens où il est susceptible dans des situations complexes de ne pas trouver de solution alors qu'il en existe une. L'architecture modulaire, séparant la dimension temporelle des heuristiques de décision, facilite néanmoins l'**évolution** et l'**adaptativité** du système, permettant d'envisager du même coup une amélioration sensible des heuristiques, afin d'être à même de traiter de manière satisfaisante des cas de figure plus complexes.

Signalons pour finir que les résultats encourageants de cette application de simulation ont suggéré la mise au point d'une expérimentation "grandeur réelle" avec les robots du groupe, mêlant capacités de navigation distribuées et processus global d'allocation.

Chapitre VI. Discussions et Prospectives

Bien que certaines extensions aient déjà été évoquées dans les chapitres précédents, nous avons choisi de consacrer un chapitre complet à une discussion plus approfondie sur les suites qui doivent ou peuvent être envisagées à partir de notre travail. Ces prospectives ne se veulent rien d'autre que des réflexions informelles jalonnant un ensemble de pistes de recherche sur lesquelles, nous l'espérons, nous-mêmes ou d'autres auront à coeur de s'engager.

1. Portée et Limitations des Choix Algorithmiques

Nous allons dans un premier temps reprendre sur un mode volontairement critique certains de nos choix en matière de techniques et d'algorithmes de gestion de contraintes temporelles numériques, tels qu'ils ont été présentés au chapitre 3.

1.1. La Proportion d'Instants Numériques

Nous avons développé au chapitre 3 (§ 2.1) l'argument selon lequel la proportion d'instants numériques était relativement faible en planification. Ceci était dû, avons-nous dit, au fait que l'occurrence d'un effet se positionne de manière simplement symbolique par rapport à la tâche. En fait, on peut d'abord remarquer que l'on aurait tort de généraliser. Souvent, positionner plus précisément (donc numériquement) l'occurrence d'un effet par rapport au début de la tâche peut permettre de guider plus efficacement la recherche d'un plan. Par ailleurs, il est souvent possible pour des tâches simples d'approximer la représentation en considérant que l'effet débute en même temps que la tâche par exemple, ce qui rend inutile l'utilisation d'un instant supplémentaire. La perte d'information engendrée par cette approximation n'est pas forcément pénalisante pour la recherche du plan.

Mais il y a plus important. L'introduction d'une dichotomie entre contraintes contingentes et libres au chapitre 4 nous amène à formuler les choses de manière légèrement différente. Dire en effet que l'occurrence de l'effet est reliée symboliquement aux instants de début et de fin de la tâche suppose implicitement que cette occurrence est libre (un Lien étant toujours libre, cf chapitre 4, § 3.1), ce qui est rarement le cas. Puisque la durée d'une tâche est généralement contingente, il en est de même pour l'occurrence des effets à l'intérieur de celle-ci. En fait, de deux choses l'une:

- soit l'occurrence de l'effet est contingente, et alors il serait logique de disposer de la durée contingente le séparant du début de la tâche, ce qui en fait un instant numérique.
- soit l'occurrence de l'effet est libre, et alors rien n'empêche de considérer arbitraire-

ment qu'il est confondu avec le début ou la fin de la tâche, comme nous l'avons suggéré plus haut.

Ces premiers éléments de réflexion ne remettent pas forcément en cause notre méthode, mais il paraît nécessaire de les approfondir en analysant de manière plus fine des exemples réels de planification.

♣♣♣

Une autre question à creuser est la possibilité d'appliquer ce critère de faible proportion numérique à **d'autres domaines** que la planification. Le chapitre 5 a permis de mettre en avant le fait que l'ordonnancement et l'allocation de ressources présentaient des applications caractérisées par des contraintes temporelles presque exclusivement numériques. Il resterait à analyser par exemple le domaine du diagnostic, dans lequel la relation d'antériorité joue par contre un rôle prépondérant [Console92].

1.2. Compromis entre Requêtes et Mises à Jour

Notre choix s'est résolument porté au chapitre 3 sur la priorité donnée aux requêtes, conduisant au maintien du réseau minimal. Néanmoins, même avec la restriction à un sous-graphe, la propagation numérique reste une étape pénalisante dans le système IxTeT. Il pourrait être avantageux à l'exemple de [Barber93] de faire cohabiter deux types d'algorithmes distincts dans le même système, l'un privilégiant les requêtes, et l'autre les mises-à-jour, et de comparer les avantages de l'un et de l'autre selon les cas de figures.

Une approche plus tentante consisterait à s'inspirer de ce que nous avons fait au chapitre 5 pour remettre en cause nos conclusions du chapitre 3, et tenter d'extraire certaines caractéristiques structurelles liées à la planification de manière à restreindre l'exigence de complétude des requêtes à des "**composantes**" particulières du graphe. L'objectif étant au niveau de la propagation de limiter l'explicitation des contraintes minimales à celles seules susceptibles d'être requises par le planificateur, tout en garantissant bien sûr le prérequis de complétude des mises-à-jour. Il serait utile également de se pencher sur la littérature abondante traitant des améliorations apportées aux **algorithmes** "fondateurs" des méthodes CSP tels que PC-2.

1.3. Retraits

Il est un point que nous avions provisoirement laissé de côté au début de ce travail de recherche, et qui n'aura finalement pas été traité. Rappelons-nous que les algorithmes de ges-

tion des contraintes symboliques traitent aussi bien les ajouts que les retraits. Il serait souhaitable de disposer des mêmes fonctionnalités dans le cadre des contraintes numériques, de manière à pouvoir faire face efficacement à un **retour-arrière** au niveau de l'algorithmique de planification. En effet, la vérification de cohérence (ou de contrôlabilité) a priori ne suffit pas. Dans le cas d'un retour-arrière intelligent, il est possible d'incriminer une ou plusieurs contraintes ajoutées à une étape précédente.

Le problème est loin d'être trivial. Comme le fait remarquer B. Choueiry [Choueiry94], les techniques classiques de propagation des CSPs présentent l'inconvénient de gérer difficilement la **non-monotonie**. Une réflexion de fond s'impose donc sur cet aspect.

2. Enrichir la Notion d'Incertitude Temporelle

La prise en compte d'un degré d'incertitude temporelle par l'intermédiaire du Graphe de Décision constitue sans nul doute la contribution qui nous ouvre le plus de prospectives prometteuses. Le chapitre 4 se terminait (section 6) par l'évocation de quelques extensions envisageables à plus ou moins court-terme. Nous allons dans un premier temps revenir rapidement sur celles-ci, après quoi nous développerons d'autres aspects qui nous semblent s'inscrire dans une continuité logique du travail réalisé.

2.1. Variables Aléatoires Dépendantes

Le lecteur a certainement encore en tête notre discussion concernant la difficulté inhérente à la prise en compte de **variables aléatoires dépendantes**. En guise de conclusion sur ce thème, nous nous contenterons de dire qu'il est indispensable de mener rapidement une analyse formelle du problème selon deux axes.

- D'une part, une réflexion récente amène à penser que le concept de contrôlabilité ou de cohérence forte joue un rôle non négligeable dans les limitations du modèle. En fait, il semblerait que la propagation dans le Graphe de Décision impose en définitive une **contrôlabilité "plus forte"** que celle que l'on souhaitait. Les résultats du paragraphe 4.4 du chapitre 4 auraient avantage a être affinés, et il y a fort à parier que la cohérence dans $\mathcal{D}$ se révélerait être en définitive équivalente à une forme de contrôlabilité forte dans un sous-ensemble de $\mathcal{N}$. Pour mettre cela en lumière, il paraît de fait nécessaire de reprendre de manière plus rigoureuse les fondements théoriques de notre méthode.

- Quoi qu'il en soit, il reste intéressant et sans doute nécessaire d'envisager d'enrichir la représentation par une **prise en compte satisfaisante de contraintes contingentes dépendant les unes des autres.** Néanmoins, de notre point de vue, cet aspect relève des prospectives à long-terme, nécessitant sans aucun doute le développement de techniques nouvelles qui dépassent le cadre du Graphe de Décision.

2.2. Contraintes Symboliques Contingentes

Nous avons étudié dans le chapitre 4 la prise en compte des contraintes numériques contingentes. Pourquoi ne pas envisager de prendre également en compte des contraintes **symboliques** contingentes ? Ramené à l'algèbre d'instants, cela signifie que l'on pourrait avoir à tenir compte d'un événement devant arriver **avant OU après** un autre, et ceci de manière incontrôlable. Néanmoins, comme nous l'avons argumenté plus haut (cf § 1.1), cela signifie que nous disposons en fait nécessairement de la distance temporelle imprécise entre ces deux événements, c'est-à-dire que l'on se ramène à une contrainte numérique contingente (CTG faible). On voit là qu'une fois encore, la représentation numérique englobe par défaut la représentation symbolique.

2.3. Nécessité de Moduler l'Incertitude

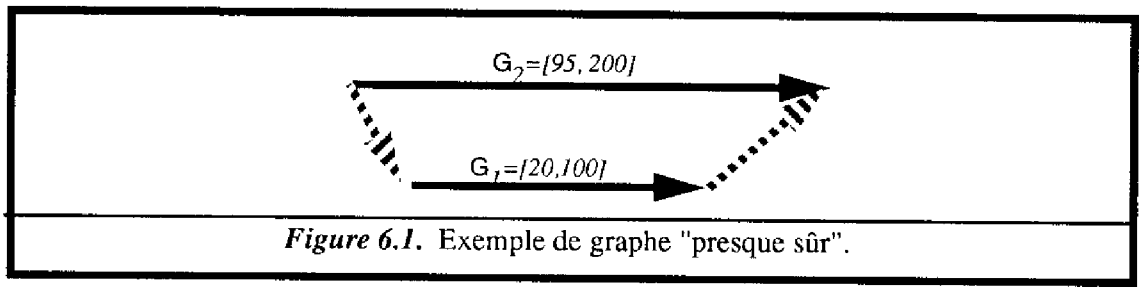


Figure 6.1. Exemple de graphe "presque sûr".

Le chapitre 4 traite de l'incertitude de manière binaire: le réseau est contrôlable ou ne l'est pas, ce qui signifie que le plan est sûr ou ne l'est pas. Ceci constitue une approche trop "**pessimiste**" des problèmes de planification. Considérons un exemple très simple, où une tâche de durée contingente G_1=[20,100] doit avoir lieu *pendant* une autre tâche de durée également contingente G_2=[95,200]. Si l'on considère qu'il y a équiprobabilité entre les diverses valeurs possibles des durées aléatoires, alors on peut dire que l'on est "**presque sûr**" que cette meta-contrainte sera satisfaite. Pourtant elle sera considérée comme incontrôlable au niveau du Graphe de Décision.

Cet aspect est traité dans le chapitre 5 sous la forme d'une mesure du *degré de recouvrement* d'intervalles numériques. Une manière plus élégante et mieux fournie en outils algorithmiques adaptés consiste à "moduler" l'incertitude. On peut se référer pour cela aux modélisations **probabilistes** (voir par exemple [Sadeh89] où la flexibilité introduite au niveau des contraintes temporelles aide à diriger la résolution de problèmes d'allocation de ressources en ordonnancement), ou bien aux modélisations **possibilistes** [Kerr89, Dubois89]. Dans ce dernier cas, on caractérise une durée contingente par un ensemble de valeurs "plus ou moins possibles", conduisant à une représentation sous forme de trapèzes. Nous avons déjà parlé à la fin du chapitre 4 des travaux de l'équipe de D. Dubois et H. Prade, notamment de l'article [Dubois93] que nous avons comparé sur certains points à notre approche. Un travail a été initié dans le cadre de la thèse pour étudier la possibilité d'introduire une **modélisation floue des contraintes temporelles dans IxTeT**, dans le cadre d'une extension du paradigme du Graphe de Décision. Cette extension est apparue aisément réalisable sur le modèle des travaux réalisés par H. Fargier. Cette étude n'a cependant pas été suffisamment poussée pour pouvoir être présentée dans ce mémoire. Il s'agirait donc de poursuivre ce travail qui pourrait être achevé dans des délais relativement brefs.

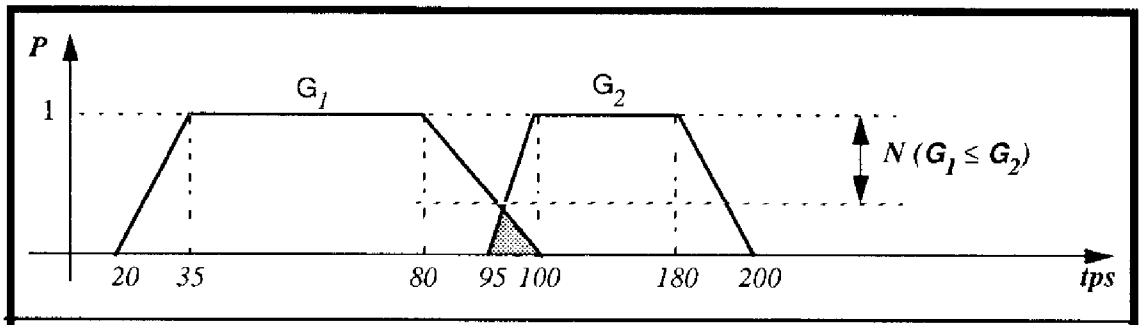


Figure 6.2. Modélisation possibiliste de l'exemple.

Essayons de visualiser la méthode très succinctement au travers de l'exemple évoqué au début de ce paragraphe. Une modélisation floue de celui-ci pourrait conduire à une représentation des deux contraintes sous la forme des deux trapèzes de la figure ci-dessus. Pour G_1 par exemple, les valeurs comprises entre 35 et 80 (la "base" du trapèze) sont "tout-à-fait possibles" (degré de possibilité P égal à 1), alors que celles comprises entre 80 et 100 ainsi que celles comprises entre 35 et 20 sont "de moins en moins possibles" (degré de possibilité P tendant vers 0). La figure montre également une mesure du degré de certitude que la tâche de durée G_1 ait effectivement lieu *pendant* la tâche de durée G_2, c'est-à-dire $N(G_1 \leq G_2)$. Intuitivement, cette certitude sera d'autant plus faible que les deux trapèzes auront tendance à se recouvrir. S'ils sont distincts, alors $N(G_1 \leq G_2)=1$. Si les bases des trapèzes s'intersectent, alors $N(G_1 \leq G_2)=0$. La figure illustre un cas de figure intermédiaire.

Pour faire le lien avec le Graphe de Décision, $N(G_1 \leq G_2)$ fournit un **degré de contrôlabi-**

lité locale pour les deux CTGs considérées. L'intérêt d'une modélisation floue en planification tient à la possibilité de moduler le **degré de satisfaction du plan produit** en fonction des degrés de certitudes pesant sur les contraintes numériques en entrée, permettant donc d'accepter des plans considérés comme étant presque sûrs. Le problème qui reste à résoudre est bien entendu le passage d'une mesure locale à une estimation du degré de contrôlabilité globale du graphe.

2.4. Limitations et Problèmes Ouverts

Une modélisation sous forme de durées plus ou moins possibles peut tout d'abord paraître artificielle. L'ensemble des durées possibles d'une tâche donnée ne se modélise en effet pas forcément "naturellement" sous la forme de trapèzes ... Se pose donc déjà le problème de la forme des données d'entrée, laquelle conditionne directement le degré de satisfaction du plan que l'on obtiendra.

Par ailleurs, ce que nous venons de présenter ne suffit pas en planification. Il faudrait pouvoir distinguer également des notions d'**utilité**, pour dire par exemple que telle ou telle contrainte doit être absolument satisfaite, car son non respect peut mettre en danger la sécurité du robot, ou de **préférence** [Fargier94], pour indiquer que l'on souhaite satisfaire une contrainte donnée, mais que le plan sera quand même accepté si tel n'était pas le cas. Limiter la durée totale du plan constitue par exemple généralement une préférence. Il s'agit là d'extensions aisées à mettre en place dans le cadre de la logique possibiliste, en considérant des exigences locales de degré de satisfaction spécifiques en plus d'une exigence globale.

Tout ceci peut faciliter par la suite la mise en place de techniques de **relaxation** de contraintes évoluées en cas d'échec, aspect qui n'a pas non plus été abordé dans le cadre de ce mémoire.

2.5. Des Contraintes mi-Contingentes, mi-Libres ...

Dans une autre perspective, on peut remarquer que la **dichotomie stricte** entre CTGs et CLBs apparaît dans la réalité comme étant parfois trop rigide. Certaines contraintes semblent vouloir se situer plus ou moins à la frontière entre les deux.

En premier lieu, nous sommes souvent confrontés dans le monde réel à des tâches dont la durée "de base" est contingente, mais qui peuvent être par contre **prolongées** autant qu'on le désire. Par exemple, pour me rendre au bureau le matin, je sais que si je prends le chemin le

plus court, et à une allure ne dépassant pas les limites imposées par le code de la route, cela me prendra entre dix minutes et un quart d'heure, suivant la circulation, et la couleur affichée par les divers feux de circulation au moment où j'arriverai à leur hauteur. Autant de choses qu'il m'est impossible de prévoir à l'avance. La tâche de déplacement jusqu'à mon lieu de travail constitue donc une CTG. Par contre, rien ne m'interdit si je le souhaite de rouler à 10 kms/h, ou de prendre des chemins de traverse, de manière à prolonger mon trajet, aussi longtemps que le niveau de mon réservoir d'essence me le permet.

De telles tâches peuvent être facilement représentées dans notre modèle sous la forme d'une succession de deux contraintes, une CTG G_1 et une CLB L_2. Par contre, une restriction importante doit être prise en compte: d'un point de vue sémantique, l'instant fin de G_1 et début de L_2 ne représente aucun événement réel. Aucune contrainte, de quelque type que ce soit, ne devra donc être ajoutée par la suite entre cet instant virtuel et un autre instant du graphe.

Evoquons également le cas de figure inverse. On peut admettre qu'une durée, contingente en principe, puisse être **réduite** si nécessaire dans des situations d'urgence. Par exemple, si j'ai un rendez-vous extrêmement important, je peux choisir finalement (à mes risques et périls) d'oublier pour un temps le code de la route, et de griller allégrement feux rouges, stops et limitations de vitesse ... Dans notre modèle, cela peut être géré

- durant la planification, en acceptant des plans susceptibles de violer de telles contraintes contingentes, c'est-à-dire en autorisant un degré de contrôlabilité locale relativement faible vis-à-vis de ces contraintes;

- durant l'exécution, en réagissant dès que l'on constate qu'une telle contrainte risque de faire échouer le plan (situation de "pire cas"): on peut alors modifier le mode d'exécution de la CTG, en forçant une restriction de son domaine, ou même en modifiant son statut par une mutation en CLB.

2.6. Application aux Problèmes de Fusion de Plan ...

Si l'on regarde de plus près le projet MARTHA abordé au sein du groupe, il est intéressant de constater qu'au niveau de la planification et de l'exécution distribuée des trajets des robots, on se trouve amené à gérer un problème de **fusion de plans**. En effet, chaque robot doit construire un plan d'action local pour réaliser son but (qui lui a par exemple été assigné par le module décrit au chapitre 5). Il doit alors non seulement vérifier que son plan est cohérent, mais également s'assurer qu'il est compatible avec le plan global, qui n'est autre que l'union des plans de tous les robots présents sur le site. Si tel est le cas, il est alors en mesure d'effectuer une opération de fusion incrémentale de son plan dans le plan global.

Une première ébauche d'étude de ce problème a été effectuée par M. Chater [Chater95], qui en a conclu qu'il pouvait être avantageux d'adapter le formalisme des STPUs. En effet, pour un robot $Robot_1$ par exemple, le graphe temporel correspondant au plan global est presque exclusivement composé de **contraintes temporelles contingentes**, puisqu'il ne contrôle ni la durée, ni le déclenchement des tâches de ses "congénères". Le problème est que du même coup, ce graphe temporel global ne contient quasiment pas de variables de Déclenchement, et se pose donc de nouveau le problème d'un ensemble de variables aléatoires interdépendantes. Il nous apparaît donc que l'utilisation des contraintes contingentes pour la fusion de plan en environnement multi-robots n'est pas envisageable directement. On peut par contre imaginer d'extraire du plan global les événements sur lesquels $Robot_1$ doit se synchroniser. Ces événements induiraient alors un ensemble restreint de contraintes contingentes bien identifiées que l'on chercherait alors à insérer dans le plan local de $Robot_1$. Il s'agirait en fait d'une sorte de **vérification de cohérence a priori**, qui permettrait ensuite une fusion du plan local dans le plan global en toute sécurité. Ceci n'est bien sûr qu'une idée de départ, qu'il conviendrait d'approfondir. En particulier, extraire du plan global les CTGs utiles à $Robot_1$ ne semble pas être une opération triviale ...

2.7. ... et à bien d'autres choses encore

Néanmoins, le problème précédent nous paraît extrêmement intéressant en tant qu'application de notre modèle à des problématiques appropriées dépassant le cadre de la seule planification mono-agent. Qui plus est, il semble évident que le **champ d'application** de ces techniques est extrêmement vaste, la prise en compte de la notion de contingence intéressant assurément tous les domaines, ou peu s'en faut, dans lesquels on doit raisonner sur des contraintes temporelles complexes, tels que le diagnostic, le langage naturel, la supervision, etc.

3. Exécution et Réactivité

Les chapitres 3 et 4 se sont surtout attachés à analyser les problèmes de gestion de contraintes temporelles lors de la phase de planification. Il serait également du plus grand intérêt de pouvoir effectuer un travail similaire au niveau de l'**exécution du plan produit**, où l'efficacité des algorithmes devient encore plus cruciale.

Le recours à une modélisation floue nécessite en particulier de mettre à jour en permanence pendant l'exécution le degré de satisfaction global du plan, ou pour être plus précis le degré de satisfaction des contraintes restant à exécuter. Ceci de manière à surveiller l'occurrence d'une

situation néfaste conduisant au **pire cas**, c'est-à-dire à l'échec du plan, ou au contraire à détecter un fonctionnement inespéré, en "**meilleur cas**" en quelque sorte, qui permet d'envisager une optimisation opportuniste de l'exécution du plan. Ces deux aspects, illustrant le problème de la gestion par **anticipation** des **incertitudes liées à l'exécution** [Berry92] (cf chapitre 5, § 1.1), nécessitent un retour d'information pertinente au planificateur pour guider la phase de **replanification** qui s'avère alors nécessaire.

A côté de cela, il convient également de réfléchir aux capacités de **réaction** du système de supervision de plan (cf chapitre 2, § 2.1) à un dysfonctionnement, tel qu'une panne ou l'occurrence d'un événement imprévu. Pour gérer **en temps réel** ces **incertitudes liées à l'exécution** [Berry92] (cf chapitre 5, § 1.1), il conviendrait d'être capable de vérifier de manière quasi-instantanée si le graphe reste malgré tout cohérent, et dans le cas contraire, de déclencher des **actions de réparation d'urgence** avant d'envisager là aussi une phase de **replanification**.

4. Expérimentation Grandeur Nature

Une dernière catégorie de prospectives, quoiqu'évidente, ne doit pas être oubliée. Les travaux présentés dans ce mémoire sont avant tout des travaux **théoriques**, qui se doivent d'être complétés par une confrontation avec le réel. Pour cela, il est tout d'abord indispensable d'**intégrer** le Graphe de Décision dans le planificateur IxTeT afin de le tester sur des simulations empruntées à des exemples réels de planification. A plus long terme, et je parle là aussi bien du planificateur IxTeT que du module d'allocation de ressources présenté au chapitre 5, il serait du plus grand intérêt de constater l'applicabilité de ces systèmes à des **expérimentations complètes de robotique**. L'infrastructure du groupe nous semble parfaitement appropriée à de telles réalisations dans un avenir plus ou moins proche.

Conclusion Générale

En raisonnement temporel, les approches basées sur des **modules séparés de gestion de contraintes** sont désormais monnaie courante. Quel que soit le domaine d'application visé, de tels modules disposent généralement de prérogatives similaires. La tâche qui leur est dévolue est double:

(1) gérer les **mises-à-jour**, tout en maintenant la **cohérence** de l'ensemble,
et (2) fournir une réponse à toute **requête** provenant du système global.

La distinction entre contraintes **symboliques** et **numériques** prévaut également dans de nombreux domaines d'application. Elle est en effet représentative de la nature même du temps, vis-à-vis duquel s'expriment aussi bien des notions d'antériorité que de positionnement temporel précis.

Les choix en termes de compromis entre pouvoir expressif et efficacité des techniques de propagation de contraintes varient par contre d'un domaine à l'autre. Notre première contribution a consisté à développer une technique de gestion des contraintes temporelles numériques dans le cadre particulier de la **planification**. La fréquence importante de requêtes, l'exigence extrême en termes de complétude, mais aussi d'efficacité, et enfin la faible proportion de contraintes numériques nous ont amené à définir le paradigme du Sous-Graphe Numérique (chapitre 3). Il s'agit là de toute évidence d'une méthode qui reste étroitement liée au domaine dont elle est issue. Néanmoins, elle nous a permis de mettre en avant certains aspects propres au raisonnement temporel dans sa généralité, qui méritaient d'être regardés de plus près. En effet, la nécessité pour des raisons de complexité de **décomposer** notre modèle en sous-graphes virtuels maintenant des **contraintes hétérogènes** (symboliques, numériques et d'inégalité) à

l'aide d'algorithmiques spécifiques, a permis de caractériser finement les diverses **interactions** suscitées. Il a notamment été possible de montrer clairement l'importance que pouvaient avoir les **précédences induites** par la propagation numérique.

Après quoi l'objectif principal a consisté à s'intéresser aux problèmes d'**incertitude temporelle**. En effet, on représente souvent les durées et les dates sous forme d'intervalles de valeurs possibles, mais sans véritablement appréhender le degré d'incertitude qu'une telle représentation peut sous-entendre. Il y avait là une carence que nous avons cherché à combler, au moins en partie. Cela s'est traduit d'une part par la distinction faite entre contraintes contingentes et libres (chapitre 4), et d'autres part par l'analyse de la pertinence d'heuristiques temporelles en présence de telles incertitudes (chapitre 5). Dans les deux cas, nous avons été amenés à adapter ou préciser certaines techniques classiques de manière à ce qu'elles tiennent compte de l'incertitude. On peut tirer deux enseignements, parmi d'autres, de ces études.

- Tout d'abord, prendre en compte l'incertitude de manière explicite se fait généralement **sans modification ni du principe, ni des performances globales** du système, comme nous l'avons vu notamment en planification dans le cadre du Graphe de Décision. Il s'agit juste de modifier la définition du problème initial, et d'adapter les structures et techniques existantes à ces nouvelles définitions.

- Ensuite, il devient vite absurde de chercher à prendre une décision fiable (cf chapitre 5) et pertinente (cf chapitre 6) en présence d'incertitudes. C'est dire que l'incertitude des données d'entrée conduit généralement à une modulation des techniques de résolution. On parle alors de degré de satisfaction, ou de degré d'optimalité d'une solution, ou encore de **qualité** de la solution prodiguée, lesquelles dépendent bien entendu des degrés d'incertitude en entrée.

Ces réflexions nous amènent à considérer l'utilisation de techniques telles que la logique floue ou les distributions probabilistes comme incontournables dès lors que l'on souhaite appréhender correctement les problèmes temporels entachées d'incertitude. C'est là le pas qu'il reste à franchir dans notre travail de recherche, pour parvenir à une prise en compte réellement satisfaisante.

Signalons pour finir que nous sommes intimement persuadés que les techniques développées dans ce mémoire dans le cadre de la gestion de l'incertitude ne sauraient être limitées au cadre strict d'un domaine ou d'un autre. Elles abordent en fait des problèmes clés du raisonnement temporel dans sa généralité. En particulier, la dichotomie entre ce qui est libre et ce qui est contingent n'est pas propre à la planification. Au contraire, la complexité des contraintes temporelles manipulées dans notre Graphe de Décision en font un outil expressif et performant susceptible d'être utilisé dans **d'autres domaines d'application**.

Car l'enjeu de ces nouveaux modèles de raisonnement, qu'ils s'inscrivent dans la lignée de notre approche ou de celle de l'équipe de D. Dubois à l'IRIT, qui de notre point de vue sont complémentaires, dépasse largement le cadre de tel ou tel domaine de prédilection. Pendant longtemps, on a cru que le rôle d'un système automatisé était de fournir une **réponse précise** à la question qui lui était posée. On sait maintenant que la machine, si elle peut être plus rapide ou plus économique, ne saurait connaître autre chose que ce que l'homme lui-même pourrait déduire. En l'absence d'informations suffisantes, il n'y a tout simplement pas de réponse précise. Ce qui n'empêche nullement l'homme de raisonner et de progresser vers une solution plus ou moins satisfaisante. En d'autres termes, si les machines veulent sortir des usines, des ateliers et des laboratoires pour évoluer dans un **monde réel,** donc nécessairement "chaotique", elles doivent s'attendre à ce que leurs demandes d'informations se heurtent à des "*Peut-être*" et à des "*Plus ou moins*". A elles alors d'être capables de s'en contenter, de **raisonner** quand même et de répondre à la question posée "*Oui*" ou "*Non*", le cas échéant, mais aussi "*Sans doute*" ou "*Attendez, je vais voir ...*" La communauté scientifique est depuis de nombreuses années déjà tendue vers ce but, mais le chemin est encore long et visiblement passionnant.

Terminons par un clin d'oeil, sous la forme d'une question que d'aucuns pourraient bien s'aventurer à poser au vu des élucubrations qui précèdent: "mais alors, les machines seront-elles un jour aussi ***intelligentes*** que l'homme ?" Je me contenterais de répondre qu'elles seront surtout plus autonomes et sans doutes plus efficaces (tant il est vrai que l'exigence de certitudes finit généralement par être source d'inaction), et que leur champ d'action sera plus vaste. Et puis, pour reprendre les termes d'Edgar Dijkstra, "*la question de savoir si une machine peut penser est à peu près aussi pertinente que de savoir si un sous-marin sait nager*" ...

Références Bibliographiques

- [Aguilar95] L. Aguilar, R. Alami, S. Fleury, M. Herrb, F. Ingrand, F. Robert - *Ten Autonomous Mobile Robots (and even more) in a Route Network Like Environment.* Proc. IROS'95, Pittsburgh, USA, Août 1995.

- [Allen83] J.F. Allen - *Maintaining Knowledge about Temporal Intervals.* Communications of the ACM, vol. 26, n. 11, Novembre 1983 - pp.832-843.

- [Allen84] J.F. Allen - *Towards a General Theory of Action and Time.* Artificial Intelligence, 23, 1984 - pp. 123-154.

- [Badaloni94] S. Badaloni & M. Berati - *Dealing with Time Granularity in a Temporal Planning System.* Proc. 1rst Int. Conf. on Temporal Logic ICTL'94, Bonn, Germany, Juillet 1994.

- [Barber93] F. Barber - *A Metric Time-Point and Duration-Based Temporal Model.* SIGART Bulletin, Vol. 4, n. 3, 1993 - pp. 30-49.

- [Bernard92] D. Bernard - *Un système de Raisonnement Temporel basé sur le Calcul d'Evénements pour l'Ordonnancement en Maintenance Aéronautique.* Thèse de Doctorat de l'Université Paul Sabatier, Juillet 1992, Toulouse.

- [Berry92] P. M.Berry, B. Y.Choueiry & L. Friha - *Multi-Agent Architecture for a Dstributed Approach to Resource Allocation using Temporal Abstractions.* Technical Report, EPFL, TR-92/18, 1992.

- [Bessière92] C. Bessière - *Systèmes à Contraintes Evolutifs en Intelligence Artificielle.* Thèse de Doctorat de l'Université de Montpellier II, Septembre 1992

- [Boddy93] M. Boddy - *Temporal Reasoning for Planning and Scheduling.* SIGART Bulletin, 4(3), 1993.

- [Borillo90] M.Borillo & B.Gaume - *Une Extension Cognitive du Calcul d'Evénement de KOWALSKI et SERGOT et son Application au Raisonnement Spatio-Temporel.* Revue d'Intelligence Artificielle, vol. 4, n.4, 1990 - pp. 7-26.

- [Brusoni94] V. Brusoni, L. Console, B. Pernici, P. Terenziani - *LaTeR: a General Purpose Manager of Temporal Information.* In Methodologies for Intelligent Systems 8, Lecture Notes in Computer Science 869, Springer Verlag, 1994, pp. 255-264.

- [Chapman87] D. Chapman - *Planning for Conjunctive Goals.* Artificial Intelligence, vol. 32, 1987 - pp. 333-377.

- [Chater95] M. Chater - *De la Planification Incrémentale vers la Planification Multi-Robot.* Rapport de Stage DEA, Paris-IX Dauphine / LAAS-CNRS, Septembre 1995.

- [Choueiry94] B. Choueiry - *Abstraction Methods for Resource Allocation.* Thèse de Doctorat, Ecole Polytechnique Fèdèrale de Lausanne, Suisse, Novembre 1994.

- [Collinot87] A. Collinot & C. Le Pape - *Controlling Constraint Propagation.* Proc. 10th IJCAI, Milan (It) 1987.

- [Collinot88] A. Collinot, C. LePape, G. Pinoteau - *SONIA: A Knowledge-Based Approach to Industrial Job-Shop Scheduling.* International Journal for Artificial Intelligence in Engineering, 3(2), 1988.

- [Console93] L. Console, P. Torasso. *Temporal Constraint Satisfaction on Causal Models,* Information Sciences, vol. 68 (1),1993, pp. 1-32.

- [Dean86] T. Dean - *Intractability and Time-Dependent Planning.* "Reasoning About Actions and Plans", 1986.

- [Dean87] T. Dean & D.V. McDermott - *Temporal Data Base Management.* Artificial Intelligence, 32, 1987 - pp.1-55.

- [Dean89] Thomas Dean - *Using Temporal Hierarchies to Efficiently Maintain Large Temporal Databases*. Journal of the Association for Computing Machinery, vol.36, n.4, Octobre 1989 - pp.687-718.

- [Dechter88] R. Dechter & J. Pearl - *Network-Based Heuristics for Constraint Satisfaction Problems*. Artificial Intelligence, vol. 34, pp. 1-38.

- [Dechter91] R. Dechter, I. Meiri & J. Pearl - *Temporal Constraint Networks*. Artificial Intelligence, 49, 1991 - pp. 61-95.

- [Dorn92] J. Dorn - *Temporal Reasoning in Sequence Graphs* - Proc. 10th AAAI, San Jose (CA), 1992 - pp. 735-740.

- [Dorn93] J. Dorn, R. Kerr & G. Thalhammer - *Reactive Scheduling in Fuzzy-Temporal Framework* - Tech Report CD-TR 93/55, Technische Universität Wien, 1993.

- [Dorn94] J. Dorn - *Hybrid Temporal Reasoning*. In Proc. 11th ECAI, Amsterdam, Pays-Bas, 1994 - pp. 625-629.

- [Dousson94] C. Dousson - *Suivi d'Evolution et Reconnaissance de Chroniques*. Thèse de Doctorat de l'Université Paul Sabatier, Septembre 1994, Toulouse.

- [Dubois89] D. Dubois & H. Prade - *Processing Fuzzy Temporal Knowledge* - IEEE Transactions on Systems, Man and Cybernetics, vol. 19, n. 4, Juillet.Août 1989.

- [Dubois93] D. Dubois, H. Fargier & H. Prade - *The Use of Fuzzy Constraints in Job-Shop Scheduling* - Proc. IJCAI-93 Workshop on Knowledge-Based Planning, Scheduling and Control. Chambery(France), 1993.

- [Erschler91] J. Erschler, P. Lopez & C. Thuriot - *Raisonnement Temporel sous Contrainte de Ressource et Problemes d'Ordonnancement*. Revue d'Intelligence Artificielle, vol. 5, n. 3, 1991.

- [Fargier94] H. Fargier - *Problèmes de Satisfaction de Contraintes Fllexibles - Application à l'Ordonnancement de Production* - Thèse de Doctorat de l'Université Paul Sabatier, Juin 1994, Toulouse.

- [Fargier95] H. Fargier, J. Lang & T. Schiex - *Mixed Constraint Satisfaction: a Framework for Decision Problems under Incomplete Knowledge* - Proc. IJCAI'95, Montreal, Canada, Août 1995.

- [Fox90] M.S. Fox & N. Sadeh - *Why is Scheduling Difficult ? A CSP Perspective.* Proc. AAAI 90, 1990.

- [French82] S. French - *Sequencing & Scheduling: an Introduction to the Mathematics of Job-Shop.* Whiley, 1982.

- [Fikes71] R.E. Fikes & N.J. Nilsson - *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.* Proc. of 2nd IJCAI, Septembre 1971 - pp.189-208.

- [Freuder82] E. C. Freuder - *A Sufficient Condition for Backtrack-Free Search.* Journal of the ACM , vol.29, n 1, Janvier 82 - pp. 24-32.

- [Garcia95] F.Garcia & P.Laborie - *Hierarchisation of the Search Space in Temporal Planning.* Proc. EWSP-95, Assise, Italie, Septembre 1995 - pp. 235-249.

- [Gerevini93] A.Gerevini, L.Schubert & S.Schaeffer- *Temporal Reasoning in Timegraph I-II.* SIGART Bull., 4(3), 1993 - pp. 21-25.

- [Ghallab89] M. Ghallab & A. Mounir-Alaoui - Relations Temporelles Symboliques: Représentations et Algorithmes. Revue d'Intelligence Artificielle, vol. 3, n. 3, 1989.

- [Hertzberg91] J. Hertzberg, S. Materne & H. Voβ - *On Clipping Persistence (or whatever must be clipped) in Time Maps.* Abeitspapiere der GMD, Juin 1991, Sankt Augustin.

- [Isli94] A. Isli - *Constraint-Based Temporal Reasoning: a Tractable Point-Based Algebra Combining Qualitative, Metric and Holed Constraints.* Rapport technique LIPN, URA CNRS 1507, Décembre 1994.

- [Kaufmann69] A. Kaufmann & G. Desbazeille - *La Méthode du Chemin Critique: application aux programmes de production et d'études de la méthode PERT et de ses variantes.* éd. Dunod, Paris, 1969.

- [Kautz91] H.A.Kautz & P.B.Ladkin - *Integrating Metric and Qualitative Temporal Reasoning.* Proc. 9th AAAI, Anaheim, CA, 1991 - pp. 241-246.

- [Kerr89] R.M. Kerr & R.N. Walker - *A Job-Shop Scheduling System Based on Fuzzy Arithmetic.* Proc. 3rd Int. Conf. on Expert Systems, Hilton Head Island, USA, Mai 1989.

- [Koomen88] J.Koomen - *The TIMELOGIC Temporal Reasoning System.* Technical Report 231, Univ. of Rochester, Dept. of Computer Science, 1988.

- [Kowalski86] R. Kowalski & M. Sergot - *A Logic-Based Calculus of Events*. New Generation Computing, 3, 1986.

- [Laborie94] P. Laborie - *Planification et Gestion de Ressources*. Rapport Technique, LAAS-CNRS, Toulouse, France, 1994.

- [Laborie95a] P. Laborie & M. Ghallab - *Planning with Sharable Resource Constraints*. Proc. IJCAI'95, Montréal, Canada, Août 1995.

- [Laborie95b] P. Laborie & M. Ghallab - *IxTeT: an Integrated Approach for Plan Generation and Scheduling*. INRIA/IEEE Conf. on Emerging Technologies and Factory Automation, Paris, France, Octobre 1995.

- [Laruelle94] H.Laruelle- *Planification Temporelle et Exécution de Tâches en Robotique*. Thèse de Doctorat de l'Université Paul Sabatier, Avril 1994, Toulouse.

- [LePape90] C. Le Pape - *A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling*. Proc. IEEE Robotics and Automation, 1990.

- [LePape94] C. Le Pape - *Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems*. «Intelligent Systems Engineering», 1994.

- [Ligozat91] G. Ligozat - On Generalized Interval Calculi. Proc. 9th AAAI, Anaheim, CA, 1991 - pp. 234-240.

- [Mackworth77] A.K. Mackworth - *Consistency in Networks of Relations*. Artificial Intelligence, 8, 1977 - pp.99-118.

- [Mackworth85] A.K.Mackworth & E.C.Freuder - *The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems*. Artificial Intelligence, 25(1), 1985 - pp. 65-74.

- [McAllester91] D. McAllester & D. Rosenblitt - *Systematic NonLinear Planning*. Proc. 9th AAAI, Anaheim, CA, 1991.

- [McDermott82] D.V. McDermott - A *Temporal Logic for Reasoning about Processes and Plans*. Cognitive Science, 6, 1987 - pp.101-155.

- [Meiri91] I.Meiri - *Combining Qualitative and Quantitative Constraints in Temporal Reasoning*. In Proc. 9th AAAI, Anaheim, CA, 1991 - pp. 260-267.

- [Miller85] D. Miller, R. James Firby & T. Dean - *Deadlines, Travel Time and Robot Problem Solving*. Proc. of 9th IJCAI 1985 - pp 1052-1054.

- [Montanari,74] U. Montanari - *Networks of Constraints: Fundamental Properties and Applications to Picture Processing*. Information Sciences, 7, 1974 - pp.95-132.

- [Mounir90] A. Mounir-Alaoui - *Raisonnement Temporel pour la Planification et la Reconnaissance de Situations*. Thèse de Doctorat de l'Université Paul Sabatier, Octobre 1990, Toulouse.

- [Nebel94] B.Nebel & H.J.Bürckert - *Reasoning about Temporal Relations: a Maximal Tractable Subclass of Allen's Interval Algebra*. In Proc. 12th AAAI, Seattle, WA, 1994 - pp. 356-361.

- [Poesio91] M. Poesio & R.J. Brachman - *Metric Constraints for Maintainaing Appointments: Dates and Repeated Activities*. Proc. 9th AAAI, Anaheim, CA, 1991 - pp. 253-259.

- [Prosser94] P. Prosser & I. Buchanan - *Intelligent Scheduling: Past, Present and Future*. Intelligent Systems Engineering, vol. 3, n. 2, été 1994.

- [Régnier92] P. Régnier - *Recherche et Exploitation du Parallélisme en Planification*. Thèse de Doctorat de l'Université Paul Sabatier, Mai 1992, Toulouse.

- [Rit88] JF. Rit - *Modélisation et Propagation de Contraintes Temporelles pour la Planification*. Thèse de Doctorat de l'INPG, Mars 1988, Grenoble.

- [Rutten93] E.Rutten & J.Hertzberg - *Temporal Planner = Nonlinear Planner + TimeMap Management*. AI COM, 6(1), 1993 - pp.18-26.

- [Sadeh89] N. Sadeh & M.S. Fox - *Preference Propagation in Temporal/ Capacity Constraint Graphs*. Rapport Technique CMU-TR-TR-89-2, Carnegie Mellon University, Pittsburgh, USA, Janvier 1989.

- [Sadeh91] N. Sadeh - *Look-Ahead Techniques for Micro-Opportunistic Job-Shop Scheduling*. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, USA, Mars 1991.

- [Shoham87] Y. Shoham - *Temporal Logics in A.I.: Semantical and Ontological Considerations*. Artificial Intelligence 33, 1987 - pp. 89-104.

- [Steffen86] M.S. Steffen - *A Survey of A.I.-based Scheduling Systems*. Fall Industrial Engineering Conference, Boston (MA), USA, 1986.

- [Stillman93] J. Stillman, R. Arthur & A. Deitsch - *TACHYON: a Constraint-Based Temporal Reasoning Model and its Implementation*. SIGART Bull., Vol. 4, n. 3, 1993 - pp. T1-T4.

- [Tolba91] H. Tolba, F. Charpillet & JP. Haton - *Representing and Propagating Constraints in Temporal Reasonning*. IEEE- Int. Conf. on Tools for A. I., 1991 - pp.181-184.

- [Tolba92] 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 de Nancy I, Décembre 1992.

- [Tsang87a] E. Tsang - *Time Structures for A.I.* Proc. IJCAI'87, Milano, Italie, 1987 - pp. 456-461.

- [Tsang87b] E. Tsang & R. Howarth - *Scheduling in both Space and Time*. 11th Int. Conf. on Expert Systems & their Applications, vol.3, 1991, Avignon - pp.361-372.

- [vanBeek,89] P. van Beek - *Approximation algorithms for temporal reasonning*. Proc. 11th IJCAI, 1989, Detroit - pp.1292-1296.

- [vanBeek91] P.vanBeek - *Temporal Query Processing with Indefinite Information*. Artificial Intelligence in Medicine, vol. 3, 1991 - pp. 325-339.

- [vanBenthem83] J. van Benthem - *The Logic of Time*. Kluwer Academic, Dordrecht, 1983.

- [Vere83] S. A. Vere - *Planning in Time: Windows and Durations for Activities and Goals*. IEEE-PAMI, 5(3), Mai 1983- pp.246-267.

- [Vidal94] T. Vidal, M. Ghallab & R. Alami - *Dynamical allocation of predefined tasks to a large team of robots*. Rapport Technique LAAS n.94303, Septembre 1994.

- [Vidal95] T.Vidal & M.Ghallab - *Efficient Temporal Management through an Application-Dependent Graph Decomposition*. Proc. TIME'95, Melbourne Beach (FL), USA, Avril 1995.

- [Vila93a] Ll.Vila & H.Reichgelt - *The Token Reification Approach to Temporal Reasoning.* Tech. Rep. 1/93, Dept. of Computer Science, UWI.

- [Vila93b] Ll.Vila - *Instants, Periods and the Divided Instant Problem.* Proc. of QUARDET'93, IMACS.

- [Vila94a] Ll.Vila - *A Survey on Temporal Reasoning in Artificial Intelligence.* AI COM, Vol. 7, n. 1, 1994 - pp 4-28.

- [Vila94b] Ll. Vila - *Re-formulating Event Calculus in Terms of Metric Temporal Constraints* (communication personnelle), IIIA-CEAB, Blanes (Catalonia, Spain), 1994.

- [Vilain86] M.Vilain & H.A.Kautz - *Constraint Propagation Algorithms for Temporal Reasoning.* In Proc. AAAI, Philadelphia, PA, 1986 - pp. 377-382.

- [Vilain89] M.Vilain, H.A.Kautz & P.vanBeek - *Constraint Propagation Algorithms: a Revised Report.* Readings in Qualitative Reasoning about Physical Systems, Morgan Kaufman, 1989..

- [Wallace93] M. Wallace - *Constraints in Planning, Scheduling and Placement Problems.* Working Paper ECRC-Report CORE-93-6, 1993.

- [Wilkins88] D.E. Wilkins - *Practical Planning: Extending the Classical A.I. Paradigm.* Morgan Kaufman, 1988.

- [Williamson93] M.Williamson & S.Hanks - *Exploiting Domain Structure to Achieve Efficient Temporal Reasonning.* In Proc. 13th IJCAI, Chambery (France) 1993.

- [Yampratoom93] E. Yampratoom & JF. Allen - *Performance of Temporal Reasoning Systems.* SIGART Bulletin, Vol. 4, n. 3, 1993 - pp. 26-29.

Table des Matières

Liste des Figures

Liste des Tableaux

Le Temps en Planification et en Ordonnancement:
Vers une gestion complète et efficace de contraintes hétérogènes et entachées d'incertitude

Mots-Clés: Raisonnement temporel - Satisfaction de contraintes - Incertitudes - Planification - Ordonnancement

Le planificateur temporel IxTeT, véritable "intelligence embarquée" d'un robot autonome, s'appuie sur un gestionnaire de contraintes temporelles dont le rôle est double: maintenir la cohérence du réseau de contraintes, et répondre rapidement et correctement à une interrogation émanant du planificateur. La prise en compte de contraintes symboliques (simples précédences) et numériques (dates et durées imprécises) oblige alors à une propagation coûteuse des contraintes numériques. Néanmoins, ces dernières étant proportionnellement peu nombreuses en planification, nous pouvons restreindre cette propagation à un sous-graphe numérique.

Par ailleurs, nous devons tenir compte en planification de durées "contingentes", dont la valeur est aléatoire. Ces incertitudes nous obligent à redéfinir la notion classique de cohérence, et à nous ramener à un modèle temporel dual, appelé Graphe de Décision, dans lequel nous pouvons utiliser les algorithmes classiques de propagation.

Le mémoire s'achève par la présentation d'une application distincte relevant du domaine de l'ordonnancement de tâches pour un ensemble de robots. L'incertitude pesant sur les contraintes numériques oblige ici à allouer les ressources au fur et à mesure de l'exécution. Les caractéristiques propres à l'application suggèrent une décomposition du graphe temporel conduisant à une efficacité optimale des algorithmes de propagation.

Time in Planning and Scheduling:
Towards a complete and efficient management of heterogeneous and uncertain constraints.

Keywords: Temporal reasoning - Constraint satisfaction - Uncertainties - Planning - Scheduling

The IxTeT planning system, dedicated to autonomous robots, relies upon a temporal constraint manager, addressing both symbolic (simple precedence) and numerical (dates and durations) constraints. Its purpose is to maintain the consistency of the constraint network and give swift and sound answers to temporal queries made by the planner. Those requirements compel us to use a costly numerical constraint propagation process. Nevertheless, numerical constraints being rather scarce in planning applications, we can restrain this propagation to a numerical subgraph.

Moreover, we must pay attention in planning applications to "contingent" durations, whose values are unpredictable. Such uncertainties lead to a new characterization of the network consistency and to the proposal of a dual temporal model, called the Decision Graph, where we can use classical propagation processes.

A distinct task scheduling application with several robots is presented in the last part. Numerical constraint uncertainty requires the allocation of resources in a sliding short-term horizon, as the execution process runs. Fortunately, an application-dependent graph decomposition technique allows us to get very efficient propagation algorithms.