Année 1995

THESE

préparée au

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

en vue de l'obtention du

Doctorat de l'École Nationale Supérieure des Télécommunications

Spécialités : Intelligence Artificielle et Robotique

par

Philippe LABORIE

Ingénieur ENST

IxTeT : UNE APPROCHE INTÉGRÉE POUR LA GESTION DE RESSOURCES ET LA SYNTHÈSE DE PLANS

Soutenue le 14 décembre 1995 devant le jury composé de :

Georges	GIRALT	Président
Malik	GHALLAB	Directeur de thèse
Marie-Odile	CORDIER	Rapporteurs
Jean-Paul	HATON	
Jacques	ERSCHLER	Examinateurs
Henri	FARRENY	
Frédérick	GARCIA	
Alain	GRUMBACH	
Jean-Paul	KRIVINE	

Cette thèse a été préparée au Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS
7, avenue du Colonel Roche 31077 Toulouse Cedex

Avant-Propos

Le travail de recherche décrit dans ce mémoire est avant tout le fruit de son environnement. Il n'est certainement pas une idée exposée ici que je ne doive, directement ou non, à une discussion avec un collègue ou un ami, au croisement d'idées exposées par d'autres, à leur soutien. Le reste n'est que mise en forme : de la diversité naît la vrai richesse. Aussi je voudrais exprimer ici ma profonde gratitude à tous ceux qui ont contribué - par leur confiance, leurs compétences, leur compréhension, leur amitié ou leur amour - à ce travail qui est aussi le leur.

Je tiens tout d'abord à remercier MM Alain Costes, directeur du LAAS, Georges Giralt et Malik Ghallab, directeurs du groupe de recherche en Robotique et Intelligence Artificielle pour m'avoir acceuillis parmi eux. Au cours de mon séjour, Malik fut bien plus que mon directeur de thèse et je voudrais souligner, par delà les mots, combien j'ai été sensible à ses qualités humaines, à sa disponibilité en toute occasion et à ses judicieux conseils.

Tout travail ne prend sens que lorsqu'il est analysé et éclairé par un regard extérieur, objectif et avisé. Que tous les membres de mon jury, Marie-Odile Cordier, Jacques Erschler, Henri Farreny, Frédérick Garcia, Malik Ghallab, Georges Giralt, Alain Grumbach, Jean-Paul Haton, Jean-Paul Krivine soient assurés, à ce titre, de ma reconnaissance pour l'attention particulière qu'ils ont bien voulu porter à cette thèse.

Je remercie également tous les collaborateurs du projet PADRE: Emmanuel Robinet et Christophe Lansade à IXI, Arnaud Robert de Saint Vincent et Yann Parrod à MMS, Malik Ghallab et Félix Ingrand au LAAS qui ont fourni à ce travail un cadre de recherche et d'applications industrielles particulièrement riche et motivant.

C'est grâce à de multiples échanges, discussions et collaborations au sein de "l'équipe IxTeT" dirigée par Malik que les idées développées dans ce mémoire ont pu voir le jour. Amine Mounir-Alaoui, Patrick Albers, Christophe Dousson, Frédérick Garcia, Gérome Gout, Hervé Laruelle, Thierry Vidal : cette thèse est aussi en partie la leur ; le fait de leur enthousiasme, de leur esprit d'initiative et de leur bonne humeur.

Ma nature quelque peu distraite craint trop d'oublier des noms pour citer tous ceux qui, stagiaires, thésards ou permanents au sein du groupe RIA savent conserver cette cohésion et cette atmosphère chaleureuse que je n'ai retrouvé nulle part ailleurs. Je sais qu'ils se reconnaîtront, les amateurs de café serré et les autres ...

A mes meilleurs copains, Nicolas, Christophe, Béatrice,

A Cathy, *à travers nos différences nous nous ressemblons tant*

A Maman, *je dédie ce travail à ton amour, à ton courage*

A Marina, *je pense à un acrostiche sur ton prénom*

Aux trois dames de mes pensées

Table des matières

Table des matières **7**

Liste des figures **13**

Chapitre 1 Quelques travaux en planification **21**

1.1 Introduction 21

1.2 Synthèse de plans 21

1.2.1 Recherche dans l'espace d'états 23

1.2.2 Recherche dans l'espace des plans partiels 24

1.2.3 Vers un formalisme plus riche 27

1.2.3.1 Variables 27

1.2.3.2 Planification temporelle 28

1.2.3.3 Autres extensions 30

1.3 Affectation de ressources 30

1.4 Ordonnancement 32

1.4.1 Les problèmes d'ordonnancement 32

1.4.2 Quelques approches de résolution 35

1.4.2.1 Approches issues de la RO 35

1.4.2.2 Approches liées à l'IA 36

1.5 Les travaux d'intégration 37

1.6 Positionnement de notre approche 40

Chapitre 2 Représentation **43**

2.1 Introduction 43

2.2 Le temps 43

2.2.1 L'approche symbolique 44

2.2.2 L'approche métrique 46

2.2.3 Représentation du temps dans IxTeT 48

2.3 Les variables atemporelles 50
2.3.1 Variables et contraintes 50
2.3.2 Algorithmes de propagation 52
2.4 Les attributs d'état 55
2.4.1 L'état du monde 55
2.4.2 La persistance et le changement 57
2.5 Les ressources et leur utilisation 57
2.5.1 Qu'est-ce qu'une ressource? 57
2.5.2 Utilisation des ressources 59
2.6 Le formalisme de tâches 61
2.6.1 base de connaissances temporelles IxTeT 61
2.6.2 Les opérateurs de planification 62
2.6.3 Le scénario initial 63
2.7 Comparaison avec le formalisme STRIPS 64
2.7.1 Le formalisme STRIPS 65
2.7.2 Le prédicat *assert* 65
2.8 Conclusion 68

Chapitre 3 Contrôle de base **69**

3.1 Introduction 69
3.2 Le critère de cohérence d'état 70
3.2.1 Définition de la cohérence d'état 70
3.2.2 Un critère de cohérence local 72
3.3 Le critère de disponibilité de ressource 73
3.3.1 Définition de la cohérence de ressource 74
3.3.2 La représentation à base de "use" 75
3.3.3 Un critère de cohérence local 76
3.4 Problème de planification 78
3.5 L'algorithme global : défauts et résolvantes 78
3.5.1 Résolvantes d'une proposition non-expliquée 79
3.5.2 Résolvantes d'une menace 80
3.5.3 Résolvantes d'un ensemble critique minimal 81
3.5.4 Bilan 85
3.5.5 L'algorithme non-déterministe de contrôle 85
3.6 Les choix non-déterministes 87

3.6.1 La stratégie de moindre engagement 87
3.6.1.1 Engagement lié à une contrainte de précédence 90
3.6.1.2 Engagement lié à une contrainte de différence 91
3.6.1.3 Engagement lié à une contrainte d'égalité 91
3.6.1.4 Engagement lié à l'insertion d'un lien causal 91
3.6.2 Une recherche opportuniste 92
3.7 Analyse des sous-buts 94
3.8 Analyse des menaces 95
3.9 Filtrage des résolvantes 97
3.9.1 L'approche utilisée dans WATPLAN 97
3.9.2 Filtrage des résolvantes sur IXTET 99
3.10 Exploration de l'arbre de recherche 99
3.11 Conclusion 100

Chapitre 4 Gestion de ressources partageables 103

4.1 Introduction 103
4.2 Le problème de base 104
4.3 Graphes d'intersections possibles 105
4.4 Recherche d' ensembles critiques minimaux 108
4.4.1 L'arbre de recherche des ECM 109
4.4.1.1 Principe de la recherche des ECM 109
4.4.1.2 Cadre formel de la recherche des ECM 111
4.4.2 Procédures de développement d'un nœud 112
4.4.2.1 Une variante systématique de l'algorithme glouton 112
4.4.2.2 DECLIC : Vers une contraction de l'espace de recherche . . . 114
4.4.3 Exploration de l'arbre de recherche 115
4.4.3.1 Test de consistance nécessaire de la base 116
4.4.3.2 Recherche d'ECM particuliers dans IXTET 117
4.4.3.3 Complexité de la recherche avec l'algorithme DECLIC 117
4.5 Minimisation des résolvantes d'un ECM 119
4.5.1 Notion de minimalité d'une contrainte temporelle 119
4.5.2 Application aux contraintes de non-intersection 122
4.6 Allocation de ressources partageables 123
4.7 Cas des ressources productibles 125
4.8 Bilan et conclusion 127

Chapitre 5 Hiérarchisation de la recherche **131**

5.1 Introduction 131
5.2 Hiérarchies d'abstraction 132
5.3 La hiérarchie de moindre engagement d' IXTET 134
5.3.1 Hiérarchie d'abstraction pour IXTET 135
5.3.2 Une approche de moindre engagement 136
5.4 Quelques améliorations de la hiérarchie 140
5.4.1 Effets principaux et secondaires 140
5.4.2 Prise en compte du problème à résoudre 141
5.4.3 Vers une hiérarchie plus fine 143
5.5 Implémentation, résultats et limitations 145
5.5.1 Génération automatique de la hiérarchie 145
5.5.2 Gestion de la hiérarchie d'abstraction 146
5.6 Conclusion 148

Chapitre 6 Exemples, résultats et analyses **151**

6.1 Introduction 151
6.2 Un exemple complet : COLUMBUS 151
6.3 Autres exemples 157
6.3.1 ATELIER 157
6.3.2 ARIANE 158
6.3.3 FINITION PIECE 160
6.3.4 GAMMES ALTERNATIVES 162
6.3.5 INTREPIDE 163
6.4 Analyse des différentes stratégies utilisées 165
6.5 Conclusion 166

Annexe A Principales notations utilisées **172**

Annexe B Eléments de Théorie des Graphes **174**

B.1 Quelques notions générales sur les graphes 174
B.2 Graphes faiblement triangulés 175
B.3 Graphes triangulés 176
B.4 Graphes de comparabilité 177
B.5 Graphes d'intervalles 178
B.6 Bilan 178

Annexe C Eléments sur les C.S.P. **180**

C.1 Définitions 180

C.2 Filtrage et propagation de contraintes 182

C.2.1 Consistance d'arc 182

C.2.2 Consistances d'ordre supérieur 182

C.3 Stratégies d'exploration de l'arbre de recherche 183

Annexe D Preuves **185**

Références bibliographiques **205**

Liste des figures

0.1 Architecture du système et plan du mémoire 19

1.1 Un graphe d'états 24
1.2 Un espace de plans partiels 25
1.3 Validité d'un fait dans un plan partiel 26
1.4 Trois approches pour s'assurer de la validité d'un fait 27
1.5 Exemple d'axiome dans TIMELOGIC 30
1.6 Un problème d'allocation de ressources 31
1.7 Une typologie des problèmes d'ordonnancement pur 33
1.8 L'approche classique pour la planification 38
1.9 Bilan de quelques techniques et travaux en planification 41

2.1 Les relations de base de l'algèbre d'intervalles d'Allen 44
2.2 Un réseau de contraintes sur l'algèbre d'intervalles 45
2.3 Quelques algèbres d'intervalles et restrictions associées 46
2.4 Graphe potentiel-tâches 47
2.5 Propagation des contraintes sur un STP 48
2.6 Interactions entre le réseau symbolique et le réseau numérique 50
2.7 Plan simplifié du laboratoire 51
2.8 Insertion et propagation d'une contrainte d'égalité 54
2.9 Exemple de représentation d'un état du monde 56
2.10 Une évolution des instances de l'attribut "SUR" 58
2.11 Décomposition en types de ressources 59
2.12 Deux attributs de ressources 60
2.13 Représentation des ressources 61
2.14 Un exemple de tâche 63
2.15 Deux opérateurs STRIPS 66
2.16 Un opérateur STRIPS, deux tâches IxTeT 67

2.17 Un opérateur STRIPS, une tâche IxTeT 68

3.1 Instance incohérente et instance cohérente 72
3.2 Une base nécessairement cohérente au sens large 72
3.3 La transformation Θ 76
3.4 Un ensemble critique minimal 77
3.5 Architecture globale du système de planification 87
3.6 La stratégie de moindre engagement 89
3.7 Engagement lié à une contrainte de précédence 90
3.8 Engagement lié à l'insertion d'un lien causal 93
3.9 Opportunité d'un défaut 94
3.10 Graphe ET/OU pour l'analyse d'une proposition non-expliquée 95
3.11 Les différents types de menaces 96
3.12 Conflit faisant intervenir une assertion non-expliquée 97

4.1 Le module d'analyse des ressources 104
4.2 Le sous-ensemble Δ_1 105
4.3 Un exemple de GIP 107
4.4 Quelques notations sur un arbre de recherche 110
4.5 Principes de la recherche des ECM 110
4.6 Propriétés de l'arbre de recherche des ECM 112
4.7 Arbre de recherche des ECM : Algorithme glouton 113
4.8 Arbre de recherche des ECM : Algorithme DECLIC 114
4.9 Deux types de croissance d'un plan partiel 119
4.10 Gestion de ressources de même type à capacités différentes 124
4.11 Un plan partiel surcontraint suite à une production de ressources 127
4.12 Architecture du module d'analyse des ressources et paragraphes concernés . 128
4.13 Performances de l'analyse des ressources 129

5.1 Graphe $\mathcal{G}$ pour le domaine COLUMBUS 138
5.2 Deux états d'abstraction consécutifs 140
5.3 Une hiérarchie dépendant du scénario initial 143
5.4 Ordre $\ll'$ et classification des objets 145
5.5 Architecture du système hiérarchisé 149

6.1 Le domaine COLUMBUS 152
6.2 Graphe $\mathcal{G}$ pour le domaine COLUMBUS 156

6.3 Plan solution dans le domaine COLUMBUS 156
6.4 Effets du plan et cumul des ressources utilisées dans COLUMBUS 157
6.5 Evolution et répartition du temps passé dans les modules d'analyse 159
6.6 Le domaine ARIANE 160
6.7 Facteur de branchement et heuristique le long de l'arbre de recherche 161
6.8 Le domaine FINITION PIECE 161
6.9 Heuristique avec hiérarchie d'abstraction 162
6.10 Exemple de plan solution 163
6.11 Le domaine INTREPIDE 164
6.12 Transition de phase sur le domaine INTREPIDE 164
6.13 Gain apporté par la stratégie de moindre engagement 166
6.14 Gain apporté par une prise en compte hâtive des ressources 166
6.15 Gain apporté par la hiérarchisation de la recherche 167
6.16 Acquis et perspectives 170

B.1 Quelques problèmes sur un Graphe Faiblement Triangulé 175
B.2 Recherche des cliques maximales sur un graphe triangulé 177
B.3 Un graphe qui n'est pas de comparabilité 177
B.4 Un ensemble d'intervalles et le graphe associé 178
B.5 Bilan 179

C.1 Une formulation CSP du problème des 4 reines 181
C.2 Une solution au problème des 4 reines 181
C.3 Le CSP des 4 reines filtré par consistance de chemin 183

D.1 Graphe des contraintes temporelles symboliques entre instants 192
D.2 Systématicité de la recherche des ECM 195
D.3 Etats d'abstraction consécutifs sur un arbre de recherche 201

Introduction générale

Une définition du terme *planification* est : *décider dans le détail quoi faire pour atteindre un but et comment le faire*[1].

Décider *quoi faire*, c'est décider la *nature* des actions que l'on va devoir entreprendre.

Par exemple, si je planifie un déménagement, cela consistera à déterminer les différents aller-retours que je devrais effectuer et ce que je déménagerais à chaque étape.

Déterminer *comment le faire*, c'est déterminer les *moyens* que l'on devra mettre en œuvre pour exécuter ces actions et les organiser dans le temps en conséquence.

Dans l'exemple du déménagement, les moyens seront la taille du ou des véhicules que je vais utiliser ou le nombre de mes amis qui pourront venir me donner un coup de main en fonction de leur disponibilité.

Parce que certains problèmes se posent en priorité l'une ou l'autre de ces questions, elles ont et sont toujours étudiées en profondeur et de manière indépendante. La question du *"Quoi faire ?"* est traitée par la problématique de la *synthèse de plans*, la question du *"Comment le faire ?"* par celle de l'*allocation de ressources* et l'*ordonnancement*.

Assez récemment toutefois, des efforts de recherche ont été entrepris pour faire converger ces deux problématiques. Nous les rappelons dans le premier chapitre qui a pour objet de donner un aperçu de l'état de l'art en planification au sens large. Les motivations de ces efforts sont claires : les questions du *"Quoi faire ?"* et du *"Comment le faire ?"* ne sont absolument pas indépendantes.

Les approches classiques pour lesquelles *synthèse de plans* et *ordonnancement* constituent deux processus successifs de résolution du problème global, souffrent en effet d'une trop grande rigidité et d'un manque de coopération entre ces deux composantes. Cela peut avoir pour conséquence une dégradation des performances du système global suite à de nombreux retours nécessaires sur la synthèse du plan ou la production d'ordonnancements largement sous-optimaux.

Si nous reprenons l'exemple du déménagement, il est clair qu'il me sera très difficile de planifier des aller-retours si je ne sais pas de combien de véhicules je disposerais. Si je veux

[1] Cette définition est en fait une concaténation de celles du "Le Micro Robert, 1986" (entrée *planification*) et du "Collins Cobuild, 1987" (entrée *planning*).

ignorer les moyens dont je dispose, je peux faire soit des hypothèses optimistes (une infinité de véhicules) soit des hypothèses pessimistes (un seul véhicule, voire même aucun) mais dans ce cas, je risque de me retrouver avec un plan soit irréalisable, soit largement sous-optimal.

Dans ce mémoire nous défendons l'idée que ces deux questions essentielles doivent, dans la mesure du possible, être abordées de front plutôt que séparées en deux corpus de connaissances et de techniques distincts.

Notre approche se situe au carrefour de la planification pratique (*"practical planning"*) au sens de systèmes comme SIPE-2 ou O-Plan2 et d'études plus théoriques effectuées dans le cadre d'algorithmes comme ceux de TWEAK ou SNLP, de la théorie des graphes ou de celle des problèmes de satisfaction de contraintes afin de prouver des propriétés essentielles de notre algorithme comme sa *correction* et sa *complétude*.

Nous montrons qu'il est possible de développer un système capable de résoudre des problèmes réalistes sans que les résultats et les preuves formelles soient délaissées au profit de l'efficacité globale de l'algorithme. Notre approche a été implémentée en C++ sur le système de planification IxTeT .

L'organisation du mémoire suit l'architecture globale du système qui est schématisée sur la figure 0.1.

Nous traitons dans le second chapitre de la représentation utilisée pour formaliser le problème de planification. La représentation du monde s'effectue grâce à des attributs valués que l'on peut considérer comme des variables d'état décrivant l'environnement à un moment donné. La modélisation du changement s'effectue dans un langage très riche permettant d'exprimer des effets situés en dedans ou au-delà de l'intervalle de réalisation des actions (appelées *tâches* dans IxTeT). Notre principale contribution dans ce secteur est la représentation d'une large gamme de ressources et de leur utilisation par les tâches. Nous proposons aussi d'enrichir le formalisme avec un prédicat particulier permettant d'exprimer certains effets dépendant du contexte.

IxTeT est un planificateur explorant un espace de plans partiels. Le processus de planification consiste à détecter et à sélectionner certains *défauts* du plan partiel afin de les résoudre par l'insertion de *résolvantes*. Ces défauts et résolvantes sont détectés et générés dans une phase d'*analyse* du plan partiel courant. A chaque nœud de l'arbre de recherche, un défaut sélectionné est résolu par l'insertion dans le plan partiel d'une de ses résolvantes. Ces choix non-déterministes, qui sont effectués par le *module de contrôle de la recherche*, sont conduits par une *stratégie opportuniste* de résolution des défauts et selon une approche de *moindre engagement*. L'algorithme de base est décrit dans le chapitre 3. Nous montrons formellement, après avoir défini un critère de plan solution, la *correction* et la *complétude* de cet algorithme. Les heuristiques permettant de guider les choix non-déterministes sont justifiées.

Le chapitre 4 se focalise sur l'analyse des conflits de ressources potentiels (ou *ensemble critiques minimaux*) sur le plan partiel. Pour cela, nous proposons un algorithme basé sur

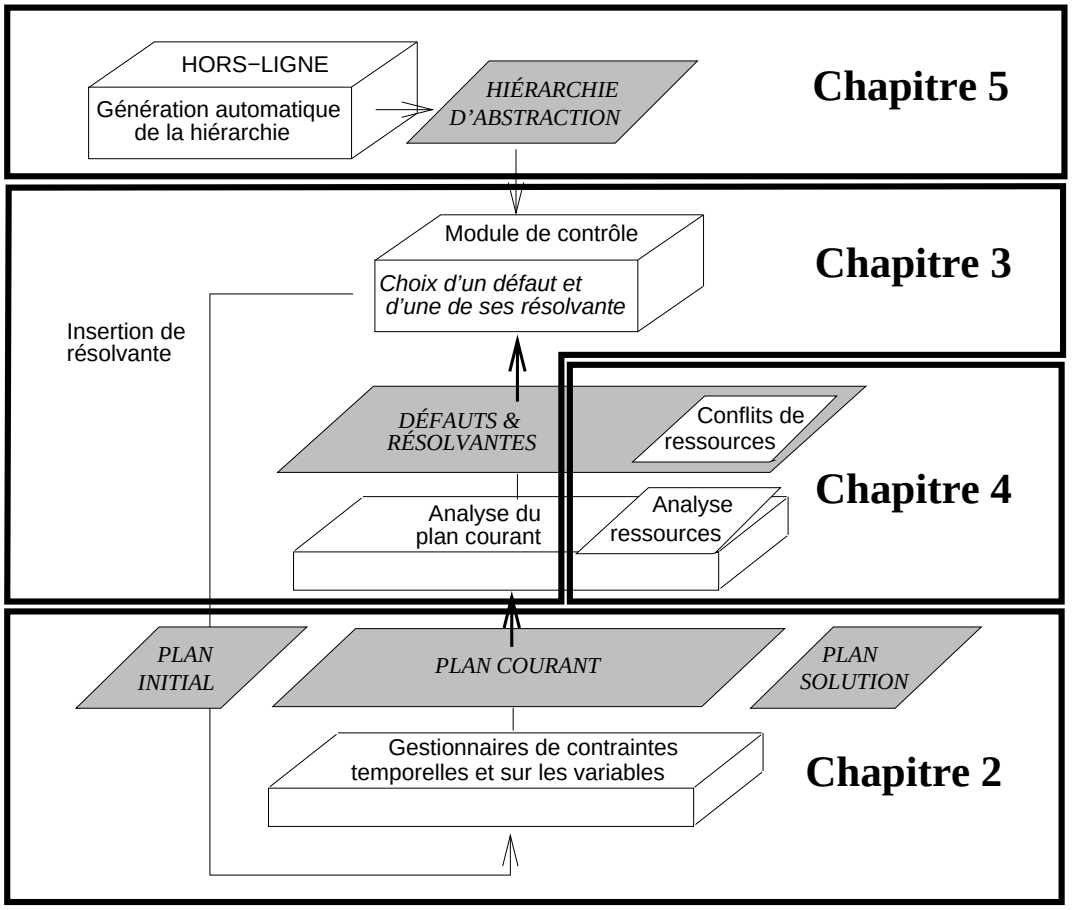


Figure 0.1: Architecture du système et plan du mémoire

la recherche de cliques dans une famille de graphes possédant des propriétés relativement sympathiques : les *graphes faiblement triangulés*. Nous montrons la *correction*, la *complétude* et la *systématicité* de cet algorithme de détection des conflits de ressources potentiels. Afin d'éliminer la redondance dans l'arbre de recherche d'un plan solution, nous définissons une notion de *minimalité de l'ensemble des résolvantes* associées à un conflit de ressource donné et décrivons l'algorithme de minimisation associé.

Etant donné nos objectifs d'efficacité de la recherche et pour faire face à la complexité du problème global de planification nous proposons, dans le chapitre 5, de hiérarchiser la recherche. L'approche que nous décrivons étend au formalisme IxTeT et à la gestion de ressources certains travaux désormais classiques en synthèse de plans permettant la génération automatique de hiérarchies d'abstraction. Au-delà de cette extension, nous introduisons la notion de *hiérarchie dynamique* qui permet de laisser au module de contrôle de la recherche l'opportunité de dérouler la hiérarchie comme il l'entend dans la mesure où il respecte certaines contraintes minimales.

Le dernier chapitre illustre, sur quelques exemples, la variété des problèmes pouvant être représentés et résolus par notre approche. On y trouvera une analyse des performances du système et une justification des stratégies utilisées.

Chapitre 1

Quelques travaux en planification

1.1 Introduction

La **planification** au sens large, telle que nous l'entendons dans ce chapitre, consiste, étant donné un objectif à établir ou à maintenir, à répondre aux trois questions suivantes:

1. *Que fait-t-on ?*, c'est-à-dire quelle est la nature des tâches que l'on va devoir entreprendre pour satisfaire l'objectif. Ce type de problème est celui typiquement étudié en **synthèse de plans** (ou planification au sens strict);
2. *Comment le fait-t-on ?*, c'est-à-dire quels moyens, quelles ressources affecte-t-on aux tâches pour assurer leur exécution. Il s'agit d'un problème d'**affectation** de ressources ; et
3. *Quand le fait-t-on ?*, c'est-à-dire comment organise-t-on les tâches dans le temps en fonction des différentes contraintes (ressources, préférences,...). Il s'agit alors d'un problème d'**ordonnancement**.

Ces trois questions ne sont bien entendu pas indépendantes, comme nous l'avons vu dans l'introduction générale. Malgré cela, certains problèmes bien étudiés peuvent se formuler dans le cadre d'une seule des trois questions ci-dessus. Dans ce chapitre qui n'a pas la prétention de couvrir tout l'état de l'art, nous présentons quelques travaux relatifs à ces trois problématiques. La conclusion principale est que ces trois questions ont été et sont toujours étudiées très en profondeur mais que, jusqu'à présent, assez peu de travaux et de résultats ont été présentés relatifs à une résolution globale de ces trois problématiques pourtant indissociables dans les problèmes complexes.

1.2 Synthèse de plans

Formellement, le problème de synthèse de plans exploite trois types de données [Weld 95]:

1. une *description du monde* dans un langage formel quelconque donné;
2. une *description des actions* (ou tâches) pouvant être exécutées sur le monde afin d'atteindre les objectifs fixés (ici encore, cette description s'effectue dans un langage formel donné) ; et
3. une description des *objectifs* de l'agent, c'est-à-dire du comportement désiré.

Le résultat de l'algorithme de synthèse de plans est un *plan* c'est-à-dire un ensemble d'actions totalement ou partiellement organisées dans le temps qui, si elles sont exécutées, permettront d'assurer les objectifs fixés.

Le type et l'expressivité des langages utilisés pour représenter le monde (point 1) et l'action et le changement sur ce monde (point 2) peuvent être très variables d'une approche à l'autre. Deux types d'approches se sont très tôt distinguées :

- les *approches logiques* où les actions et les états du monde sont des termes d'une même théorie logique. La recherche d'un plan se ramène alors à la démonstration d'un théorème [Green 69] ; et
- les *approches par opérateurs de changement d'état* où chaque état est représenté par un modèle d'une théorie logique. Chaque opérateur décrit alors le changement de formules d'une théorie logique à l'autre [Fikes 71].

Les approches logiques ont été très étudiées d'un point de vue théorique mais les aspects d'efficacité de ces approches sur des problèmes réels ont été presque entièrement ignorés. Nous n'examinerons pas ici les approches logiques. Quelques références sur celles-ci sont donnés dans [Alami 92].

Les approches par opérateurs, de leur côté, furent au départ des méthodes essentiellement informelles, dédiées à être implémentées afin de résoudre efficacement certains problèmes. Peu d'efforts furent fournis, à l'époque, pour montrer la correction de ces approches. Ce n'est qu'à partir du planificateur TWEAK [Chapman 87] que des résultats théoriques concernant la correction, la complétude et la complexité de la recherche ont été montrés. Depuis, ces approches sont l'objet de moultes recherches et enrichissements.

Beaucoup de travaux sont basés sur le formalisme introduit dans STRIPS [Fikes 71] pour décrire le monde et le changement.

Dans STRIPS, un état du monde est décrit par un ensemble de faits $S = \{p_{S1}, ..., p_{Sn}\}$ vrais dans cet état-là. Une action A est constituée de trois ensembles de faits:

- ses préconditions, $Precond(A)$, qui spécifient les faits qui doivent être vérifiés par l'état courant du monde pour pouvoir y exécuter A;
- l'ensemble $Delete(A)$ des faits qui seront supprimés suite à l'exécution de A ; et

- l'ensemble $Add(A)$ des nouveaux faits qui seront apportés par l'exécution de A.

Une action A est donc applicable dans un état S du monde si et seulement si $Precond(A) \subset S$. Dans ce cas, l'exécution de cette action fera passer le monde de l'états S à l'état S' défini par $S' = (S \backslash Delete(A)) \cup Add(A))$. Le *graphe d'état* est alors défini comme le graphe orienté dont les nœuds sont l'ensemble de tous les états possibles du monde et les arcs, les actions permettant de passer d'un état à un autre.

Un problème de planification consiste en une description de l'état initial du monde, S_0, et une caractérisation partielle, sous la forme d'un sous-ensemble de faits appelés *buts*, de l'ensemble des états finaux possibles.

La planification dans le cadre du formalisme STRIPS est un problème PSPACE-complet [Bylander 92]. Certaines restrictions d'ordre purement syntaxiques (c'est-à-dire imposant par exemple des contraintes sur le nombre de faits modifiés par un opérateur) ainsi que des restrictions d'ordre structurelles (contraintes sur la structure du graphe de transitions d'état induit par les opérateurs) permettent toutefois de caractériser quelques sous-classes de problèmes qui sont de complexité polynômiale [Bäckström 95a]. C'est par exemple le cas de l'usine de fabrication de voitures LEGO décrit dans [Klein 95].

Pour ce qui est de la recherche d'une solution dans le cas général, deux approches ont été développées selon que l'on explore un *espace d'états* ou un *espace des plans partiels*.

1.2.1 Recherche dans l'espace d'états

Chercher un plan dans l'espace d'états revient à chercher un chemin sur le graphe d'états conduisant de l'état initial à un état quelconque contenant les faits spécifiés dans les buts (cf. figure 1.1).

Deux types d'algorithmes sont distingués selon que l'on avance de l'état initial vers les buts (*progression*) ou des buts vers l'état initial (*régression*).

L'algorithme de progression consiste "simplement" à choisir une action parmi celles qui sont applicables dans l'état courant et à progresser dans le graphe d'état en appliquant cette action. Les heuristiques guidant ce choix sont en général des mesures de distance entre l'état courant et l'ensemble des états buts. Dans cette approche, tout repose sur l'efficacité de l'heuristique et dans le cas général, celle-ci est très difficile à trouver. L'intérêt de ce type d'approche est essentiellement de construire le plan de manière chronologique ce qui permet, si l'on s'interdit par exemple de revenir trop en arrière sur les choix effectués, de commencer l'exécution du plan avant qu'il ne soit complètement généré (voir la notion de tampon de retours-arrière dans [Bastié 96]).

L'algorithme de régression [Nilsson 80] présente l'avantage d'avoir un facteur de branchement en général plus faible dans la mesure où seules les actions qui sont relevantes pour la résolution des buts courants sont envisagées.

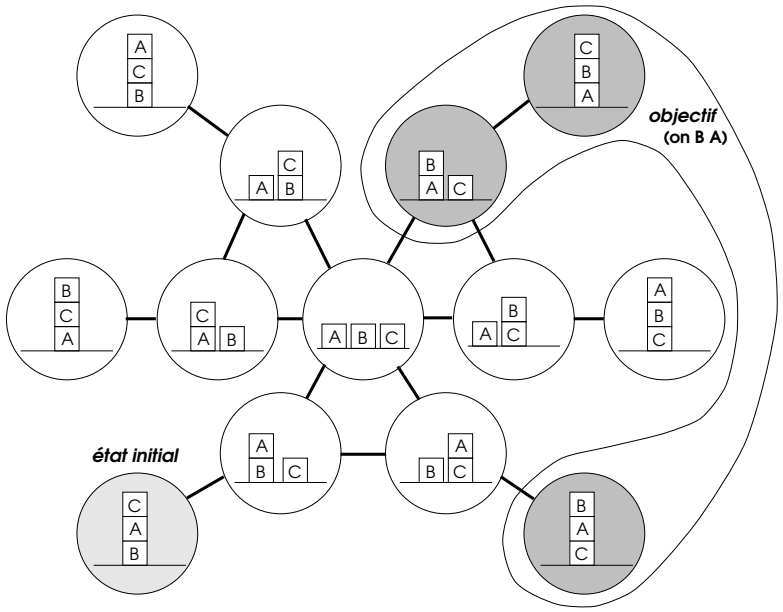


Figure 1.1: Un graphe d'états

Il est aussi possible d'entrelacer les mécanismes de progression et de régression sous la forme d'une recherche *bidirectionnelle*. C'est ce qui est fait dans le planificateur PRODIGY [Fink 95a] dans lequel le plan partiel est représenté sous la forme d'un couple $< Tête, Queue >$ où la tête est un plan linéaire généré de manière progressive et la queue un plan partiellement ordonné généré de manière régressive. A chaque étape de la planification, le contrôle de la recherche peut choisir de rajouter une opération en fin de tête ou dans la queue. On notera toutefois que dans PRODIGY, si la tête du plan est considérée comme une séquence d'actions et la progression comme une recherche dans un espace d'états, la queue est un plan non-linéaire et la régression s'effectue dans un espace de plans partiels selon des techniques que nous allons voir dans le paragraphe qui suit. Citons aussi l'approche utilisée dans UCP ([Kambhampati 95]) où progression et regression sont considérés comme différentes stratégies de raffinement du plan courant dans un cadre qui intègre planification dans l'espace d'états et planification dans l'espace des plans partiels.

1.2.2 Recherche dans l'espace des plans partiels

Le problème principal lié à la planification dans l'espace d'états est lié au fait que le plan est généré de manière linéaire et donc que le placement des actions dans le plan est imposé et sur-contraignant.

Dans le planificateur NOAH [Sacerdoti 75], la recherche s'effectue dans un graphe où les nœuds sont des plans partiels (ensemble d'actions partiellement ordonnées) et les arcs des transformations de ces plans partiels par des insertions de nouvelles actions ou de contraintes de précédence entre actions. Le plan partiel de départ ne contient que deux actions fictives:

- une action initiale I sans précondition ni liste de retrait et dont les ajouts (Add)

décrivent l'état initial ; et

- une action finale G sans liste d'ajout ou de retrait, mais dont les préconditions ($Precond$) spécifient les buts à atteindre.

Voir la figure 1.2 qui décrit un morceau de l'espace des plans partiels lié à la résolution du même problème que celui de la figure 1.1 dans le monde des cubes.

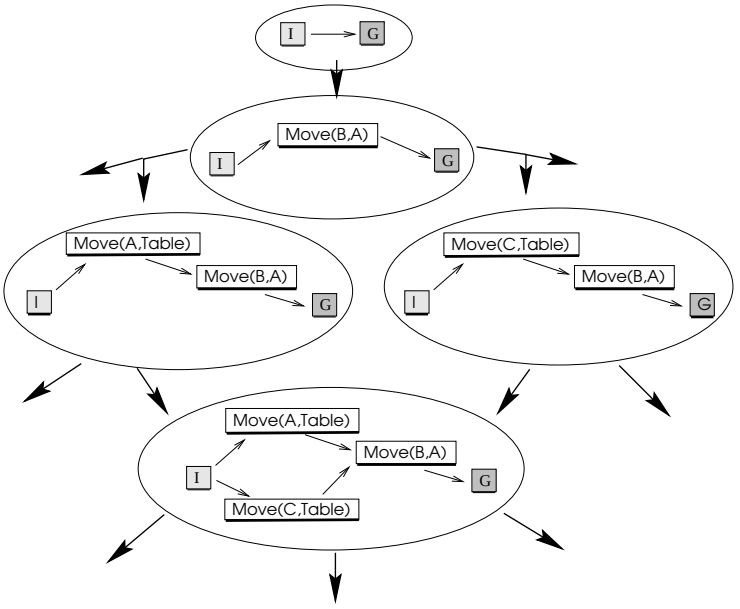


Figure 1.2: Un espace de plans partiels

Un certain nombre de propriétés et de résultats de complexité liés à la recherche dans l'espace des plans partiels ont été montrés dans le cadre des planificateurs TWEAK [Chapman 87] et SNLP [McAllester 91].

Un des problèmes essentiel posé par la planification dans l'espace des plans partiels est de s'assurer de la nécessaire validité d'un fait (c'est-à-dire, en pratique, de la précondition d'une action) dans une situation donnée. Ceci provient du fait que l'ordre des actions n'étant que partiel, on ne dispose plus de la notion d'état de laquelle l'on puisse directement extraire la validité de ce fait. Ainsi sur la figure 1.3, la précondition $clear(B)$ de l'action $Move(B, A)$ est elle nécessairement vraie quelle que soit la linéarisation possible du plan partiel ? Dans quelle mesure restera-t'elle vraie si en cours du processus de planification on rajoute de nouvelles actions au plan ?

TWEAK et SNLP apportent deux réponses différentes à ce type de questions.

Dans TWEAK un critère modal de vérité (*modal truth criterion*) est évalué chaque fois que l'on a besoin de déterminer la validité d'un fait. De complexité quadratique, ce critère détermine si, oui ou non, une précondition d'une action est satisfaite quelque soit l'ordre total des actions compatible avec l'ordre partiel du plan courant. Ainsi, sur l'exemple, il est clair que l'action $Move(C, B)$ peut venir détruire la précondition $Clear(B)$ de $Move(B, A)$

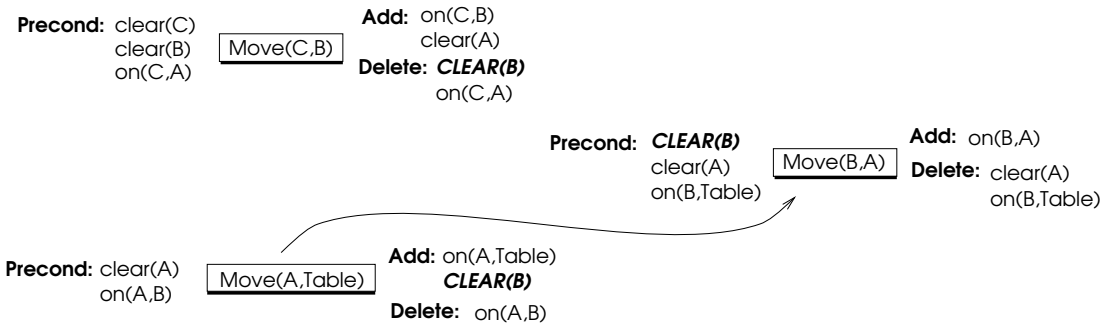


Figure 1.3: Validité d'un fait dans un plan partiel

qui pouvait être établie par les effets de l'action $Move(A, Table)$; le critère n'est donc pas vérifié. Dans le cas où le critère n'est pas vérifié, on impose la validité de la précondition par ajout de contraintes de précédence ou de nouvelles actions. Le critère de vérité doit bien entendu être reconsidéré dès qu'une nouvelle action est insérée au plan. Le plan partiel est un plan solution lorsque le critère de vérité valide toutes les préconditions de toutes les actions. Dans son papier, Chapman montre la correction et la complétude de l'algorithme.

Dans SNLP, à la différence de TWEAK, une action appelée *établisseur* est choisie une fois pour toute pour établir une précondition donnée. Une *protection* du fait est posée entre l'*établisseur* et l'action dont on veut expliquer la précondition (notion de *lien causal*). L'ensemble des protections est mémorisé. Par la suite, toute action risquant d'entrer en conflit avec une protection sera considérée comme une *menace* et ordonnancée avant ou après la protection.

L'algorithme est résumé ci-dessous de manière informelle:

Algorithme $SNLP$(P:Plan partiel, SB:Liste des sous-buts)

1. **Terminaison:** si SB est vide retourner P
2. **Sélection d'un sous-but** b **dans** SB. b est précondition d'une action A
3. **Sélection d'un établisseur** $A_{établisseur}$ **de** b**:**
   - $A_{établisseur}$ est soit une action déjà insérée dans le plan,
     soit une nouvelle action que l'on insère avant A.
   - mémorisation du lien causal $\{A_{établisseur} \xrightarrow{b} A\}$
4. **Mise à jour des sous-buts:**
   - retirer b de SB
   - si $A_{établisseur}$ est une nouvelle action, rajouter à SB les nouveaux sous-buts
     introduits par $A_{établisseur}$
5. **Protection des liens causaux:**
   - pour chaque action A_m pouvant menacer un lien causal $\{A_{établisseur} \xrightarrow{b} A\}$,
     ajouter $A_m < A_{établisseur}$ (demotion) ou bien $A < A_m$ (promotion)
   - si aucune possibilité n'est consistante : retourner ECHEC
6. **Appel récursif** de $SNLP$(P,SB)

L'algorithme est *systématique* c'est-à-dire qu'étant donné deux plans partiels qui ne sont pas sur un même chemin de l'arbre de recherche, ces deux plans ne possèdent aucune linéa-

risation possible en commun. La propriété de systématicité permet d'éliminer la redondance dans l'arbre de recherche. Outre la systématicité de la recherche, un des avantages de cette approche est d'éviter l'évaluation du critère de vérité après chaque modification du plan partiel courant. Par contre, les plans partiels examinés par un algorithme de type SNLP sont plus contraints que ceux examinés par TWEAK dans la mesure où plus de choix ont été effectués. Une étude et une comparaison des algorithmes de TWEAK et SNLP est présentée dans [Knoblock 93] où les auteurs montrent dans quelles conditions l'un des algorithmes se comporte mieux que l'autre: SNLP est plus performant que TWEAK lorsque le nombre de menaces négatives dépasse le nombre de menaces positives[1].

En fait, les approches TWEAK et SNLP peuvent être vues comme des solutions extrêmes en ce qui concerne la redondance et la systématicité de la recherche. Un compromis a été proposé dans [Kambhampati 93] avec l'approche multi-contributeurs où contrairement à SNLP, ce n'est pas un seul mais un ensemble disjonctif d'établisseurs qui est mémorisé pour expliquer une précondition. La notion de lien causal entre deux actions est alors étendue à celle de lien causal entre un ensemble d'actions (les établisseurs possibles ou *contributeurs*) et l'action dont on explique une précondition. Les trois approches sont résumées sur la figure 1.4.

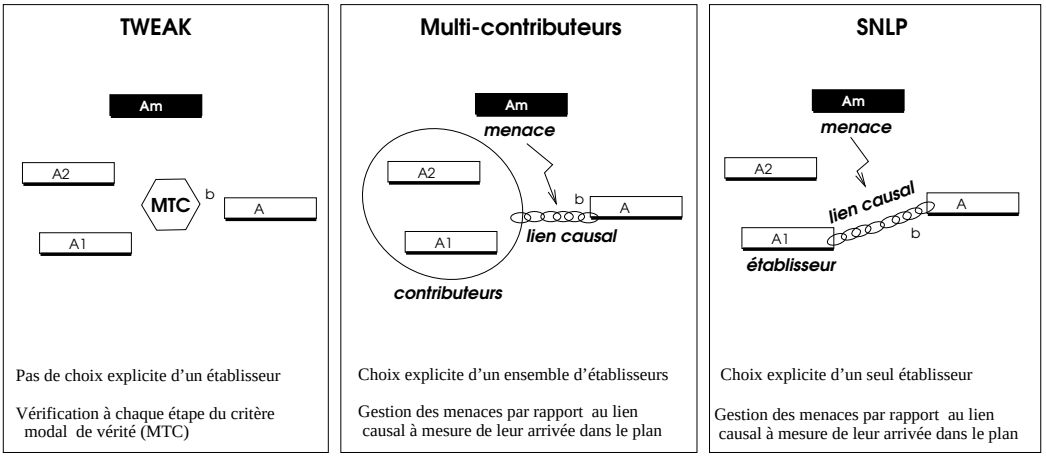


Figure 1.4: Trois approches pour s'assurer de la validité d'un fait

1.2.3 Vers un formalisme plus riche

1.2.3.1 Variables

L'idée d'utiliser des variables dans les opérateurs et dans le plan permet d'étendre la notion de moindre engagement qui avait déjà motivé le passage de la recherche dans un espace d'états à la recherche dans un espace de plans partiels. D'autre part, les variables permettent

[1] Une *menace négative* est une action risquant de détruire la protection; une *menace positive* est une action qui pourrait elle-même expliquer la précondition protégée.

d'économiser un travail fastidieux de description de l'ensemble des actions instanciées du domaine.

Un opérateur avec variables est plus souple qu'une action totalement instanciée. Il peut être inséré dans le plan sans que toutes ses variables soient encore instanciées. L'instanciation des variables peut ainsi être reportée à un point de l'arbre de recherche où cela semblera plus opportuniste.

Au cours du processus de synthèse, la vérification d'un critère de vérité (pour un système de type TWEAK) ou bien la gestion des menaces (pour un système de type SNLP) est enrichie par la possibilité de poser des contraintes sur les variables du plan (égalité, différence).

Généralement, l'ensemble des variables du plan et des contraintes qui pèsent sur ces variables est géré sous la forme d'un problème de satisfaction de contraintes (Constraint Satisfaction Problem ou CSP)[2]. Si, comme dans TWEAK, on se restreint à des variables à domaine infini, la propagation des contraintes ne pose pas de problème et peut être effectuée en temps polynômial [Chapman 87]. Si par contre on gère, comme cela est très utile pour des problèmes réels, des domaines discrets et finis, le seul problème de vérification de la consistance du réseau devient NP-difficile à cause des contraintes d'inégalités entre variables. Dans la mesure où toutes les variables intervenant dans les effets d'un opérateur sont aussi présentes dans au moins une de ses conditions, il est facile de voir que dans un plan solution et pour un algorithme de type SNLP, toutes les variables du plan sont instanciées. En conséquence, même si la propagation des variables d'inégalité est incomplète, cela ne remet pas en cause la correction de l'algorithme de planification à condition que le CSP permette de répondre à la consistance ou à la non-consistance d'une instanciation totale des variables. Ce qui, force est de reconnaître, est une des fonctionnalités minimales d'un CSP.

1.2.3.2 Planification temporelle

Même si le temps a toujours été, par essence, un élément essentiel de tout système de planification (les planificateurs dans l'espace d'états ont une notion séquentielle du temps, les approches de type TWEAK ou SNLP reposent fortement sur la notion de précédence entre opérateurs), le besoin s'est rapidement fait sentir d'enrichir la représentation temporelle pour, en particulier, prendre en compte des informations métriques sur l'écoulement du temps.

La première représentation explicite du temps a été introduite avec le planificateur DEVISER [Vere 83]. DEVISER étend le formalisme STRIPS en associant à chaque action (nommée *activité*) une durée. Les préconditions doivent être maintenues pendant toute la durée de l'activité. Les effets de l'activité ne deviennent effectifs qu'à la fin de celle-ci. Une autre contribution essentielle de DEVISER est la possibilité de représenter des *événements attendus* (par exemple le fait qu'un magasin ait un cycle connu de périodes d'ouverture et

[2]Quelques notions et techniques de base sur les CSP sont rappelées en annexe C.

de fermeture). Le plan produit doit intégrer ces événements. Evénements et activités sont décrits avec le même formalisme. DEVISER synthétise le plan selon une procédure classique en établissant les sous-buts par un effet d'une activité (déjà dans le plan ou devant y être insérée) ou bien, ce qui est nouveau, par un événement. Les conflits sont gérés comme dans SNLP dès qu'ils apparaissent suite à l'insertion d'une nouvelle activité. La résolution s'effectue par l'ajout de contraintes de précédence entre activités parallèles. Le réseau de contraintes de précédence entre activités et les durées de celles-ci sont représentés sous la forme d'un réseau de type PERT qui permet de propager à chaque modification du plan les fenêtres temporelles [date au plus tôt,date au plus tard] de chaque activité. Une des limites de DEVISER est la rigidité de la représentation qui ne permet pas de représenter des changements ayant lieu en cours d'activité. D'autre part, la représentation qui impose que les préconditions doivent rester vraies durant toute l'activité est quelquefois trop contraignante. Dans le planificateur TRIPTIC [Hertzberg 93], un certain nombre de ces limites sont repoussées (les préconditions doivent être vraies au début de l'activité, les effets seront effectifs à la fin) bien que toute la richesse potentielle d'une représentation explicite du temps ne soit pas exploitée ; par exemple, il n'est pas possible d'y représenter une action susceptible d'établir et de maintenir un fait uniquement pendant son intervalle de réalisation.

Ce type d'action est traitée dans TIMELOGIC [Allen 83b, Allen 91]. TIMELOGIC repose sur l'algèbre d'intervalles d'Allen [Allen 84]. Dans ce système les actions, comme les faits permettant de décrire l'état du monde, sont des prédicats du langage. Ainsi l'action de déplacer un bloc A sur un bloc B sera décrite par le prédicat $Deplace(A, B, e, i)$ où e est l'événement lié à l'action[3] et i l'intervalle temporel de l'action. Les conditions et les effets d'une action sont alors décrits par un axiome. Par exemple:

$$\forall i, a, b, c, t, e\ Deplace(A, B, e, i) \wedge On(A, C, t) \wedge Overlaps(t, i)$$
$$\rightarrow Clear(C, eff3(e)) \wedge Meets(t, eff3(e)) \wedge Meets(t, con1(e))$$

Cet axiome s'applique dans une situation où A est sur C à l'instant où l'on commence l'action de déplacement. Il décrit alors le fait que C sera libre dès que A ne sera plus sur C ce qui arrivera pendant l'intervalle de l'action (cf. figure 1.5).

Etant donné une situation initiale W_{init} et un ensemble de buts B, le processus de synthèse de plans consiste à rechercher une suite d'hypothèses d'actions $A_1, ..., A_n$ telles que, à partir de W_{init}, il est démontrable que $A_1 \wedge ... \wedge A_n \rightarrow B$.

Ce processus s'effectue en deux étapes:

- une étape de prédiction maintenant à jour l'état du monde à partir des hypothèses courantes ; et
- une étape de planification proprement dite pour générer les hypothèses d'actions.

[3]Cette notion d'événement utilisée par Allen est très différente de la notion d'événement instantané que nous utilisons dans IXTET . Ici, un événement est associé à un intervalle temporel.

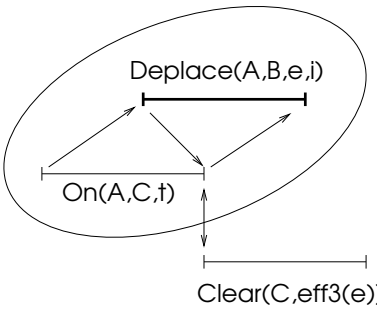


Figure 1.5: Exemple d'axiome dans TIMELOGIC

Dans TIMELOGIC, le formalisme de représentation de l'action est très souple et très riche. En particulier, la logique permet de représenter énormément d'axiomes du domaine et de traiter des effets dépendant du contexte. Une des limites principale de l'approche est la complexité qui se cache derrière le processus de prédiction.

1.2.3.3 Autres extensions

L'extension du formalisme et des algorithmes de base s'effectue aussi dans d'autres directions très intéressantes.

Le planificateur UCPOP [Penberthy 92] est un planificateur de type SNLP pouvant gérer des actions à *effets conditionnels* ainsi que des *quantificateurs universels* (par exemple le fait que quelque soit l'objet contenu dans une valise, celui-ci se déplacera suite à l'exécution de l'action de déplacement de la valise). Cette représentation, plus riche que celle de STRIPS, repose sur le langage ADL [Pednault 89].

Citons aussi les travaux en planification sous incertitudes où l'hypothèse de déterminisme des actions est levée [Kushmerick 94, Thiébaux 95].

1.3 Affectation de ressources

Dans les problèmes d'affectation de ressources purs, la composante temporelle est inexistante ou est une donnée d'entrée du problème. Le problème peut alors s'exprimer de la manière suivante.

Etant donnés :

- un ensemble de ressources non-partageables $R = \{r_1, ..., r_m\}$,
- un ensemble de tâches $\{T_i\}$ exprimant chacune un besoin sur une ressource parmi un ensemble $R_i \subset R$, et
- un ensemble C de contraintes restreignant les combinaisons de ressources allouables aux tâches,

affecter une ressource r à chaque tâche T_i en respectant les contraintes de C.

Dans le cas où l'on possède un critère qualifiant une affectation donnée, on peut envisager la résolution du problème d'optimisation correspondant.

On voit qu'une telle définition d'un problème d'affectation est, en fait, très similaire à celle d'un problème de satisfaction de contraintes. Ceci explique pourquoi beaucoup de techniques d'affectation de ressources sont basées sur la technologie CSP.

Parmi les problèmes d'affectation, nous distinguerons en particulier l'affectation *sans réemploi* et l'affectation *avec réemploi* des ressources.

Dans les problèmes d'*affectation sans réemploi* l'aspect temporel est inexistant ou, tout au moins, il n'est absolument pas structurant. Il s'agit d'affecter au mieux des ressources aux tâches, sachant qu'une ressource ne peut être affectée qu'à une tâche au maximum. Dans [Roy 69], l'auteur cite l'exemple de n tâches (exécutées en parallèle) qui devront être confiées chacune à un ouvrier donné (sur une équipe de m ouvriers). La tâche exprime un besoin parmi un certain nombre d'ouvriers possédant les qualifications requises. Le problème est transformable en un problème d'optimisation si chaque ouvrier exprime une affinité particulière pour certaines tâches. Si ce problème est facilement soluble par un algorithme spécifique, il n'en est pas toujours de même pour le problème général d'affectation sans réemploi. On trouve dans [Keng 89] une formalisation très générale de ce type de problèmes dans le formalisme des CSP sous l'appellation de *problèmes de ressources contraints* (ou Constrained Resource Problems, CRP) ainsi que des heuristiques de choix permettant de guider la résolution.

Les problèmes d'*affectation avec réemploi* , souvent appelés problèmes d'*allocation* de ressources consistent à allouer des ressources à un ensemble de tâches déjà positionnées dans le temps (voir l'exemple sur la figure 1.6).

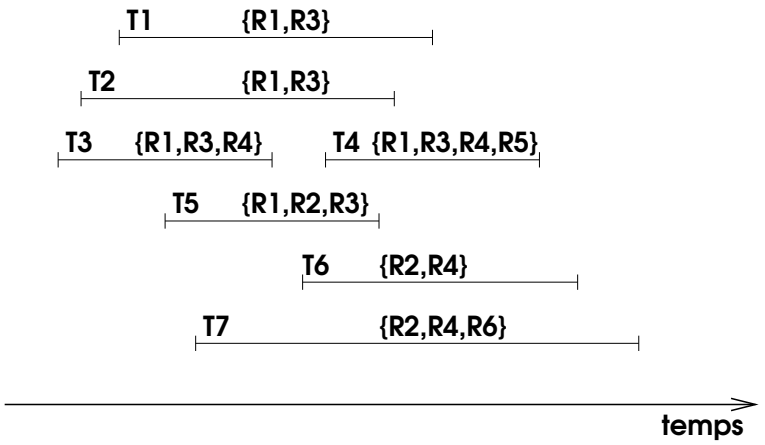


Figure 1.6: Un problème d'allocation de ressources

Deux types de techniques sont généralement adoptées pour résoudre cette famille de problèmes:

- celles issues de la théorie des graphes reposent sur une représentation du problème sous la forme d'un *graphe d'intervalles* dont les nœuds sont les tâches T_i et les arêtes, les paires $\{T_i, T_j\}$ telles que les intervalles temporels associés à T_i et T_j se chevauchent. Résoudre le problème d'allocation revient alors à résoudre le problème de coloration d'un graphe d'intervalles avec une liste de couleurs possibles spécifiques à chaque sommet. Ce problème, plus complexe que celui de la coloration minimale reste NP-complet sur les graphes d'intervalles[4];

- celles issues des CSP. Les tâches sont les variables du CSP et les domaines de valeurs qui leurs sont associées sont les ressources allouables. Une contrainte de différence sur les ressources allouées est posée entre toutes les paires de tâches s'intersectant dans le temps et ayant au moins une ressource en commun. Plusieurs stratégies sont alors applicables pour résoudre ce CSP spécifique. Dans [Chouery 95], une stratégie originale de report de l'allocation des ressources les plus demandées (VAD) permet de décomposer le problème original en sous-problèmes plus simples. Si une ressource r est très demandée par les tâches, on réagit dans un premier temps comme si celle-ci n'existait pas et donc, on alloue aux tâches des ressources moins demandées. L'ensemble des ressources dont l'allocation a ainsi été reportée est mémorisé et ce n'est que lorsqu'on ne peut plus allouer une ressource libre à une tâche que l'on va piocher dans l'ensemble des ressources reportées. Cette approche se montre relativement performante sur des problèmes de taille réaliste. Par contre, des propriétés comme la complétude de la recherche sont difficiles à montrer.

1.4 Ordonnancement

"Ordonnancer un ensemble de tâches, c'est programmer leur exécution en leur allouant les *ressources* requises et en fixant leurs dates de début" [GOThA 93]. Dans ce paragraphe, nous nous intéressons essentiellement aux problèmes où les inconnues sont uniquement les dates de début des tâches [5]. Nous supposons que les ressources utilisées par celles-ci sont des données du problème.

Nous résumons ici quelques problèmes classiques d'ordonnancement ainsi que certaines approches de résolution. Le lecteur plus curieux sur les problèmes et techniques d'ordonnancement pourra se référer à [Carlier 88, MacCarthy 93, Pinedo 95].

1.4.1 Les problèmes d'ordonnancement

La typologie des problèmes d'ordonnancement est très large. Si l'on se restreint aux problèmes dits d'*ordonnancement pur*, c'est-à-dire ceux pour lesquels la durée des tâches ainsi

[4] Quelques notions de théorie des graphes utilisées dans ce mémoire sont rappelées dans l'annexe B.

[5] C'est-à-dire les problèmes de type T selon la typologie définie dans [Roy 69], tome 2 ; par opposition aux problèmes d'affectation de ressources de type W où les dates sont des données et où les inconnues sont les ressources allouées aux tâches. Les problèmes de type W ont été évoqués dans le paragraphe 1.3 précédent.

que la nature et la quantité des ressources qu'elles utilisent sont des données du problème, on obtient une typologie classique du type de celle décrite sur la figure 1.7.

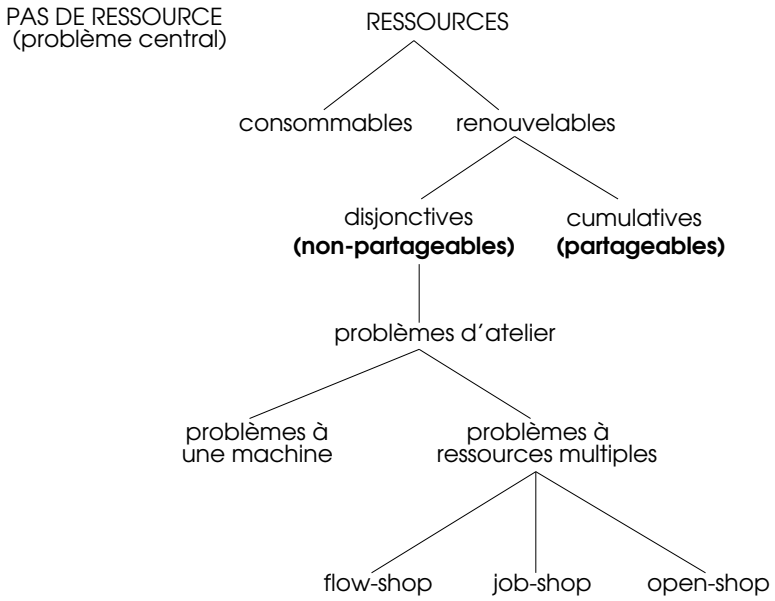


Figure 1.7: Une typologie des problèmes d'ordonnancement pur

Le problème uniquement temporel, où l'on suppose que les tâches n'utilisent pas de ressource, est appelé *problème central de l'ordonnancement* : il s'agit d'allouer des dates de début à un ensemble de tâches qui ne sont soumises qu'à des contraintes de succession ou de localisation temporelle (date au plus tôt, date au plus tard). On cherche alors, en général, à minimiser la durée totale de l'ordonnancement. Ceci s'effectue classiquement par des méthodes polynômiales de chemin critique sur un graphe potentiel-tâche tel que celui que nous donnons sur la figure 2.4. Cette même méthode peut être étendue, par des techniques de décalage, pour prendre en compte des *ressources consommables* par les tâches et produites de manière exogène au plan à certaines dates (par exemple, des dates de financement pour un projet) [Carlier 82].

Dans le cas général avec ressources on distingue classiquement :

- le cas de **ressources consommables** : l'exécution des tâches va entraîner une diminution de la quantité de ressource disponible (la tâche consistant à se rendre en voiture de A à B va consommer de la ressource carburant) ; et
- le cas de **ressources renouvelables** pour lesquelles la disponibilité n'est affectée que pendant la durée de la tâche (ainsi le fait que la ressource "voiture" devienne indisponible pour un autre conducteur pendant la durée du trajet de A à B). A noter que dans ce mémoire nous parlerons dans ce cas d'*emprunt* de la ressource.

Dans le cas où les ressources sont renouvelables, selon qu'une même ressource peut être ou non, à un instant donné, partagée entre plusieurs tâches, on distingue le cas des *ressources*

cumulatives et des *ressources disjonctives*. Dans ce mémoire, nous utiliserons des termes un peu différents, issus de la terminologie en synthèse de plans, pour désigner ces ressources : les ressources cumulatives seront appelées *ressources partageables* et les ressources disjonctives des *ressources non-partageables*.

Un cas très étudié en ordonnancement est celui des ressources renouvelables disjonctives, autrement nommés *problèmes d'atelier* dans la mesure où, bien souvent, il s'agit d'ordonnancer des travaux (*job*) sur différentes machines ; un travail se décomposant en un ensemble d'opérations et une machine ne pouvant effectuer qu'une seule opération à la fois. Il s'agit alors d'ordonnancer n travaux J_j, chacun étant constitué d'un certain nombre d'opérations o_{ij} et devant s'effectuer avant une date limite d'achèvement d_j. Chaque opération possède une durée p_{ij}, et exprime un certain nombre de besoins en ressources.

Dans ce cadre-là, on distingue trois types de problèmes:

- l'atelier à cheminement unique (plus connu sous le terme de *flow-shop*) où la séquence des machines sur lesquelles doit passer un produit est fixe quel que soit le produit ;
- l'atelier à cheminement multiple (ou *job-shop*) pour lequel chaque produit peut posséder une gamme spécifique;
- l'atelier à cheminement quelconque (ou *open-shop*) pour lequel l'ordre d'exécution des opérations d'un travail est libre.

En général, les problèmes d'ordonnancement sont des problèmes d'optimisation où l'on cherche à minimiser un critère donné sur l'ordonnancement solution. Citons, parmi les critères les plus utilisés:

- l'*étendue temporelle de l'ordonnancement (makespan)* : durée entre la date de début d'exécution de la première opération de l'ordonnancement et la date de fin d'exécution de la dernière opération.
- le *retard maximum* : c'est le retard du travail qui est le plus en retard vis-à-vis de sa date limite d'achèvement.
- la *somme pondérée des dates d'achèvement* : une pondération est affectée à chaque travail.
- la *somme pondérée des délais* : le délai associé à un travail est la différence entre sa date effective d'achèvement et sa date limite d'achèvement. Le délai est une quantité qui peut être négative si le travail est fini à temps (à la différence du retard).
- le *nombre pondéré de travaux en retard*.

Selon le problème d'ordonnancement à résoudre et les critères à optimiser, plusieurs approches peuvent être envisagées. Nous les résumons dans le paragraphe ci-dessous.

1.4.2 Quelques approches de résolution

Deux types d'approches sont généralement distinguées : d'une part les approches issues de la *recherche opérationnelle (RO)* et d'autre part celles liées à l'*intelligence artificielle (IA)*. Les approches purement issues de la recherche opérationnelle permettent de résoudre de manière très efficace des problèmes très particuliers, mais aussi de présenter une palette de techniques pour résoudre une gamme plus large de problèmes, pour peu que ceux-ci aient été bien formalisés. Une des limites de ces techniques est liée au fait qu'il n'y a pas de distinction stricte faite entre les connaissances liées au domaine et les connaissances et algorithmes liés au processus de résolution du problème [Grant 86].

L'introduction de techniques d'IA pour résoudre certains problèmes d'ordonnancement a donc principalement été motivée :

- par des raisons de souplesse de la représentation des connaissances (à la fois les connaissances sur l'environnement de l'atelier - machines, opérations - et les connaissances stratégiques liées au processus d'ordonnancement) ; et
- par la relative indépendance de ce type d'approches par rapport au problème considéré.

1.4.2.1 Approches issues de la RO

Une bibliographie très complète sur ces approches pourra être trouvée dans [GOThA 93].

Outre un certain nombre d'algorithmes très spécifiques trouvant des ordonnancements optimaux en temps polynômial pour des problèmes particuliers (par exemple les problèmes à une machine ou bien le job-shop à deux machines), on compte aussi:

- Les méthodes par *construction progressive* où, à chaque itération, on complète une solution partielle en y positionnant une opération supplémentaire. Ces méthodes peuvent ou non suivre le déroulement du temps. L'optimalité de certaines de ces méthodes a été montrée pour des problèmes particuliers mais elles conduisent, en général, à des solutions approchées.
- Les méthodes par *voisinage*, où l'on part d'une solution complète x_0 en essayant de la transformer localement (par exemple en inversant deux opérations ou en changeant de machine pour une opération); c'est-à-dire en choisissant une nouvelle solution parmi ses voisines $v(x_0)$. L'objectif est alors de minimiser le critère $f(x)$ associé aux solutions explorées. Le choix de la solution voisine peut s'effectuer selon des stratégies comme la plus forte pente, le recuit simulé ou des méthodes de type Tabou [Glover 93].
- Les méthodes de *relaxation* qui consistent à relâcher certaines contraintes du problème initial afin de simplifier la recherche et à utiliser ces solutions (qui ne sont pas toujours réalisables) pour guider le processus d'ordonnancement.

Parallèlement à ces approches, diverses *méthodes de décomposition* du problème global en sous-problèmes plus simples sont étudiées. Citons par exemple:

- la décomposition *hiérarchique* permettant de tout d'abord résoudre le problème à un niveau agrégé avant de résoudre les niveaux inférieurs plus détaillés;
- la décomposition *temporelle* qui permet de se focaliser d'abord sur les travaux dont l'échéance est la plus proche dans le temps. L'ordonnancement est alors construit portion par portion;
- la décomposition *spatiale* qui décompose l'atelier en sous-ateliers dans la mesure où ces sous-ateliers sont relativement indépendants les uns des autres.

1.4.2.2 Approches liées à l'IA

Une des premières "intrusion" de l'IA dans les problèmes d'ordonnancement s'est effectuée dans le cadre du système ISIS [Fox 84] au début des années 80. Pour la première fois, au lieu d'utiliser une modélisation très simplifiée de l'atelier de production, ISIS a tenté de gérer l'ensemble des contraintes et des objectifs pouvant être rencontrés dans le domaine de la gestion de production. Ce système qui présentait un certain nombre de faiblesses dues à la rigidité de la procédure de résolution utilisée a été amélioré dans le cadre du projet OPIS [Smith 86]. OPIS introduit la notion d'*opportunisme de la recherche*. Les auteurs partent de la constatation que les goulots d'étranglement de ressources, c'est à dire les points sur lesquels doit se focaliser le processus d'ordonnancement, ne sont pas donnés a priori mais évoluent dynamiquement au cours de la résolution du problème en fonction des choix effectués par l'algorithme. Ils argumentent alors la nécessité de réanalyser l'ordonnancement partiel courant dès que celui-ci a été modifié (par l'affectation d'une date ou d'une ressource à une opération) et ceci afin de déterminer les nouveaux points de focalisation de l'algorithme. Dans OPIS, les goulots d'étranglement de ressources sont en fait constitués d'une ressource particulière pour laquelle on s'attend à ce qu'elle puisse être sur-utilisée à un moment donné. La recherche opportuniste dans OPIS s'effectue donc en ordonnançant les opérations sur la ressource la plus conflictuelle, puis, une fois que celle-ci a été gérée, en analysant l'ordonnancement courant afin de déterminer, parmi les ressources non examinées celle qui est maintenant la plus conflictuelle et qui va devoir être gérée. Cette stratégie est dite *macro-opportuniste* dans la mesure où la granularité se situe au niveau des ressources. Des stratégies plus fines (dites *micro-opportunistes*) permettent de considérer des goulots d'étranglement beaucoup plus locaux qui sont certes liés à une ressource mais dont la portée est limitée dans le temps. Il devient alors possible de résoudre certains goulots d'étranglement sur une ressource donnée, puis de passer à la résolution d'un autre goulot sur une autre ressource (considéré comme plus opportun) avant même d'avoir résolu tous les goulots sur la première ressource. Une telle approche micro-opportuniste a été utilisée pour le système MICRO-BOSS [Sadeh 91].

Une manière de définir l'opportunité de la résolution d'un goulot d'étranglement ou d'un conflit de ressource potentiel peut être la facilité qu'il y aura à le résoudre. Un conflit sera d'autant plus facile à résoudre qu'il fera apparaître moins de choix non-déterministes lors de sa résolution. Cette stratégie classique consiste donc à résoudre en priorité les goulots d'étranglements très contraints. Il est, dans ce cadre précis, particulièrement intéressant de détecter tous les goulots d'étranglements pouvant être résolus de manière déterministe (c'est-à-dire ceux qui ne font apparaître aucune contrainte disjonctive).

Cette idée est étendue dans [Lopez 91, Erschler 90] en intégrant le cas plus général des ressources partageables. Une analyse très poussée des contraintes de l'ordonnancement courant est effectuée afin de déduire un maximum d'informations permettant de caractériser tous les ordonnancements solution atteignables à partir de celui-ci. L'approche repose sur des considérations énergétiques: une certaine quantité de ressource Q disponible sur un intervalle temporel de durée D correspond à une énergie $W = Q \times D$. Une analyse fine du bilan de l'offre et de la demande en énergie d'une ressource d'un type donné sur des intervalles temporels bien choisis permet alors de réactualiser les contraintes temporelles en rétrécissant les dates possibles pour les instants de début et de fin des tâches.

Une approche similaire d'analyse sous contraintes est utilisée dans le système OPAL pour gérer des problèmes d'atelier à ressources non-partageables [Bensana 88]. Un module spécifique d'OPAL (*constraint-based analysis module*) permet de détecter les conflits de ressources pour lesquels une seule possibilité d'ordonnancement est permise (notion de *conflit déterministe*). Un conflit de ressource est défini comme un couple d'intervalles utilisateurs d'une même ressource non-partageable et qui peuvent s'intersecter temporellement étant donné les contraintes de l'ordonnancement courant. Lorsque des choix non-déterministes sont nécessaires pour résoudre un conflit, une base de règles stratégiques permet de choisir en fonction du critère à optimiser, la meilleure décision à prendre. Certaines de ces règles pouvant donner des résultats contradictoires, la décision finale est le résultat d'une pondération du vote de chacune des règles. Ce vote est formalisé dans le cadre de la théorie des ensembles flous.

Citons enfin, pour conclure, d'autres approches liées à l'IA utilisant des techniques comme les algorithmes génétiques [Falkenauer 91] ou les réseaux neuronaux [Gallone 95].

1.5 Les travaux d'intégration

Après cette revue de quelques travaux relatifs aux trois questions que nous avons soulevées en introduction, nous voyons comment des éléments de réponses peuvent être apportés à chacune d'entre elles.

Il est clair toutefois que ces trois questions ne sont pas indépendantes. Ainsi en synthèse de plans, on ne se contente pas de sélectionner un ensemble de schémas d'opérateurs. Lorsque l'on résout une menace, des contraintes d'ordonnancement sont posées sur le plan. D'autre part, bien souvent, les variables utilisées dans la représentation ont une sémantique de

ressource (penser par exemple aux différents blocs) et dans ce cadre-là poser sur le plan partiel une contrainte entre deux variables revient à poser une contrainte sur l'allocation des ressources correspondantes.

La nature de certains problèmes d'ordonnancement, en particulier les problèmes de job-shop pour lesquels on a le choix entre plusieurs gammes possibles pour la fabrication d'un produit donné, nous amène aussi à nous poser la question du *"quoi faire ?"* en plus du *"comment et quand le faire ?"*.

Lorsque la nature du problème soulève à la fois les trois questions fondamentales de la planification, plusieurs approches sont possibles.

L'approche classique se décompose en deux étapes successives:

1. *synthèse du plan* sans prendre en compte toutes les contraintes de ressources ; et
2. *allocation des ressources* et *ordonnancement* du plan généré.

Les deux processus peuvent travailler sur des représentations très différentes du plan[6] ou bien sur des représentations identiques (voir la figure 1.8).

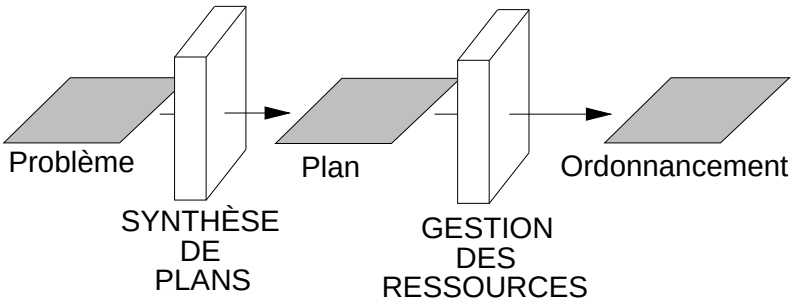


Figure 1.8: L'approche classique pour la planification

Une approche très structurée de ce type est définie dans le système DISA [Berry 94] où le problème global est décomposé en une hiérarchie à trois niveaux : la planification à long terme, l'ordonnancement des activités selon une ressource critique et l'allocation de ressources secondaires. Le système suit dans ce sens une approche très classique. Son originalité principale est due à une décomposition du temps sous la forme d'agents correspondant à une période donnée. A un niveau donné de la hiérarchie (par exemple l'ordonnancement selon une ressource critique), un agent donné (par exemple celui correspondant à la semaine prochaine) va se voir confier des activités à ordonnancer dans son domaine temporel. En cas d'impossibilité de tout intégrer dans son domaine, l'agent va pouvoir négocier avec ses voisins (dans le temps) pour leur demander de le décharger d'un certain nombre d'activités.

L'approche classique pose un problème dans la mesure où les interactions entre le processus de synthèse de plans et celui de gestion des ressources sont, soit quasiment inexistantes,

[6] Un cas extrême est celui où le plan est généré par un opérateur humain qui possède sa propre représentation mentale du plan et qui va la traduire dans le formalisme d'un système d'allocation et d'ordonnancement afin de gérer les questions de ressources.

soit très complexes et très difficiles à formaliser en dehors d'applications particulières. Ce problème se révèle sous deux angles différents:

- le processus de gestion de ressources ne possède pas de connaissance liée à l'historique de la synthèse du plan. En particulier, en cas de non-cohérence d'un plan au niveau des ressources, ce processus ne peut pas retourner d'informations très riches en direction de la synthèse de plans ; et
- le processus de synthèse de plans ne possède pas de connaissance sur les ressources et leur disponibilité. Cela peut lui faire rater un certain nombre de décisions relatives à l'allocation de ressources ou à l'ordonnancement qu'il aurait été opportun de prendre en compte plus tôt dans le processus global.

SIPE a été le premier système de synthèse de plans d'actions à intégrer la notion de ressource. Le système permet la représentation de ressources consommables afin d'élaguer l'arbre de recherche de toutes les branches représentant des plans partiels sur-consommateurs de ressource [Wilkins 88, Wilkins 92]. SIPE utilise des opérateurs avec variables et permet de représenter une dépendance fonctionnelle entre certaines variables de l'action et la quantité de ressources consommée par celle-ci. Ces relations de dépendance peuvent être exploitées pour filtrer les domaines des variables en éliminant certaines valeurs qui conduiraient nécessairement à une surconsommation de ressource. SIPE code aussi la notion de ressource non-partageable en ordonnançant les conflits d'accès à de telles ressources.

O-Plan2 [Currie 91, Tate 94] gère le même type de ressources que SIPE-2. Les contributions principales de O-Plan2 concernent:

- l'analyse du plan qui consiste à détecter un ensemble de défauts (*flaws*) et permet à un module de contrôle une gestion *opportuniste* de ces défauts. De manière générale, l'opportunité de résoudre un défaut est liée à sa nature, au nombre de méthodes de résolution associées à ce défaut et à sa position temporelle ;
- la gestion de ressources partageables à l'aide d'un "critère" de ressources basé sur l'analyse de profils optimistes et pessimistes d'utilisation de ressources [Drabble 94]. Ce critère semble permettre de détecter et de résoudre un sous-ensemble des conflits de ressources. Dans le cadre d'un planificateur poursuivant une stratégie de moindre engagement et tendant à éviter une linéarisation totale du plan, ce critère ne semble toutefois pas être suffisant pour assurer la correction de la recherche à moins de faire des hypothèses fortes sur la manière dont le plan est généré.

Un autre type d'approche pour vérifier la validité d'un plan partiel au niveau des ressources est d'utiliser des méthodes de simulation en tirant plusieurs instances temporelles aléatoires du plan partiel et en analysant sur ces échantillons la disponibilité des ressources à des moments sélectionnés [Miller 85, Muscettola 87, Muscettola 93]. Il est bien entendu très

difficile de prouver des résultats formels pour ces approches même si elles se révèlent efficaces pour certains problèmes. De telles approches peuvent aussi être utilisées pour donner au module de contrôle des heuristiques permettant d'effectuer des choix tactiques.

Citons enfin les travaux autour de *parc*PLAN [Lever 94, El-Kholy 96] qui traitent un problème très similaire à celui que nous nous proposons de traiter mais en suivant une approche relativement différente. *parc*PLAN permet la représentation de l'utilisation de ressources non-partageables ou partageables par les actions sous la forme de prédicats $use(ressource)@[début, fin)$ ou $use(quantité)@[début, fin)$. La représentation des conditions et effets classiques des actions s'effectue grâce à une logique réifiée avec un prédicat affirmant la persistance d'un fait sur un intervalle temporel : $fait@[début, fin)$. Toutes les contraintes du plan (temporelles ou non) sont gérées par le langage de programmation par contraintes ECL^iPS^e.

Dans *parc*PLAN une première phase (phase de faisabilité) consiste à sélectionner un ensemble d'actions en ignorant les interactions entre les sous-buts. Cette phase permet de générer l'ensemble des variables et des contraintes du plan. La seconde phase consiste ensuite à gérer en concurrence les menaces et les conflits de ressources sous la forme d'un problème classique de programmation logique par contraintes. Cette approche est un pas vers une intégration des processus de synthèse de plans et de gestion de ressources mais, comme le font remarquer les auteurs, le retour-arrière sur la sélection de l'ensemble d'actions qui sont présentes dans le plan est un problème très difficile. On notera aussi dans [El-Kholy 96] une technique originale de recherche et de résolution des conflits sur une ressource partageable à l'aide de méta-variables booléennes associées à un couple d'intervalles utilisateurs de ressources en fonction de leur position temporelle relative.

1.6 Positionnement de notre approche

Comme on le voit, la majorité des systèmes intégrant réellement gestion de ressources et synthèse de plans abandonnent souvent, du fait de la complexité du problème, des visées plus formelles comme la preuve de la complétude voire de la correction de l'algorithme.

L'approche que nous proposons dans ce mémoire tente de répondre à un triple objectif :

1. permettre la représentation d'un large éventail de problèmes de planification au sens large du terme;
2. permettre une résolution performante et efficace de ces problèmes en terme de temps de calcul aussi bien qu'en terme de qualité de la solution trouvée ; et
3. assurer des propriétés formelles essentielles telles que la correction et la complétude de la recherche.

Pour cela nous apportons des idées originales mais aussi en empruntons un certain nombre aux travaux cités dans ce chapitre (voir la figure 1.9).

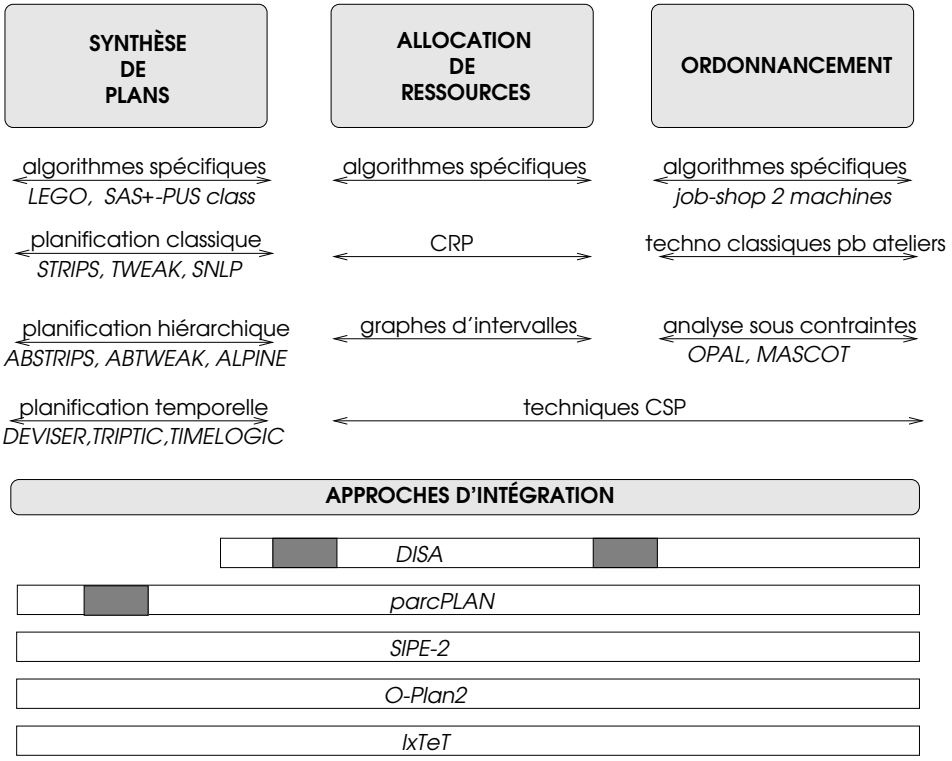


Figure 1.9: Bilan de quelques techniques et travaux en planification

Les blocs grisés représentés sur certaines approches d'intégration signifient que, bien que pour ces sytèmes il y ait intégration de la représentation du problème, la résolution s'effectue toujours, dans une certaine mesure, de manière séquentielle de la synthèse de plans vers l'ordonnancement.

Tout d'abord, nous nous basons sur l'acquis lié au planificateur temporel IxTeT , sa représentation du temps [Mounir-Alaoui 90, Vidal 95] et le principe de son contrôle de la recherche [Laruelle 94]. Nous étendons et quantifions la notion de *moindre engagement* utilisée dans la majorité des planificateurs classiques. D'autre part, nous enrichissons les travaux classiques sur la planification hiérarchique en introduisant la notion de *hiérarchie dynamique*.

Nos emprunts aux techniques d'allocation de ressources sont liés à l'utilisation de *graphes d'intervalles* et à la gestion, comme dans le cadre des *CRP*, d'un réseau de contraintes sur les variables représentant les ressources allouées aux tâches.

Enfin, les approches d'*analyse sous contraintes* nous ont aussi inspiré pour ce qui est de l'ordonnancement de ressources partageables avec IxTeT .

Chapitre 2

Représentation

2.1 Introduction

La représentation utilisée dans IxTeT est le fruit d'un compromis entre une grande expressivité pour pouvoir décrire des problèmes réalistes et un souci d'efficacité globale et de complétude du système.

Dans toute la gamme de problèmes qui va de la synthèse de plans d'actions à l'ordonnancement, le *temps* joue un rôle essentiel de référentiel permettant de positionner les activités. Une des caractéristiques de la représentation concerne donc le temps et l'expression de contraintes temporelles entre activités. La synthèse de plans, qui consiste à atteindre certains objectifs en fonction d'une description dynamique de l'environnement nécessite une formalisation précise du *monde* et du *changement* qui soit adaptée à la représentation temporelle. Une large gamme de *ressources* doit pouvoir être prise en compte pour modéliser certaines contraintes physiques de l'environnement. Enfin, les *opérateurs de planification* doivent eux-mêmes être très expressifs puisqu'ils constituent les futures briques de base de nos plans.

Nous verrons dans ce chapitre comment IxTeT répond à ces multiples exigences sans perdre de vue les contraintes d'efficacité et de complétude de la recherche ; lesquelles contraintes nous amènent à faire certaines hypothèses restrictives sur la représentation comme la *complétude de la représentation du monde* ou le *déterminisme* des opérateurs.

Dans la dernière partie de ce chapitre, le formalisme IxTeT ainsi défini sera comparé au formalisme STRIPS bien connu de la littérature.

2.2 Le temps

Planifier consiste avant tout à organiser des actions dans le *temps*. Un système de planification réaliste nécessite donc une représentation riche et précise du temps et des contraintes temporelles.

Au cours des dernières années, plusieurs formalismes ont été proposés pour raisonner sur le temps et sur les contraintes temporelles. Notre objet n'est pas ici d'en faire un état de

l'art complet mais plutôt d'en éclairer certains aspects afin de mieux situer et de justifier les choix effectués dans IXTET . Historiquement, deux approches peuvent être distinguées, lesquelles approches tirent leur origine des différents problèmes qui les sous-tendent: l'approche *symbolique* et l'approche *métrique*.

2.2.1 L'approche symbolique

L'algèbre d'intervalles de Allen [Allen 84] représente les treize relations de base (ou **primitives**) pouvant exister entre deux intervalles temporels quelconques :

$$B = \{b, m, o, s, f, d, eq, b', m', o', s', f', d'\}$$

(cf. figure 2.1 et les relations inverses).

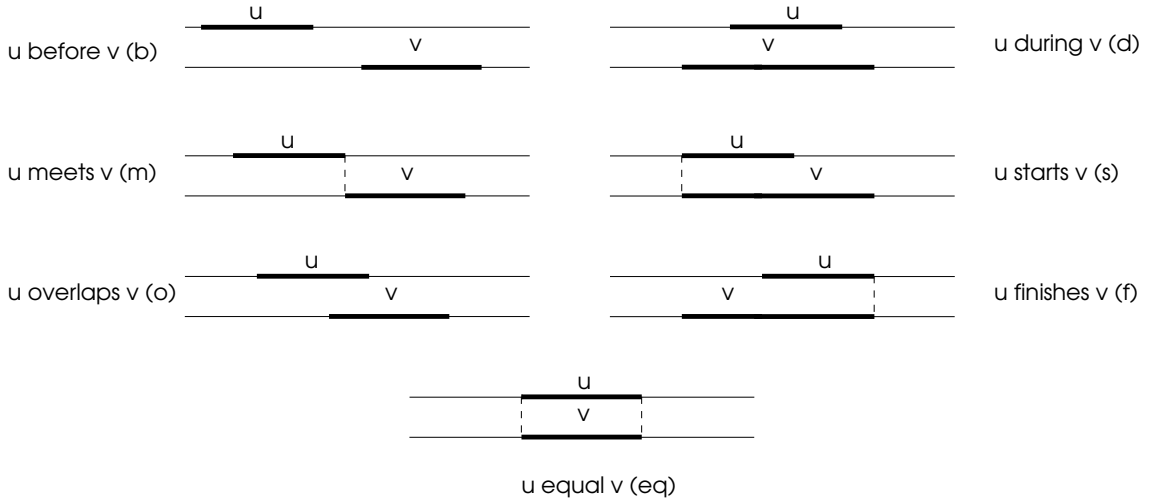


Figure 2.1: Les relations de base de l'algèbre d'intervalles d'Allen

Toute incertitude ou contrainte sur la position relative de deux intervalles peut ainsi être représentée par une *disjonction* de ces primitives c'est-à-dire un élément de 2^B. Par exemple : $disjoint = b \vee b'$.

La loi de *composition* permet de déduire la relation liant u à w lorsque l'on connaît la relation liant u à v et celle liant v à w: si $q(u, v)$ et $r(v, w)$ alors $q \circ r(u, w)$.

Les relations de *disjonction* ($\bigvee$) et de *composition* ($\circ$) sur 2^B confèrent à $(2^B, \bigvee, \circ)$ une structure d'algèbre.

Un réseau de contraintes sur l'algèbre d'intervalles est un graphe orienté dont les nœuds sont constitués des intervalles temporels et les arcs des relations composées de l'algèbre d'intervalles exprimant des contraintes entre ceux-ci. Etant donné un réseau de contraintes sur l'algèbre d'intervalles, une solution à ce réseau est un étiquetage globalement cohérent de l'ensemble des arcs par une primitive (cf. figure 2.2, partie droite).

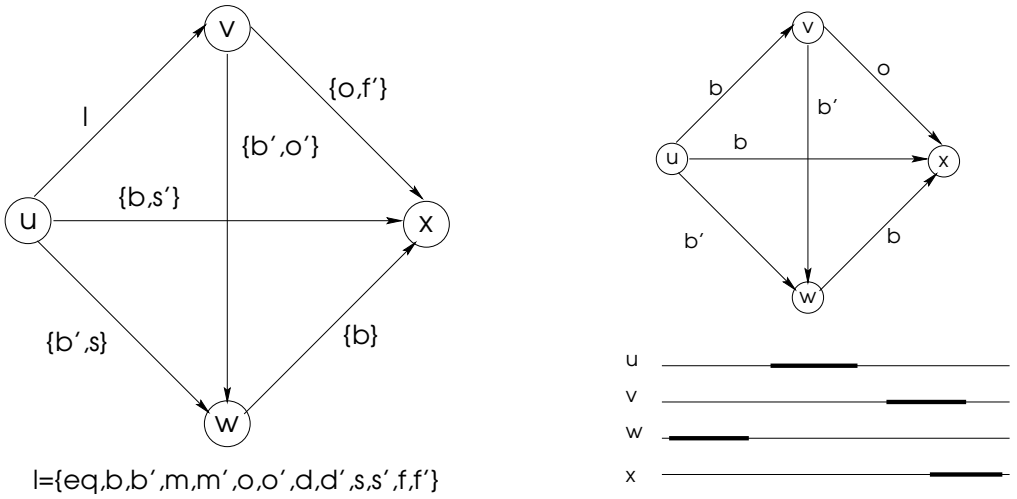


Figure 2.2: Un réseau de contraintes sur l'algèbre d'intervalles

Un résultat classique, établi dans [Vilain 86], conclut en la NP-complétude des problèmes de vérification de la cohérence du réseau (existence d'au moins une solution) et de recherche d'une solution particulière.

Ce résultat a été étendu dans [Golumbic 93] en montrant que ces problèmes demeurent NP-complet même avec un formalisme plus grossier que l'algèbre d'intervalles de Allen. Nous résumons ici ce travail car il fait un lien intéressant avec la théorie des graphes, un peu à la manière de ce qui sera utilisé dans IxTeT pour la gestion de ressources. Les auteurs y définissent, en plus des primitives de l'algèbre d'intervalles d'Allen les macro-primitives suivantes:

$$\cap = \{m, m', o, o', s, s', f, f', d, d', eq\},$$

$$\alpha = \{m, o\}, \quad \alpha' = \{m', o'\},$$

$$\subset = \{s, f, d\}, \quad \subset' = \{s', f', d'\}$$

Les algèbres $\mathcal{A}_3$, $\mathcal{A}_7$ et $\mathcal{A}_{13}$ sont définies comme celles engendrées respectivement par les ensembles de primitives $\{b, b', \cap\}$, $\{b, b', \alpha, \alpha', \subset, \subset', eq\}$ et $\{b, m, o, s, f, d, eq, b', m', o', s', f', d'\}$ ($\mathcal{A}_{13}$ n'est autre que l'algèbre d'intervalles classique). Les problèmes d'existence et de recherche d'une solution restent NP-complet sur $\mathcal{A}_3$ et $\mathcal{A}_7$.

Plusieurs restrictions des algèbres $\mathcal{A}_3$, $\mathcal{A}_7$ et $\mathcal{A}_{13}$ ont été proposées pour limiter leur complexité et appliquées avec succès à des applications particulières. De manière générale, *restreindre* une algèbre d'intervalles $\mathcal{A}_n$ consiste à ne conserver de l'algèbre qu'un sous-ensemble Δ de ses $2^n - 1$ relations pour lequel la loi de composition $\circ$ reste une loi interne.

Une des principales restrictions de $\mathcal{A}_{13}$ est l'*algèbre d'intervalles restreinte* (AIR) qui est constituée par les 187 relations de l'algèbre d'intervalles d'Allen qui peuvent s'exprimer sous la forme d'une conjonction de contraintes de précédence ($\prec$) ou d'égalité ($=$) sur les instants extrémités des intervalles [Granier 88]. Par exemple : $u\{b, m, o\}v \Leftrightarrow (u^- \prec v^-) \wedge$

$(u^+ \prec v^+)$ appartient à l'AIR, $\{b, b'\}$ n'y appartient pas. Une des propriétés essentielles de l'AIR est que les problèmes d'existence et de recherche d'une solution peuvent y être résolus en temps polynômial [Vilain 86]. Très souvent, les problèmes exprimés dans l'algèbre d'intervalles restreinte sont traduits puis gérés dans l'*algèbre d'instants*, algèbre générée par les disjonctions des trois primitives $\prec$, $=$ et $\succ$ entre deux instants. Les problèmes posés sur l'algèbre d'instants (vérification de cohérence, recherche d'une solution) sont en général résolus de manière efficace par des algorithmes spécifiques [Ghallab 89b][van Beek 90].

Golumbic et Shamir proposent dans [Golumbic 93] un certain nombre de restrictions polynômiales de $\mathcal{A}_3$. Ils montrent en particulier, que si l'on ôte de $\mathcal{A}_3$ la relation de disjonction entre intervalles $\{b \vee b'\}$, on obtient la restriction $\Delta_1 = \{b, b', \cap, b \vee \cap, \cap \vee b', b \vee \cap \vee b'\}$ qui est polynômiale car c'est un sous-ensemble de l'AIR. Un résultat plus intéressant est que la restriction $\Delta = \{b, b', \cap, b \vee b'\}$ est elle aussi polynômiale bien que contenant la disjonction $b \vee b'$; ce dernier résultat s'appuie notamment sur le fait que dans un graphe de contraintes satisfiables sur Δ, le sous-graphe uniquement constitué par les arcs $\cap$ est un *graphe d'intervalles*[1] et que de tels graphes peuvent être reconnus en temps polynômial. La figure 2.3 résume les différentes algèbres et restrictions discutées ci-dessus ainsi que leur complexité et leur inclusion mutuelle.

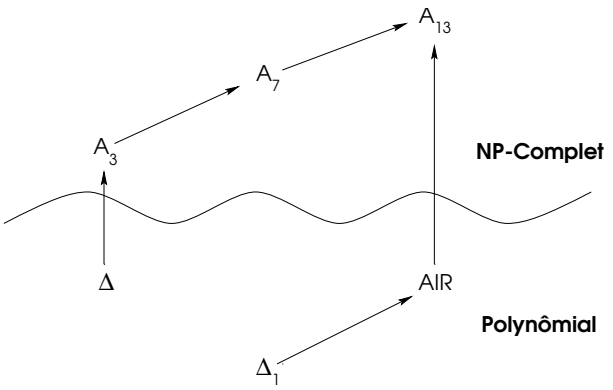


Figure 2.3: Quelques algèbres d'intervalles et restrictions associées

2.2.2 L'approche métrique

Dans la vie courante, le temps est avant tout une notion qui se mesure et beaucoup d'applications réelles, en ordonnancement et en planification notamment, nécessitent une représentation métrique du temps.

Autant l'intervalle est une notion très naturelle lorsqu'on traite de contraintes symboliques (on imagine très bien ce que peuvent être les positions relatives de deux activités dans le temps représentées par des intervalles), autant cette notion devient complexe voire inadéquate en tant que primitive lorsqu'on y introduit des informations métriques. La raison en

[1] voir l'annexe B pour des informations complémentaires au sujet des graphes d'intervalles.

est très simple et facilement transposable dans l'espace : alors qu'une description symbolique de la position relative de deux éléments peut être faite sans entrer dans les détails de la constitution de ceux-ci (ex : tel objet est au-dessus de tel autre), une description métrique nécessite de représenter l'objet sous la forme de points pour la bonne raison que la définition d'une distance fait toujours intervenir le point comme primitive de base. De la même manière, les approches métriques utilisent en général l'*instant* comme primitive de base.

En ordonnancement, les contraintes temporelles métriques sont classiquement représentées sous la forme d'un *graphe potentiel-tâches* $G = (E, V)$ où $E = \{u_1, ..., u_n\}$ est un ensemble d'intervalles temporels (les activités à ordonnancer) et V un ensemble d'arcs (u_i, u_j) valués par la durée minimale entre l'instant de début de u_i et celui de u_j. Les graphes potentiel-tâches permettent en particulier de définir les *chemins critiques* (plus longs chemins entre l'opération initiale u_0 et l'opération finale u_*) et de déterminer, pour chaque opération, les ordonnancements au plus tôt et au plus tard compatibles avec ces chemins critiques (cf. figure 2.4).

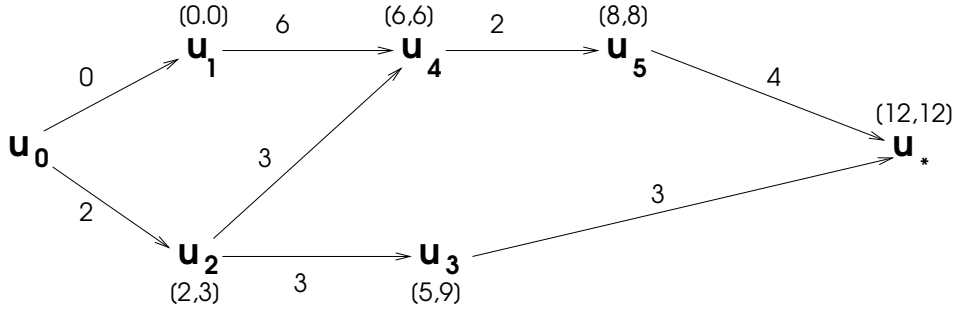


Figure 2.4: Graphe potentiel-tâches

Un cadre beaucoup plus général pour la représentation de contraintes métriques est fourni par les *Temporal Constraint Satisfaction Problems* [Dechter 91] qui sont un cas particulier de CSP[2] dont:

- les variables sont les instants du problème T ;
- les contraintes sont, soit des contraintes unaires sur la date d'échéance des instants (si $t_i \in T$, $d(t_i) \in [a_1, b_1] \vee ... \vee [a_n, b_n]$), soit des contraintes binaires entre deux instants $(d(t_j) - d(t_i) \in [a_1, b_1] \vee ... \vee [a_n, b_n])$.

Bien que ce formalisme ne permette pas d'exprimer toutes les relations symboliques de l'algèbre d'Allen (il faudrait pour cela pouvoir exprimer des contraintes quaternaires sur les instants), il n'en demeure pas moins extrêmement riche et la résolution du problème de cohérence globale est NP-difficile à cause des disjonctions sur les intervalles de dates. Les auteurs extraient un sous-problème des TCSP soluble en temps polynômial : le cas où toutes les contraintes portent sur des intervalles convexes de $\mathbb{R}$ (*Simple Temporal Networks*

[2] voir l'annexe C pour une brève introduction sur le formalisme général des CSP et les techniques classiques de résolution.

ou STP). Les STP constituent une formalisation mathématique de l'approche décrite dans [Dean 87]. Un algorithme de filtrage par consistance de chemin [Mackworth 77] permet, dans le cas particulier des STP, de vérifier la cohérence *globale* du réseau et de propager les contraintes temporelles numériques avec une complexité en $O(n^3)$.

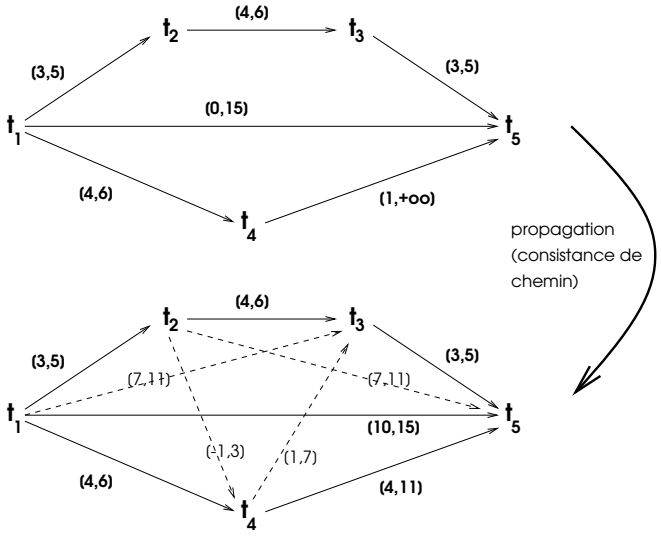


Figure 2.5: Propagation des contraintes sur un STP

Récemment, plusieurs approches sont allées dans le sens d'une intégration des contraintes métriques avec les contraintes symboliques de l'algèbre d'Allen afin d'augmenter l'expressivité de la représentation. Dans [Meiri 91], contraintes symboliques et contraintes métriques sont intégrées dans un même réseau de contraintes d'une grande expressivité. Les primitives sont à la fois les instants et les intervalles, et les contraintes, des relations de l'algèbre d'Allen, des relations instant/intervalle ou bien des contraintes métriques entre instants. La gestion et la résolution du réseau s'effectue par des techniques classiques de CSP (filtrage, recherche avec retour-arrière, ...). Une approche alternative, décrite dans [Kautz 91], gère les deux types de contraintes dans deux réseaux différents : un réseau qualitatif représentant les contraintes d'Allen et un réseau métrique de type STP. Les interactions entre les deux réseaux sont assurées par des algorithmes spécifiques de traduction des contraintes.

2.2.3 Représentation du temps dans IxTeT

Comme la planification est avant tout un raisonnement sur le *changement* du monde et que, dans notre approche, nous considérons le changement comme *instantané*, une représentation du temps basée sur les instants est des plus adéquates. Dans notre représentation, tous les changements du monde, qu'ils soient prévisibles indépendamment des opérations du plan ou qu'ils soient planifiés, seront associés à un instant particulier.

En cours de planification et indépendamment de l'expressivité du formalisme temporel, les fonctionnalités demandées au gestionnaire de contraintes temporelles sont les suivantes :

- **insertions**:
  - de nouveaux instants lors de l'ajout d'opérateurs dans le plan ;
  - de nouvelles contraintes temporelles lors de l'ajout d'opérateurs ou lors de la résolution de conflits ;
- **requêtes** :
  - sur la cohérence globale du réseau de contraintes ; ou
  - sur la position relative de deux instants et la possibilité d'ajouter une contrainte de précédence sans remettre en cause la cohérence globale.

Planifier est une activité complexe où chacune des décisions prises quant à l'élaboration du plan peut être lourde de conséquences pour la suite de la recherche. Comme nous le préciserons par la suite, nous considérons que l'étape d'*analyse* du plan courant, destinée à choisir la prochaine transformation qui sera appliquée à celui-ci, est une étape cruciale qui doit être la plus précise et la plus informée possible. Or cette étape met en œuvre quantité de requêtes temporelles. Les insertions, elles, n'ont lieu qu'après qu'une décision de transformation du plan courant ait été prise c'est-à-dire beaucoup plus rarement. Dans la pratique, les requêtes sont donc beaucoup plus fréquentes que les insertions. Des mesures effectuées font état d'un rapport variant entre 10 et 50 selon le problème. Cette dissymétrie entre requêtes et insertions doit être prise en compte pour la conception d'un gestionnaire de contraintes efficace.

Dans le cadre d'IxTeT , un premier formalisme temporel basé sur l'algèbre d'instants a été défini dans la thèse d'Amine Mounir-Alaoui [Mounir-Alaoui 90]. Le réseau de contraintes permet d'exprimer des contraintes de précédence et d'inégalité entre instants et d'insérer de nouveaux instants. Des algorithmes efficaces de gestion du réseau de contraintes symboliques sont donnés qui permettent à la fois une propagation des contraintes et une réponse aux requêtes avec une complexité moyenne qui évolue linéairement en fonction du nombre d'instants du réseau.

Gérer uniquement des contraintes temporelles qualitatives de précédence, d'égalité ou d'inégalité entre instants est toutefois insuffisant pour pouvoir représenter et résoudre des problèmes réalistes où la durée des activités, les créneaux de disponibilité des ressources et les dates limites sur la réalisation du plan nécessitent une représentation métrique du temps. Les travaux décrits ci-dessus ont été étendus dans le cadre de la thèse de Thierry Vidal pour prendre en compte des contraintes métriques [Vidal 95]. Afin de pouvoir continuer à exploiter les performances du gestionnaire de contraintes symboliques, et partant de l'hypothèse que les contraintes numériques sont en minorité par rapport aux contraintes symboliques, deux réseaux de contraintes sont maintenus à jour : un premier réseau ($\mathcal{S}$) contenant tous les instants sur lesquels ne portent que des contraintes symboliques est géré par les algorithmes de propagation décrits dans [Mounir-Alaoui 90] tandis qu'un second réseau ($\mathcal{N}$), de taille plus modeste et contenant les autres instants est chargé de la gestion des contraintes

numériques grâce à des algorithmes de propagation classiques dans les STP. Les interactions entre les deux réseaux de contraintes sont assurées par des algorithmes spécifiques décrits dans [Ghallab 95a] et permettant de détecter toutes les contraintes de précédence induites dans un réseau et de les propager dans l'autre (cf. figure 2.6).

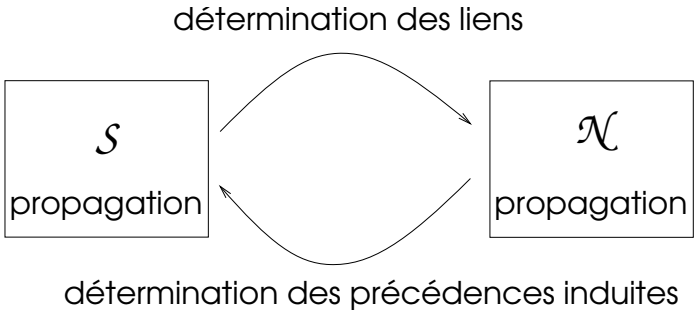


Figure 2.6: Interactions entre le réseau symbolique et le réseau numérique

Une étude montre que la complexité de la propagation d'une contrainte temporelle est alors en $O(p.(p+n+m^2))$ si n est le nombre d'instants de $\mathcal{S}$, m le nombre d'instants de $\mathcal{N}$ et p le degré de parallélisme maximal du réseau $\mathcal{N}$ défini comme la taille du plus grand ensemble d'instants non-reliés par une contrainte de précédence. Dans la mesure où m est faible devant n et où, dans les applications de planification, p reste faible en pratique (degré maximal de parallélisme du plan), cette complexité est à comparer avec ce que serait la propagation sur un STP contenant tous les instants en $O(n^2)$. Pour ce qui est des requêtes, leur complexité est en $O(p.n+p^2)$ soit une complexité linéaire pour p borné.

Une autre extension décrite dans [Ghallab 95b] concerne la prise en compte d'incertitudes dans les contraintes. Deux types de contraintes numériques sont distinguées : les contraintes **libres** dont la valeur peut a priori être librement choisie à l'intérieur du domaine et les contraintes **contingentes** dont la valeur est incontrôlable et imprévisible à l'intérieur du domaine (par exemple pour modéliser la durée d'une tâche). Cette extension est en cours d'intégration au système. Nous ne l'aborderons pas dans ce mémoire où nous supposerons que toutes les contraintes temporelles sont des contraintes libres telles que celles manipulées dans les TCSP classiques.

2.3 Les variables atemporelles

2.3.1 Variables et contraintes

L'expressivité de la représentation utilisée dans IxTeT passe par la représentation et la gestion de variables atemporelles[3] dans le plan. Ces variables sont issues des opérateurs de

[3]Dans la suite de ce mémoire, le terme de "*variable*" désignera uniquement les variables atemporelles ; pour les variables temporelles, nous emploierons le vocable d'"*instant*". Par convention et pour ne pas les confondre avec leurs valeurs possibles, nous faisons toujours précéder le nom d'une variable par un point d'interrogation: $?x$.

planification qui ont été insérés dans le plan et qui ne sont pas entièrement instanciés. Par exemple $ALLER\_DE\_A(robot, ?l, grande\_salle)$ avec $?l \in \{couloir, salleD\}$. A chacune des variables est associé un *domaine* qui représente les valeurs envisageables de celle-ci dans un plan solution. Au fur et à mesure de la planification, des *contraintes* sont posées sur les variables du plan pour assurer sa cohérence.

Dans IxTeT , nous considérons uniquement des variables à domaines *discrets* et *finis*. Le **gestionnaire de contraintes sur les variables** permet de propager les contraintes posées et de répondre à un certain nombre de requêtes. Le réseau de contraintes sur les variables peut être modélisé sous la forme d'un CSP binaire (V, D, C, R) (cf. Annexe C).

Les contraintes envisagées sont les suivantes:

1. **restriction** du domaine d'une variable, $?x_i \in D'_i$;
2. **égalité** de deux variables, $?x_i = ?x_j$;
3. **contrainte de différence** entre deux variables, $?x_i \neq ?x_j$; et
4. **dépendance** entre deux variables : si $?x_i$ prend sa valeur dans un ensemble D'_i alors, nécessairement, $?x_j$ doit prendre la sienne dans un ensemble $?D_j$, $?x_i \in D'_i \Rightarrow ?x_j \in D'_j$.

Les contraintes de *restriction de domaine* et de *dépendance* proviennent essentiellement des contraintes internes aux opérateurs de planification.

Par exemple, étant donné les connexités entre pièces décrites sur le plan de la figure 2.7, l'opérateur $ALLER\_DE\_A(?robot, ?lieu_{départ}, ?lieu_{arrivée})$ impose entre autre les contraintes :

$$?lieu_{départ} \in \{couloir, salleA, salleB, salleC, salleD, grande\_salle\}$$

$$?lieu_{départ} \in \{salleA, salleB, salleC\} \Rightarrow ?lieu_{arrivée} \in \{couloir\}$$

$$?lieu_{départ} \in \{grande\_salle\} \Rightarrow ?lieu_{arrivée} \in \{couloir, salleD\}$$

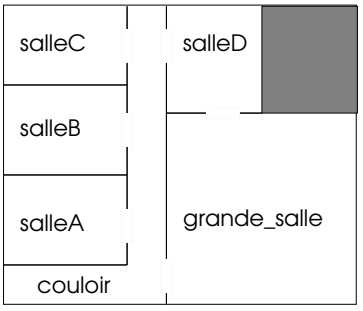


Figure 2.7: Plan simplifié du laboratoire

Les contraintes d'*égalité* et de *différence* proviennent, soit de la définition initiale des opérateurs (par exemple la contrainte $?lieu_{départ} \neq ?lieu_{arrivée}$ dans l'opérateur

$ALLER\_DE\_A(?robot, ?lieu_{départ}, ?lieu_{arrivée})$), soit des contraintes rajoutées en cours de planification.

Les requêtes qui doivent être supportées par le gestionnaire sont les suivantes :

1. le réseau de contraintes est-il globalement consistant ?
2. est-il possible d'unifier deux variables $?x_i$ et $?x_j$ étant donné les contraintes déjà posées ?
3. est-il possible de poser une contrainte de différence entre deux variables $?x_i$ et $?x_j$ étant donné les contraintes déjà posées ?

Il est important de noter que les contraintes de dépendance ($\Rightarrow$) permettent d'exprimer toutes les contraintes binaires possibles entre deux variables[4]. L'expressivité du gestionnaire de contraintes sur les variables utilisé dans IxTeT est donc la même que celle des *CSP binaires généraux*. Une conséquence évidente est que répondre aux requêtes ci-dessus est un problème NP-complet. On peut d'autre part facilement montrer que même sans les contraintes de dépendance, les contraintes de différence suffisent à entraîner cette NP-complétude (en réduisant le problème à celui de la k-coloration d'un graphe, par exemple). Comme la quantité de contraintes et de requêtes devant être gérées en cours de planification est très élevée, nous nous contentons d'une propagation locale des contraintes de façon incrémentale dont la conséquence sera d'obtenir des réponses quelquefois un peu trop optimistes aux requêtes. Nous verrons par la suite que cela n'empêche en rien la complétude du système de planification du fait d'une vérification globale, *in fine*, de la consistance de tout plan solution proposé.

Dans notre représentation du réseau de contraintes, les contraintes d'égalité permettent de regrouper les variables en classes d'équivalence. Une classe d'équivalence ϵ représente un ensemble de variables égales. La notion de classe permet d'agglomérer tous les symboles de variables qui, du fait des contraintes d'égalité, représentent le même objet. La notion de domaine est alors rattachée à la classe plutôt qu'aux variables qui la constituent. Les autres types de contraintes (restriction, différence et dépendance) sont également exprimés au niveau des classes d'équivalence. Un exemple de réseau est présenté sur la figure 2.8.

2.3.2 Algorithmes de propagation

Si V désigne l'ensemble des variables, soit $\mathcal{E} = \{\epsilon_1, ..., \epsilon_p\}$ la partition de V en classes d'équivalence, $e : V \rightarrow [1, p]$ la fonction qui associe à chaque variable sa classe d'équivalence et $\mathcal{D} = \{\mathcal{D}_1, ..., \mathcal{D}_p\}$ les domaines de valeurs liés à chaque classe.

[4] Ainsi, la contrainte R_{13} pour le problème des 4 reines décrit en annexe C peut s'écrire à l'aide de 4 contraintes de dépendance : $x_1 \in \{a, c\} \Rightarrow x_3 \in \{b, d\}$, $x_1 \in \{b, d\} \Rightarrow x_3 \in \{a, c\}$, $x_3 \in \{a, c\} \Rightarrow x_1 \in \{b, d\}$, $x_3 \in \{b, d\} \Rightarrow x_1 \in \{a, c\}$

Propager une contrainte d'égalité consiste à fusionner les deux classes d'équivalence et à faire l'intersection des domaines. Une liste des contraintes d'inégalité ($C_{\neq}$) et des contraintes de dépendance ($C_{\Rightarrow}$) est conservée et mise à jour en fonction des contraintes insérées. Les algorithmes suivants décrivent le mode de propagation des différents types de contraintes utilisés dans IxTeT .

Algorithme PROPAGE_EGALITE($?x_i$,$?x_j$)
$\epsilon_{e(?x_i)} \leftarrow \epsilon_{e(?x_i)} \cup \epsilon_{e(?x_j)}$
$\mathcal{D}_{e(?x_i)} \leftarrow \mathcal{D}_{e(?x_i)} \cap \mathcal{D}_{e(?x_j)}$
$\mathcal{E} \leftarrow \mathcal{E} - \{\epsilon_{e(?x_j)}\}$
PROPAGE($\{\epsilon_{e(?x_i)}\}$)

Algorithme PROPAGE_RESTRICTION($?x_i$,D'_i)
$\mathcal{D}_{e(?x_i)} \leftarrow \mathcal{D}_{e(?x_i)} \cap D'_i$
PROPAGE($\{\epsilon_{e(?x_i)}\}$)

Algorithme PROPAGE_DIFFERENCE($?x_i$,$?x_j$)
$C_{\neq} \leftarrow C_{\neq} \cup \{DIFF(?x_i, ?x_j)\}$
PROPAGE($\{\epsilon_{e(?x_i)}, \epsilon_{e(?x_j)}\}$)

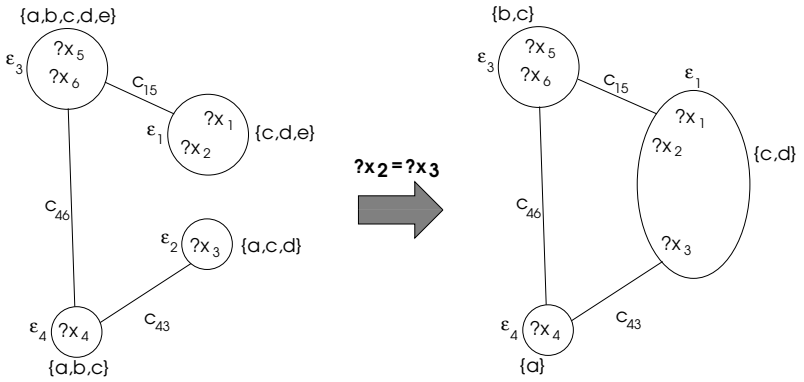
Algorithme PROPAGE_DEPENDANCE($?x_i$,D'_i,$?x_j$,D'_j)
$C_{\Rightarrow} \leftarrow C_{\Rightarrow} \cup \{DEP_{D'_i,D'_j}(?x_i, ?x_j)\}$
PROPAGE($\{\epsilon_{e(?x_i)}, \epsilon_{e(?x_j)}\}$)

L'algorithme de propagation décrit ci-dessous effectue une consistance d'arc sur le CSP des variables. Le parallèle avec l'algorithme général AC-3 de filtrage par consistance d'arc donné en annexe C est immédiat.

Algorithme PROPAGE($\mathcal{E}_{modif}$)
$\mathcal{L} \leftarrow$ contraintes de $C_{\neq}$ ou $C_{\Rightarrow}$ portant sur au moins une variable de $\mathcal{E}_{modif}$
tantque $\mathcal{L} \neq \emptyset$ **faire**
  choix puis suppression d'une contrainte $c(?x_i, ?x_j)$ dans $\mathcal{L}$
  si REVISE($c(?x_i, ?x_j)$) **alors**
    $\mathcal{L} \leftarrow \mathcal{L} \cup$ contraintes de $C_{\neq}$ ou $C_{\Rightarrow}$ portant sur $?x_i$ ou $?x_j$
  finsi
fintantque

Algorithme REVISE($c(?x_i, ?x_j)$)
$modification \leftarrow faux$
si c de type $DIFF$ **alors**
  si $|\mathcal{D}_{e(?x_i)}| = 1$ **et** $\mathcal{D}_{e(?x_i)} \subset \mathcal{D}_{e(?x_j)}$ **alors**
    $\mathcal{D}_{e(?x_j)} \leftarrow \mathcal{D}_{e(?x_j)} - \mathcal{D}_{e(?x_i)}$
    $modification \leftarrow vrai$
  si $|\mathcal{D}_{e(?x_j)}| = 1$ **et** $\mathcal{D}_{e(?x_j)} \subset \mathcal{D}_{e(?x_i)}$ **alors**
    $\mathcal{D}_{e(?x_i)} \leftarrow \mathcal{D}_{e(?x_i)} - \mathcal{D}_{e(?x_j)}$
    $modification \leftarrow vrai$
si c de type $DEP_{D'_i,D'_j}$ **alors**
  si $\mathcal{D}_{e(?x_j)} \cap D'_j = \emptyset$ **alors**
    $\mathcal{D}_{e(?x_i)} \leftarrow \mathcal{D}_{e(?x_i)} - D'_i$
    $modification \leftarrow vrai$
  si $\mathcal{D}_{e(?x_i)} \subset D'_i$ **alors**
    $\mathcal{D}_{e(?x_j)} \leftarrow \mathcal{D}_{e(?x_j)} \cap D'_j$
    $modification \leftarrow vrai$
retourne $modification$

Un exemple de propagation d'une contrainte d'égalité est donné sur la figure 2.8. Après fusion des domaines des deux classes d'équivalence ϵ_1 et ϵ_2, les contraintes qui sont révisées sont C_{15} et C_{43} puis C_{46}.



C_{15}	$?x_1 \in \{a,b,c,d\} \Rightarrow ?x_5 \in \{a,b,c\}$
C_{43}	$?x_4 \in \{b,c\} \Rightarrow ?x_3 \in \{a,e\}$
C_{46}	$?x_4 \neq ?x_6$

Figure 2.8: Insertion et propagation d'une contrainte d'égalité

La complexité de la propagation par consistance d'arc est linéaire en fonction du nombre de contraintes si on utilise un algorithme de type AC-4 [5]. D'autres algorithmes de filtrage par k-consistance, pour $k > 2$, ont été implémentés dans IxTeT ainsi qu'une procédure de recherche de solution. Toutefois ces algorithmes sont en général trop complexes pour être utilisés en cours de planification. Nous nous limitons donc au filtrage par consistance d'arc décrit ci-dessus. Après filtrage, les requêtes sont résolues de la manière suivante :

[5] Voir annexe C

1. incohérence du réseau : si un domaine $\mathcal{D}_i$ d'une classe est vide, il y a incohérence ;
2. possibilité d'unification de deux variables $?x_i$ et $?x_j$: l'unification est possible si $\mathcal{D}_{e(?x_i)} \cap \mathcal{D}_{e(?x_j)} \neq \emptyset$ et si aucune contrainte de différence n'a été posée entre $?x_i$ et $?x_j$;
3. possibilité de désunification de deux variables $?x_i$ et $?x_j$: la désunification est possible si $|\mathcal{D}_{e(?x_i)}| \neq 1$ ou $|\mathcal{D}_{e(?x_j)}| \neq 1$ ou $\mathcal{D}_{e(?x_i)} \neq \mathcal{D}_{e(?x_j)}$ (i.e. les deux domaines ne sont pas des singletons identiques) et si $?x_i$ et $?x_j$ n'appartiennent pas à la même classe.

Les requêtes sont effectuées en temps constant.

2.4 Les attributs d'état

2.4.1 L'état du monde

Dans IxTeT , la description du monde s'effectue au moyen d'un ensemble d'*attributs d'état.*

De manière informelle, un **attribut d'état** permet de représenter une caractéristique du monde qui ne peut prendre qu'une et une seule valeur à un instant donné. Par exemple, la position d'un robot, ou bien, l'état d'une machine.

Plus formellement, un attribut d'état est un tuple $(attS, ?x_1, ..., ?x_n)$ (noté plus aisément: $attS(?x_1, ..., ?x_n)$) où:

- $attS$ désigne le **nom** de l'attribut;
- $(?x_1, ..., ?x_n)$ est une liste de variables, instanciées ou non, appelées **arguments** de l'attribut;

Lorsqu'aucune confusion n'est possible, nous assimilerons l'attribut d'état à son nom $attS$.

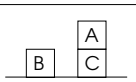
Un **attribut d'état instancié** est un attribut d'état dont tous les arguments sont instanciés.

Si $D_1, ..., D_n$ désignent respectivement les domaines des variables $?x_1, ..., ?x_n$, et si D_v désigne un domaine de valeurs, une **valuation** d'un attribut d'état $attS(?x_1, ..., ?x_n)$ est une fonction $v_{attS} : D_1 \times ... \times D_n \rightarrow D_v$ qui associe une valeur unique à chaque attribut d'état instancié.

Un **attribut valué** est un couple (attribut d'état instancié, valeur associée).

A un instant donné, l'état courant du monde est représenté par l'ensemble des valuations associées à chacun des attributs d'état du domaine. Nous faisons ainsi une hypothèse essentielle de **complétude de la description du monde**. La figure 2.9 décrit un exemple de représentation d'un état dans le monde des cubes.

Deux classes principales d'attributs d'état peuvent être distinguées:



attribute SUR(?x){
?x in {A,B,C};
?value in {A,B,C,Table};
}
	SUR(A)
	SUR(B)
	SUR(C)
		C
		Table
		Table

attribute LIBRE(?x){
?x in {A,B,C};
?value in {OUI,NON};
}
	LIBRE(A)
	LIBRE(B)
	LIBRE(C)
		OUI
		OUI
		NON

Figure 2.9: Exemple de représentation d'un état du monde

Les **attributs d'état rigides** sont tous ceux pour lesquels la valuation reste invariante au cours du temps : par exemple $CONNEXE(grande\_salle, salleD) : OUI$. Les attributs rigides interviennent dans les opérateurs pour exprimer certaines conditions. Dans IxTeT , ils sont gérés sous la forme de contraintes sur les variables.

Une valuation v_{attS} d'un attribut rigide $attS(?x_1, ..., ?x_n)$ à une valeur $?v$ peut en effet être considérée comme un ensemble de contraintes n+1-aire :

$$\forall (X_1, ..., X_n) \in D_1 \times ... \times D_n, (?x_1 = X_1) \wedge ... \wedge (?x_n = X_n) \Rightarrow ?v = v_{attS}(X_1, ..., X_n)$$

Ainsi la connexité sur l'exemple de la figure 2.7 : $(?lieu_1 = couloir) \wedge (?lieu_2 = salleA) \Rightarrow (?v = OUI)$, ..., $(?lieu_1 = salleA) \wedge (?lieu_2 = salleB) \Rightarrow (?v = NON)$,...

A noter que, comme à l'heure actuelle IxTeT gére uniquement des contraintes binaires, les seuls attributs rigides représentables sont ceux d'arité 1 (ou ceux d'arité 2 à valeur booléenne). La nécessité de représentation d'attributs rigides induisant des contraintes d'arité supérieure ne s'est pas faite sentir jusqu'à présent mais ils pourraient être pris en compte en utilisant des techniques classiques de propagations de contraintes n-aires sur des hypergraphes de contraintes.

Les **attributs d'état flexibles** sont ceux pour lesquels la valuation peut changer au cours du temps. Selon que ce changement peut être ou non contrôlé par l'agent qui planifie, nous parlons d'attributs d'état **contrôlables** ou **contingents**. Un exemple d'attribut contingent est l'éclairage par le soleil dont les évolutions jour/nuit ne sont pas contrôlables. On peut par contre considérer que la position d'un robot est un attribut contrôlable.

La représentation du monde par un ensemble d'attributs valués présente un avantage important par rapport à une représentation propositionnelle classique : elle permet de représenter de manière naturelle toutes les lois du monde qui imposent qu'un élément ne peut pas présenter simultanément plusieurs caractéristiques d'un même type. La contrainte de non-ubiquité en est un exemple particulièrement illustratif, tout comme les contraintes de changement d'état pour un système à états discrets et exclusifs.

2.4.2 La persistance et le changement

IxTeT repose sur une logique réifiée par le temps, à la manière de celle présentée dans [Shoham 88]. Dans le formalisme, deux prédicats permettent d'exprimer respectivement la *persistance* et le *changement*. Une formalisation plus poussée de la logique utilisée dans IxTeT peut être trouvée dans [Laruelle 94].

La persistance d'une valeur $V_{protégée}$ pour un attribut instancié $attS(X_1, ..., X_n)$ sur un intervalle temporel $[t_{début}, t_{fin}[$ est représentée par une **assertion** de la forme:

$$hold(attS(X_1, ..., X_n) : V_{protégée}, (t_{début}, t_{fin}))$$

Nous faisons l'hypothèse que nos attributs sont "homogènes" ou héritables sur les sous-intervalles ce qui est justifié puisqu'il s'agit de variables d'état du monde (par exemple vitesse(robot)=3m/s) plutôt que la description de changements en cours (le robot s'est déplacé de 3m).

Nous considérons le changement comme ponctuel et instantané. Si $attS$ est lié à un attribut flexible, le changement de valeur d'une ancienne valeur $V_{ancienne}$ à une nouvelle valeur $V_{nouvelle}$ d'un attribut instancié $attS(X_1, ..., X_n)$ à un instant $t_{changement}$ est représenté par un **événement** :

$$event(attS(X_1, ..., X_n) : (V_{ancienne}, V_{nouvelle}), t_{changement})$$

Il est clair que la valeur $V_{ancienne}$ doit préexister à $t_{changement}$ et que la valeur $V_{nouvelle}$ survivra à ce même instant. Réciproquement, pour un attribut donné, si une valeur V_1 est protégée jusqu'à un instant t et si une autre valeur V_2 est protégée à partir de t alors, nécessairement, un changement de valeur a dû avoir lieu à l'instant t. On a donc l'équivalence :

$$\begin{aligned} event(attS(X_1, ..., X_n) : (V_1, V_2)) \Longleftrightarrow \exists t_1, \exists t_2 / (t_1 \prec t \prec t_2) \\ \wedge hold(attS(X_1, ..., X_n) : V_1, (t_1, t)) \\ \wedge hold(attS(X_1, ..., X_n) : V_2, (t, t_2)) \end{aligned}$$

Toute évolution cohérente du monde peut donc être décrite, pour chaque attribut d'état instancié, comme une séquence d'événements instantanés séparés par des assertions qui décrivent la persistance des valeurs entre chaque événement (voir l'exemple sur la figure 2.10).

2.5 Les ressources et leur utilisation

2.5.1 Qu'est-ce qu'une ressource?

Nous définissons une **ressource** comme toute substance ou ensemble d'objets dont le coût ou la quantité disponible induit des contraintes sur les tâches qui l'utilisent. A chaque ressource

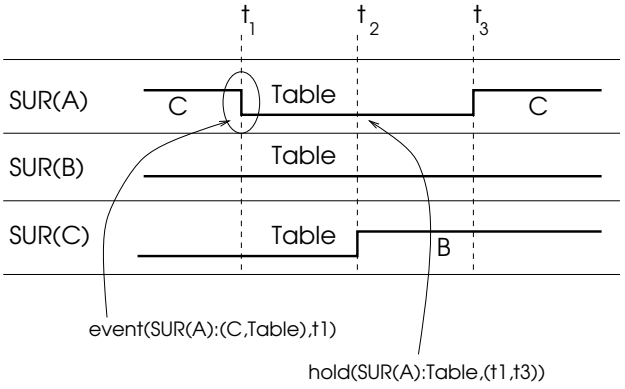


Figure 2.10: Une évolution des instances de l'attribut "SUR"

r est associée une **capacité** qui est un réel $capacity(r)$ qui représente la quantité disponible de cette ressource[6].

Plusieurs caractéristiques peuvent être attachées à une ressource.

La **granularité** d'une ressource est associée à la signification physique de la grandeur utilisée pour mesurer sa capacité maximale. On distinguera :

- les **ressources discrètes**, représentant un ensemble d'objets, pour lesquelles la capacité désigne la cardinalité de cet ensemble. Ces ressources se décomposent en ressources **unitaires** (ex : une machine donnée) et **non-unitaires** (ex : une flotille de robots, un ensemble de machines) selon ce même cardinal ;
- les **ressources continues**, représentant une substance (ex : carburant ou puissance électrique).

La granularité est donc une notion liée à la *nature* de la ressource.

Une ressource sera dite **individualisée** si certaines de ses sous-parties peuvent se distinguer des autres par des propriétés particulières (ex : un ensemble de robots ayant chacun un nom et positionnés en des lieux différents). Dans le cas contraire, la ressource n'est représentée que par une quantité numérique et nous parlerons de ressource **banalisée** (ex : une palette de briques). L'individualisation est en particulier liée au *niveau d'abstraction* utilisé dans la représentation des connaissances.

Pour une tâche donnée dans un environnement donné, plusieurs ressources peuvent être considérées comme équivalentes. Nous dirons que deux ressources sont de même **type** si et seulement si il existe au moins une tâche susceptible d'utiliser indifféremment l'une ou l'autre de ces ressources (cf. fig 2.11).

Nous associons à chaque type de ressources un **attribut de ressource** $attR(?x_1, ..., ?x_n)$ où :

[6]Dans notre implémentation, la capacité d'une ressource est un entier mais le fait de prendre un réel ne complexifie en rien nos algorithmes.

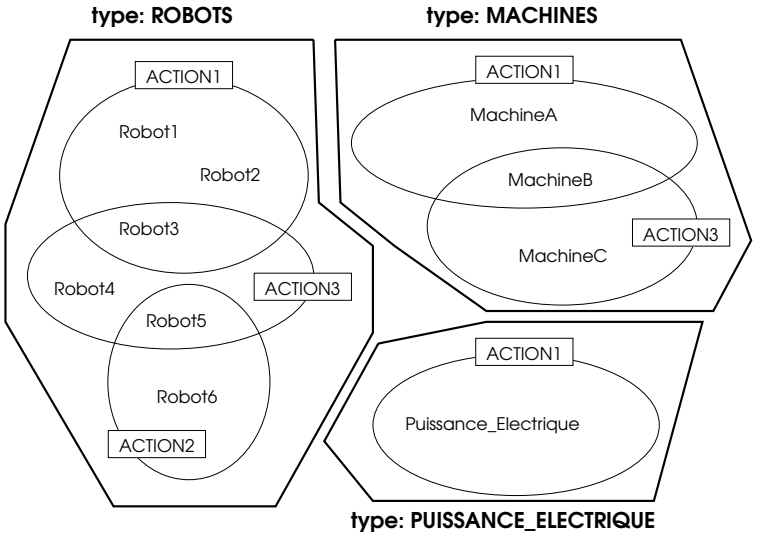


Figure 2.11: Décomposition en types de ressources

- $attR$ désigne le **type** de ressource ;
- $(?x_1, ..., ?x_n)$ est un vecteur de variables permettant d'identifier une ressource parmi l'ensemble des ressources du type. Bien souvent, un seul argument suffit dont le domaine est l'ensemble des ressources du type.

Deux exemples de déclaration de ressources sont donnés sur la figure 2.12. Lorsque toutes les ressources d'un type donné ont même capacité (en particulier lorsqu'il s'agit de ressources non-partageables comme les robots dans l'exemple), la notion de capacité est représentée non plus au niveau de la ressource mais au niveau de son type.

La différence essentielle entre un attribut de ressource et un attribut d'état est que l'état est modifié de manière *absolue* par l'action tandis que les ressources ne sont affectées que de manière *relative* en augmentant ou diminuant la quantité disponible. S'il est possible de représenter l'utilisation de ressources non-partageables par un attribut d'état (du type $ressource\_libre$ et prenant une valeur booléenne), l'utilisation de *ressources partageables* ne peut pas être représentée sous la forme d'événement sur des attributs d'état.

Cette représentation à base d'attributs permet d'exprimer, dans un même *formalisme homogène*, la plupart des ressources étudiées dans la littérature, aussi bien pour les problèmes d'ordonnancement que pour ceux d'allocation de ressources.

2.5.2 Utilisation des ressources

De la même manière que, dans le cadre de la logique réifiée, les attributs d'état sont qualifiés temporellement pour exprimer la persistance et le changement, les attributs de ressources le sont pour exprimer les différents modes d'utilisation d'une ressource par une tâche.

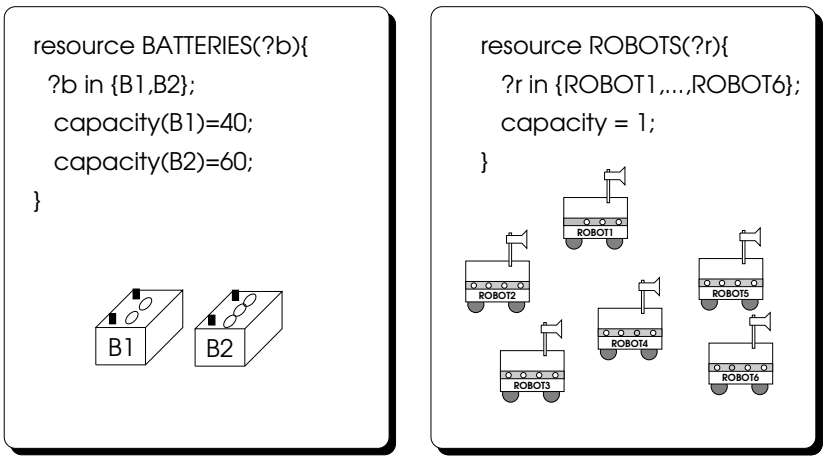


Figure 2.12: Deux attributs de ressources

L'**emprunt** d'une ressource $attR(X_1, ..., X_n)$ en une quantité donnée $q_{empruntée}$ sur un intervalle temporel $[t_{début}, t_{fin}[$ s'exprime grâce au prédicat *use* sous la forme :

$$use(attR(X_1, ..., X_n) : q_{empruntée}, (t_{début}, t_{fin}))$$

La **consommation** d'une ressource $attR(X_1, ..., X_n)$ en une quantité donnée q_{conso} à un instant t_{conso} est décrite grâce au prédicat *consume* :

$$consume(attR(X_1, ..., X_n) : q_{conso}, t_{conso})$$

La **production** d'une quantité q_{prod} d'une ressource $attR(X_1, ..., X_n)$ à un instant t_{prod} est décrite grâce au prédicat *produce* :

$$produce(attR(X_1, ..., X_n) : q_{prod}, t_{prod})$$

D'un point de vue logique, on a l'équivalence :

$$use(attR(X_1, ..., X_n) : q, (t_1, t_2)) \Longleftrightarrow \\ consume(attR(X_1, ..., X_n) : q, t_1) \\ \wedge\ produce(attR(X_1, ..., X_n) : q, t_2)$$

Il est à noter que, dans notre approche, nous ne caractérisons pas une ressource par le fait qu'elle soit productible, consommable ou uniquement empruntable, comme cela est souvent fait dans la littérature. Nous estimons que ces notions ne sont pas intrinsèques aux ressources mais fondamentalement liées à un couple tâche/ressource.

La figure 2.13 résume notre représentation.

Nous faisons l'hypothèse que toutes les quantités de ressources manipulées sont des *constantes*. En particulier, les ressources dont la quantité utilisée dépend de la durée de la tâche ou de la ressource effectivement allouée à la tâche ne sont pas, pour l'instant, supportées par notre représentation. D'autre part, nous supposons que toutes les quantités liées aux ressources évoluent de manière instantanée : les quantités évoluant continûment dans le temps comme celles étudiées dans [Penberthy 94] ne sont pas gérées.

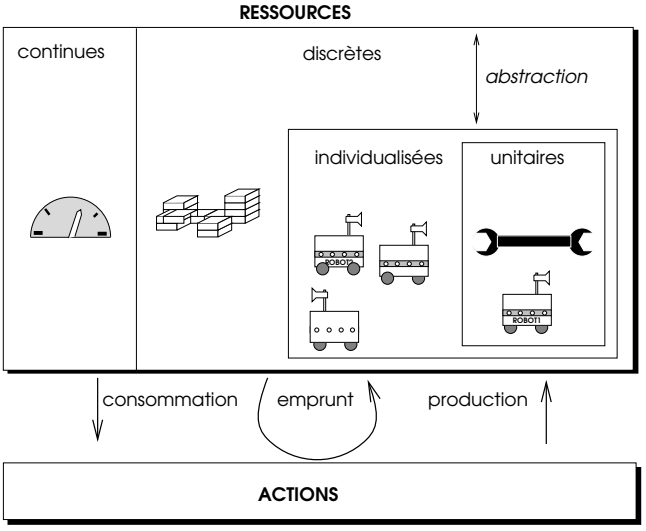


Figure 2.13: Représentation des ressources

2.6 Le formalisme de tâches

2.6.1 base de connaissances temporelles IxTeT

De manière générale, une **base de connaissances temporelles IxTeT** est une évolution ou une classe d'évolutions possibles du monde.

Une base de connaissances temporelles $\mathcal{B}$ est un tuple :
$\mathcal{B} = (EVT, HOLD, USE, CONS, PROD, C_V, C_T)$ où :

- EVT est un ensemble d'événements exprimant le changement de valeur de certains attributs d'état;
- $HOLD$ un ensemble d'assertions exprimant la persistance de certains faits ;
- USE un ensemble d'emprunts de ressources ;
- $CONS$ un ensemble de consommations de ressources ;
- $PROD$ un ensemble de productions de ressources ;
- C_T un ensemble de contraintes temporelles sur les instants référencés dans les propositions et exprimées dans le formalisme décrit en §2.2.3 ; et
- C_V un ensemble de contraintes sur les variables atemporelles référencées dans les propositions précédentes et exprimées dans le formalisme décrit en §2.3.1.

La notion d'ensemble est prise ici au sens conjonctif du terme.

Evénements, assertions, emprunts, consommations et productions de ressources sont indifféremment appelés **propositions temporelles**.

Nous dirons qu'une base $\mathcal{B}'$ **supporte** une base $\mathcal{B}$ (noté $\mathcal{B} \leftarrow \mathcal{B}'$) si et seulement si : $EVT \subset EVT'$, $HOLD \subset HOLD'$, $USE \subset USE'$, $CONS \subset CONS'$, $PROD \subset PROD'$, C'_T est un sous-réseau de C_T et C'_V est un sous-réseau C_V.

Nous appelerons **instance** d'une base de connaissances B tout tuple

$$\underline{\mathcal{B}} = (EVT, HOLD, USE, CONS, PROD, d_V, d_T)$$

où d_V et d_T sont respectivement des solutions des réseaux de contraintes C_V et C_T.

L'ensemble des instances d'une base de connaissances $\mathcal{B}$ sera noté $INST(\mathcal{B})$.

Dans IxTeT , les *opérateurs de planification* ainsi que le *scénario initial* sont représentés sous la forme d'une base de connaissance temporelle.

2.6.2 Les opérateurs de planification

Les schémas d'opérateurs de planification sont appelés **tâches**. Une hiérarchie de tâches est définie où chaque tâche $\mathcal{T}$ est composée :

- d'un ensemble de sous-tâches ;
- d'un ensemble d'événements permettant de décrire les changements du monde entraînés par l'exécution de la tâche ;
- d'un ensemble d'assertions sur des attributs d'état permettant d'exprimer certaines conditions à l'exécution de la tâche ainsi que certaines protections nécessaires à son bon déroulement;
- d'un ensemble de propositions d'utilisation de ressources (*use*, *consume* ou *produce*) spécifiant quelles ressources sont utilisées par la tâche et de quelle manière ;
- d'un ensemble de contraintes temporelles et de contraintes sur les variables liant les différentes variables (temporelles ou non) de la tâche.

Les sous-tâches auxquelles fait référence une tâche donnée sont spécifiées avec la même syntaxe. L'objectif de cette hiérarchie de représentation des opérateurs est de simplifier leur écriture par l'utilisateur. Cette hiérarchie n'est pas liée à la hiérarchie de contrôle qui sera décrite plus en détail au chapitre 5. Une phase de *compilation* des tâches permet de remettre la structure à plat et de représenter chaque tâche sous la forme d'une base de connaissances IxTeT .

Notre représentation est basée sur le fait que les tâches sont des opérateurs **déterministes** et que tout attribut dont la valeur n'est pas explicitement changée dans la déclaration d'une tâche n'est pas modifié par l'exécution de cette tâche.

Un exemple de tâche hiérarchique, pour un chantier de construction d'un bâtiment, est donné sur la figure 2.14. La tâche $TRANSFERER\_BRIQUES$ consiste à transférer un

nombre donné de briques entre deux salles. Elle se réfère pour cela aux deux sous-tâches $SAISIR\_BRIQUES$ dans la salle de départ et $DEPOSER\_BRIQUES$ dans la salle d'arrivée. A noter que les opérations de navigation entre salles ne sont pas intégrées à la tâche : elles seront insérées en cours de planification pour expliquer le fait que le robot doit se retrouver dans la salle d'arrivée avant de déposer les briques. Ceci permet, en particulier, d'exprimer la *sécabilité* de certaines tâches : ici, le robot pourra profiter de son déplacement entre salles pour faire autre chose (par exemple appuyer sur un interrupteur pour allumer la lumière) dans la mesure où cette activité ne viole pas les contraintes exprimées dans la tâche (le robot doit rester chargé et la durée maximale de la tâche de transfert ne doit en aucun cas excéder 30 mn).

```
task TRANSFERER_BRIQUES(?robot,?salle1,?salle2)(start,end){
  timepoint t1,t2;
  task SAISIR_BRIQUES(?robot,?salle1)(start,t1);
  task DEPOSER_BRIQUES(?robot,?salle2)(t2,end);
  hold(ETAT(?robot):CHARGÉ, (t1,t2));
  ?salle1 != ?salle2;
  (end - start) in (00:00:00,00:30:00);}
```

```
task SAISIR_BRIQUES(?robot,?salle)(start,end){
  event(ETAT(?robot):(VIDE,CHARGÉ), end);
  hold(POSITION(?robot):?salle, (start,end));
  use(ROBOTS(?robot):1, (start,end));
  consume(BRIQUES(?salle):20, start);
  (end - start) in (00:03:00,00:04:00);}
```

```
task DEPOSER_BRIQUES(?robot,?salle)(start,end){
  event(ETAT(?robot):(CHARGÉ,VIDE), end);
  hold(POSITION(?robot):?salle, (start,end));
  use(ROBOTS(?robot):1, (start,end));
  produce(BRIQUES(?salle):20, start);
  (end - start) in (00:03:00,00:04:00);}
```

Figure 2.14: Un exemple de tâche

Une tâche, de manière générale, représente un ensemble de conditions sur le monde et d'évolutions de celui-ci. La représentation des opérateurs sous la forme d'une base de connaissances IxTeT permet une grande souplesse. Il est par exemple possible d'utiliser une tâche dans un plan à cause de ses *effets transitoires* (ainsi, le fait que le robot reste chargé pendant une partie de la tâche $TRANSFERE\_BRIQUES$ pourrait être utilisé si le robot a besoin d'être lourd pour faire contrepoids à un treuil, par exemple). D'autre part, une tâche peut comporter des contraintes qui ne sont pas nécessairement cohérentes et contenir des conflits potentiels, conflits qui seront levés en cours de planification (ce serait par exemple le cas d'une tâche T référençant deux sous tâches T_1 et T_2 sans aucune contrainte de précédence entre elles mais qui utilisent une même ressource non-partageable. L'ordonnancement entre T_1 et T_2 sera alors choisi une fois la tâche insérée dans le contexte du plan, en cours de planification).

2.6.3 Le scénario initial

Le scénario initial $\mathcal{P}_0$ permet de représenter le problème de planification que l'on cherche à résoudre. Il s'agit d'une base de connaissance IxTeT qui décrit toutes les informations que

l'on possède sur le monde excepté un certain nombre de décisions qui seront prises par le système de planification. En général, le scénario initial contient les informations suivantes :

1. une description complète, sous la forme d'une liste d'événements et de productions de ressources, de l'état initial du monde. Dans IxTeT , comme nous ne disposons que des prédicats *event*, *hold*, *use*, *consume*, et *produce*, cela se fait à un instant fictif précédant l'instant initial ;
2. un ensemble d'événements et d'assertions sur des attributs d'état contingents décrivant toutes les persistances et changements prévus sur ces attributs-là (par exemple, les cycles jour/nuit pour la lumière naturelle) ;
3. un ensemble d'emprunts de ressource permettant de décrire les profils de disponibilité des ressources ;
4. un ensemble de tâches dont la présence est imposée dans tout plan solution ;
5. un ensemble de buts à atteindre, décrits sous la forme d'assertions ou d'événements à établir ;
6. un ensemble de contraintes temporelles liant les instants référencés dans les éléments ci-dessus ; et
7. un ensemble de contraintes sur les variables[7].

De cette manière, le scénario initial peut décrire aussi bien un pur problème de synthèse de plans (si aucune tâche imposée ne figure dans le scénario ; cf. point 4) qu'un pur problème d'ordonnancement (si le scénario ne contient aucun but ; cf. point 5). Tout panachage possible entre ces deux types de problèmes est bien sûr envisageable.

A noter particulièrement l'intérêt de pouvoir représenter un monde évolutif (points 2 et 3) pouvant changer indépendamment des tâches entreprises par l'agent planifiant à condition que le scénario d'évolution du monde soit connu au moment de la planification.

2.7 Comparaison avec le formalisme STRIPS

Dans ce paragraphe nous allons montrer que, bien que le formalisme IxTeT soit strictement plus riche que le formalisme STRIPS - du fait de la représentation explicite du temps, de l'utilisation d'attributs valués, de la gestion de ressources -, certains opérateurs STRIPS n'ont pas d'équivalents directs et uniques dans IxTeT si l'on se restreint aux seuls prédicats *hold* et *event* sur les attributs d'état. Nous proposons d'enrichir le formalisme IxTeT d'un nouveau prédicat qui permettrait de représenter n'importe quel opérateur STRIPS sous la forme d'une et d'une seule tâche IxTeT .

[7] Dans le scénario initial, tout comme dans toute base de connaissances IxTeT et tout CSP classique, il ne faut pas perdre de vue que le domaine associé à une variable possède une sémantique disjonctive et contrôlable. Ainsi un but $SUR(?x) : C$ avec $?x \in \{A, B\}$ par exemple, signifie que le système pourra choisir de résoudre $SUR(A) : C$ ou bien $SUR(B) : C$.

2.7.1 Le formalisme STRIPS

Le formalisme du premier ordre de type STRIPS que nous résumons ici est basé sur celui décrit dans [Erol 92].

Soit $\mathcal{L}$ un langage non-fonctionnel du premier ordre.

Dans le formalisme STRIPS, un **état du monde** est représenté par un ensemble S de littéraux instanciés de $\mathcal{L}$ qui décrit l'ensemble des propriétés vraies dans cet état-là. Par convention, toute propriété n'appartenant pas à un état S est fausse dans cet état-là (négation par l'absence).

Un modèle d'opérateur STRIPS Op est alors représenté sous la forme d'un quadruplet : $< Precond_+(Op), Precond_-(Op), Postcond_+(Op), Postcond_-(Op) >$ où :

- $Precond_+(Op)$ représente l'ensemble des littéraux devant être vrais avant d'exécuter l'opérateur Op ;
- $Precond_-(Op)$ représente l'ensemble des littéraux devant être faux avant d'exécuter l'opérateur Op ;
- $Postcond_+(Op)$ représente l'ensemble des littéraux qui seront rendus vrais par l'exécution de Op ;
- $Postcond_-(Op)$ représente l'ensemble des littéraux qui seront rendus faux par l'exécution de Op.

Un opérateur Op sera donc dit applicable dans un état S si et seulement si il existe une instanciation Ψ des variables de Op telle que :

$\Psi[Precond_+(Op)] \in S$ et $\Psi[Precond_-(Op)] \cap S = \emptyset$.

Si un opérateur Op est applicable dans un état S avec l'instanciation Ψ, l'état résultant de l'application de Op à S sera

$$S' = result(S, Op) = (S \cup \Psi[Postcond_+(Op)]) \backslash \Psi[Postcond_-(Op)]$$

Deux opérateurs STRIPS sont schématisés sur la figure 2.15. Le premier consiste, pour un robot, à passer d'un lieu à un autre ; le second représente le percement d'un trou dans un mur.

2.7.2 Le prédicat *assert*

La manière la plus naturelle pour passer du langage non-fonctionnel du premier ordre de STRIPS à la représentation valuée utilisée dans IxTeT est d'associer à chaque littéral STRIPS une valeur booléenne dans l'ensemble $\{vrai, faux\}$.

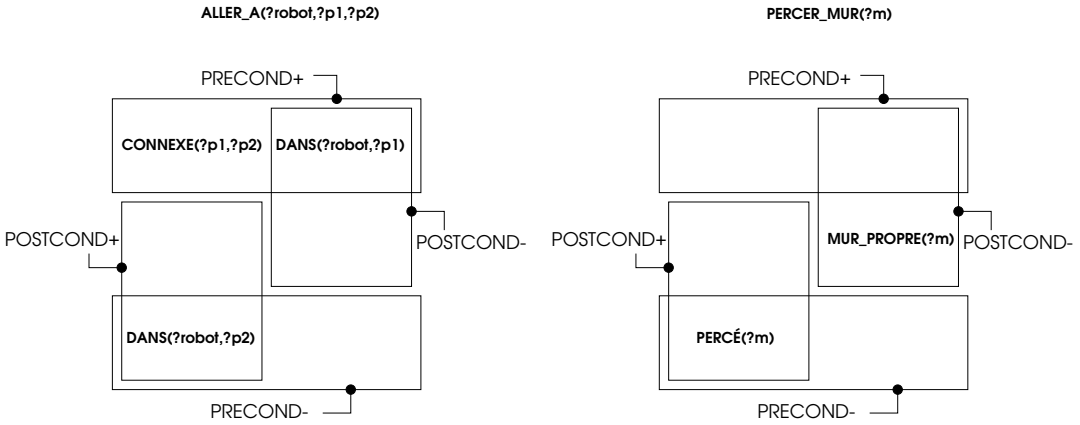


Figure 2.15: Deux opérateurs STRIPS

Dans notre système, un état du monde S_{IxTeT} sera donc représenté par l'ensemble des littéraux instanciés associés à leur valeur de vérité.

Etant donné un opérateur STRIPS Op, il sera facile de représenter avec IXTET toutes les préconditions de cet opérateur. En effet :

- si $p \in PRECOND_+ \cap POSTCOND_-$, la précondition et la postcondition associées peuvent toutes deux être représentées par un événement :
  $event(p : (vrai, faux), t)$ où t est l'instant correspondant à l'opérateur ;
- si $p \in PRECOND_+ \backslash POSTCOND_-$, la précondition associée peut être décrite par une assertion qui doit rester vraie pendant l'opérateur :
  $hold(p : vrai, (t - \epsilon, t + \epsilon))$;
- si $p \in PRECOND_- \cap POSTCOND_+$, la précondition et la postcondition associée peuvent toute deux être représentées par un événement :
  $event(p : (faux, vrai), t)$;
- si $p \in PRECOND_- \backslash POSTCOND_+$, la précondition associée peut être décrite par une assertion : $hold(p : faux, (t - \epsilon, t + \epsilon))$.

Cette représentation des préconditions permet, comme on le voit, de représenter du même coup un certain nombre de postconditions.

Il est par contre plus difficile de représenter les postconditions appartenant aux ensembles $POSTCOND_+ \backslash PRECOND_-$ et $POSTCOND_- \backslash PRECOND_+$. La raison est claire : le seul prédicat IXTET pour représenter le changement est l'événement or celui-ci suppose *nécessairement* une transition de valeur sur l'attribut. Pour une représentation bivaluée sur les booléens, cela entraîne que tout événement exprime aussi, en plus d'une postcondition associée à la nouvelle valeur de vérité, une précondition sur la valeur de vérité opposée.

Une postcondition p appartenant à $POSTCOND_+ \backslash PRECOND_-$ ou bien à $POSTCOND_- \backslash PRECOND_+$ ne peut donc pas être directement représentée sous la forme

d'un événement puisqu'elle n'est associée à aucune précondition. C'est par exemple le cas de la postcondition $MUR\_PROPRE(?m)$ dans l'exemple ci-dessus.

Si on n'utilise que les prédicats *event* et *hold*, la seule parade à ce problème de représentation consiste, pour chaque littéral p qui pose problème, à dupliquer l'opérateur STRIPS en deux tâches IxTeT . Par exemple si $p \in POSTCOND_- \backslash PRECOND_+$, l'un des opérateurs imposera une précondition positive sur p : $p \in PRECOND_+$ et contiendra l'événement $event(p : (vrai, faux), t)$, l'autre imposera une précondition négative $p \in PRECOND_-$. Le cas $p \in POSTCOND_+ \backslash PRECOND_-$ est symétrique en inversant $+$, $-$ et les valeurs de vérité des attributs (voir la figure 2.16).

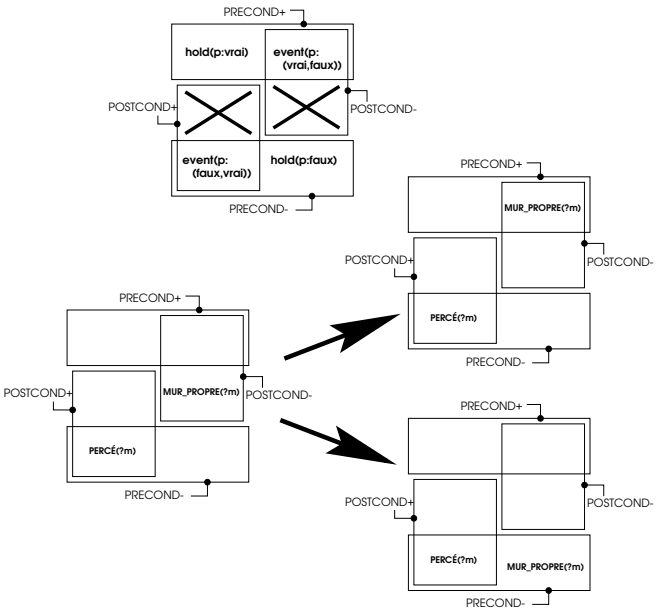


Figure 2.16: Un opérateur STRIPS, deux tâches IxTeT

Si un opérateur STRIPS contient n postconditions de ce type, il faudrait 2^n tâches IxTeT pour le représenter, ce qui deviendrait rapidement désastreux au niveau de la complexité de la planification.

Une alternative est d'enrichir le formalisme IxTeT d'un nouveau prédicat: *assert*, permettant d'exprimer un effet d'une tâche qui n'est associé à aucune précondition particulière.

$assert(attS(X_1, ..., X_n) : V_{assurée}, t)$ assure qu'après l'instant t, l'attribut $attS(X_1, ..., X_n)$ possèdera la valeur V, *quelle qu'ait été sa valeur avant* t.

Le prédicat *assert* permet ainsi de représenter tout opérateur STRIPS par une et une seule tâche IxTeT (cf. figure 2.17).

Le prédicat *assert* n'est pas à l'heure actuelle implémenté dans notre système. Il est néammoins facile de voir, en reprenant toutes les notions et algorithmes décrits dans ce mémoire que sa gestion n'entraîne aucune complexité supplémentaire par rapport au formalisme de base.

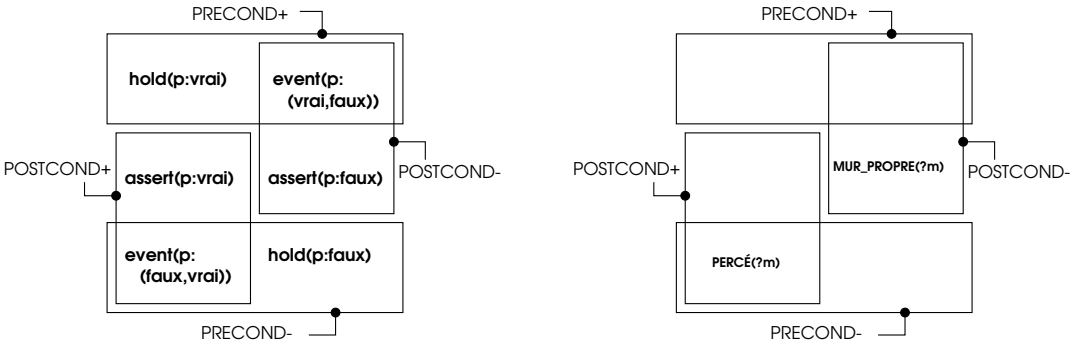


Figure 2.17: Un opérateur STRIPS, une tâche IXTET

A noter qu'une autre possibilité serait de modifier la sémantique de $event(attS(X_1, ..., X_n) : (V, V'), t)$ en tolérant $V = V'$.

2.8 Conclusion

Dans IXTET , aussi bien les opérateurs de planification appelés *tâches* que le *scénario initial* sont décrits sous la forme d'une *base de connaissances temporelles*. Une base de connaissances est constituée d'un ensemble de propositions temporelles exprimant la *persistance*, les *changements* du monde ainsi que les utilisations des différentes *ressources*. Ces propositions sont *contraintes* au niveau des instants et des variables auxquels elles se réfèrent. Les contraintes temporelles et les contraintes sur les variables sont gérées au moyen d'algorithmes spécifiques dans deux réseaux séparés.

La représentation ainsi définie, sous réserve que l'on y ajoute un nouveau prédicat, permet d'exprimer n'importe quel opérateur STRIPS. Mais son expressivité va bien au-delà grâce au formalisme temporel, à l'utilisation d'attributs valués et à la représentation de ressources. Nous allons voir dans les chapitres suivants comment cette représentation est supportée par le mécanisme de planification.

Chapitre 3

Contrôle de base

3.1 Introduction

Planifier à partir d'un scénario initial comportant des buts (ou propositions non expliquées) consiste à rendre *nécessairement cohérente* la base de connaissances temporelles associée. La notion de *cohérence* d'une telle base est définie dans ce chapitre indépendemment de la manière d'imposer cette cohérence dans la base; elle se décompose en deux critères décorrelés : un *critère de cohérence d'état* et un *critère de cohérence de ressource*. Nous montrons comment cette cohérence peut être vérifiée par des critères locaux. Dans le cas d'une base non nécessairement cohérente, ces critères locaux permettent de définir un ensemble de *défauts* de la base de connaissances qui constitue le plan partiel, défauts qu'il est suffisant de résoudre pour obtenir une base nécessairement cohérente. Ces défauts peuvent être résolus par l'insertion de *résolvantes* qui sont, soit des contraintes temporelles, soit des contraintes sur les variables, soit des ajouts de protections et/ou de nouvelles tâches au plan partiel.

IxTeT repose sur un algorithme global de planification de type SNLP [McAllester 91] enrichi du fait de notre représentation originale utilisant des attributs valués et de la gestion explicite du temps et des ressources. L'intégration des processus de synthèse de plans (gestion des attributs d'état) et de gestion des ressources s'effectue par une résolution en concomitance des défauts liés aux attributs d'état et de ceux liés aux conflits de ressources potentiels.

Dans IxTeT , la recherche d'une solution s'effectue dans un espace de plans partiels. Comme dans SNLP, le processus de planification consiste en une résolution incrémentale de tous les défauts du scénario initial dans un arbre de recherche où chaque nœud représente un défaut sélectionné et chaque branche le choix d'une résolvante pour le résoudre. Nous présentons dans ce chapitre une version simplifiée de l'algorithme de planification non-déterministe utilisé par IxTeT ainsi que la preuve de sa correction et de sa complétude.

Les choix non-déterministes sont effectués selon une stratégie de *moindre engagement* et le parcours de l'arbre de recherche est assuré par un algorithme de type A_ϵ avec une heuristique permettant d'obtenir en pratique des solutions simples et souples au problème initial.

3.2 Le critère de cohérence d'état

Dans ce paragraphe, nous considérons uniquement les attributs d'état. La notion de cohérence associée aux attributs de ressources, qui en est totalement décorrelée, sera examinée dans le paragraphe suivant.

De manière intuitive, la cohérence d'état permet d'assurer que, dans toutes les instanciations possibles de la base de connaissances, les attributs d'état instanciés ne prennent qu'une et une seule valeur à un moment donné et que les seuls changements du monde sont dûs à des événements. Comme nous le verrons plus loin, une des principales conditions pour qu'un plan soit solution d'un problème est que la base de connaissance associée soit cohérente au niveau des états.

3.2.1 Définition de la cohérence d'état

Soit $\mathcal{B} = \{EVT, HOLD, C_V, C_T\}$ une base de connaissances temporelle IxTeT ne comportant aucune proposition d'utilisation de ressources et pour laquelle nous supposerons que les réseaux de contraintes C_V et C_T sont consistants. Soit $\underline{\mathcal{B}} = \{EVT, HOLD, d_V, d_T\}$ une instance de $\mathcal{B}$.

Dans $\underline{\mathcal{B}}$, l'instanciation d_T des instants définit sur l'ensemble T des instants un ordre total que nous noterons $(t_0, t_1, t_2, ..., t_m, t_{m+1})$ (t_0 désignant l'instant initial de la base de connaissances et t_{m+1} l'instant final).

Pour un attribut d'état instancié $attS(X_1, ..., X_n)$ et un instant t_i :

on notera $EVT(attS(X_1, ..., X_n), t_i)$ l'ensemble des événements sur $attS(X_1, ..., X_n)$ à l'instant t_i:

$$EVT(attS(X_1, ..., X_n), t_i) = \{event(attS(?x_1, ..., ?x_n) : (?v_{ancienne}, ?v_{nouvelle}), t) \in EVT / \\ d_V(?x_1) = X_1, ..., d_V(?x_n) = X_n \; et \; t = t_i\} \; ;$$

on notera $HOLD^-(attS(X_1, ..., X_n), t_i)$ l'ensemble des assertions sur $attS(X_1, ..., X_n)$ imposant une valeur juste avant t_i :

$$HOLD^-(attS(X_1, ..., X_n), t_i) = \{hold(attS(?x_1, ..., ?x_n) :?v_{protégée}, (t_j, t_k)) \in HOLD / \\ d_V(?x_1) = X_1, ..., d_V(?x_n) = X_n \; et \; t_j < t_i \leq t_k\} \; ; \text{et}$$

on notera $HOLD^+(attS(X_1, ..., X_n), t_i)$ l'ensemble des assertions sur $attS(X_1, ..., X_n)$ imposant une valeur juste après t_i :

$$HOLD^+(attS(X_1, ..., X_n), t_i) = \{hold(attS(?x_1, ..., ?x_n) :?v_{protégée}, (t_j, t_k)) \in HOLD / \\ d_V(?x_1) = X_1, ..., d_V(?x_n) = X_n \; et \; t_j \leq t_i < t_k\};$$

Soit les ensembles de valeurs $V^-(attS(X_1, ..., X_n), t_i)$ et $V^+(attS(X_1, ..., X_n), t_i)$ représentant les valeurs de l'attribut instancié $attS(X_1, ..., X_n)$ respectivement juste avant et juste après l'instant t_i et définis par :

$$V^-(attS(X_1,...,X_n),t_i) = \bigcup_{h \in HOLD^-(attS(X_1,...,X_n),t_i)} val(h)$$

$$V^+(attS(X_1,...,X_n),t_i) = \bigcup_{h \in HOLD^+(attS(X_1,...,X_n),t_i)} val(h)$$

Où $val(h)$ désigne la valeur d'une assertion h.

De manière informelle, une instance $\underline{\mathcal{B}}$ sera dite cohérente pour les états si, pour chaque attribut d'état instancié, (1) les assertions imposent une et une seule valeur juste avant et juste après chaque instant ; (2) cette valeur ne change pas entre deux instants consécutifs ; et (3) sur cet attribut instancié chaque changement de valeur correspond à un unique événement.

Plus formellement, une instance $\underline{\mathcal{B}}$ est **cohérente pour les états** si et seulement si :

$\forall attS(X_1,...,X_n)$,

(1) $|V^+(attS(X_1,...,X_n),t_0)| = 1$;
$\forall i \in [1,m], |V^-(attS(X_1,...,X_n),t_i)| = |V^+(attS(X_1,...,X_n),t_i)| = 1$
$|V^-(attS(X_1,...,X_n),t_{m+1})| = 1$;

(2) $\forall i \in [0,m], V^+(attS(X_1,...,X_n),t_i) = V^-(attS(X_1,...,X_n),t_{i+1})$; et

(3) $\forall t_i : |EVT(attS(X_1,...,X_n),t_i)| \leq 1$, et
$V^-(attS(X_1,...,X_n),t_i) \neq V^+(attS(X_1,...,X_n),t_i) \Leftrightarrow$
$\exists! e \in EVT(attS(X_1,...,X_n),t_i)/$
$val^-(e) = V^-(attS(X_1,...,X_n),t_i)$ et
$val^+(e) = V^+(attS(X_1,...,X_n),t_i)$ et
$val^-(e) \neq val^+(e)$

La figure 3.1 illustre une instance incohérente et une instance cohérente.

Une base de connaissances IXTET sera dite **nécessairement cohérente pour les états** si et seulement si toutes ses instances sont cohérentes.

A noter que la définition que nous avons adoptée pour la cohérence nécessaire d'une base de connaissances IXTET est relativement forte du fait que nous imposons, sur chacune des instances, la protection à tout instant de la valeur des attributs. Ainsi la base représentée sur la figure 3.2 n'est pas, selon notre définition, nécessairement cohérente bien que chacune de ses instanciations puisse représenter une évolution correcte du monde.

Nous ne formaliserons pas ici cette notion de cohérence au sens large. L'utilisation de celle-ci pour la définition d'un plan solution ne remet toutefois pas en cause les résultats de correction et de complétude au sens restreint de notre système (cf. preuve de la complétude au sens restreint, Annexe D).

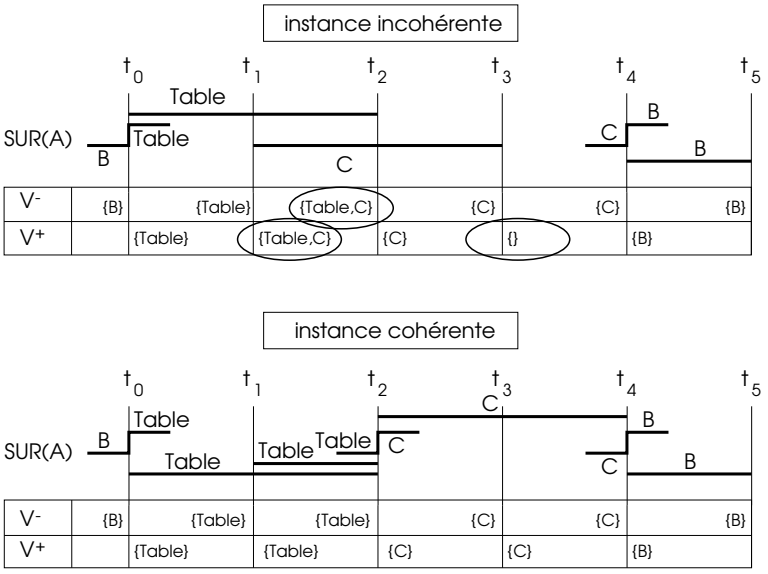


Figure 3.1: Instance incohérente et instance cohérente

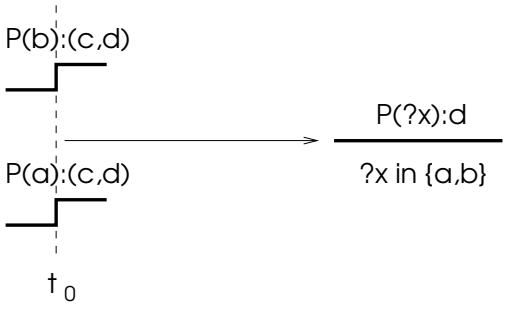


Figure 3.2: Une base nécessairement cohérente au sens large

3.2.2 Un critère de cohérence local

Dans ce paragraphe, nous définissons pour notre formalisme temporel un critère local de cohérence qui s'inspire du critère décrit dans [McAllester 91] . Nous montrons que si ce critère est vérifié sur toute la base de connaissances temporelles, alors, celle-ci est nécessairement cohérente. Auparavant, nous définissons dans notre formalisme les notions de *proposition non-expliquée* et de *menace sur une protection* dans le contexte d'une base de connaissances temporelles $\mathcal{B}$.

Définition 1 *On appelle* **proposition expliquée** *un événement* $event(attS(?x_1, ..., ?x_n) : (?v, *), t)$ *ou une assertion* $hold(attS(?x_1, ..., ?x_n) :?v, (t, *))$ *tel que*[1] *:*

1. *il existe un événement*[2] **établisseur** $event(attS(?x'_1..., ?x'_n) : (*, ?v_{est}), t_{est})$ *tel que nécessairement* $(t_{est} \leq t)$,$(?v_{est} = ?v)$,$(?x'_i = ?x_i)$ *; et*

[1] L'étoile, "*", représente ici un instant ou une variable n'intervenant pas dans la définition.

[2] Si notre formalisme est enrichi avec le prédicat $assert$, l'explication d'une proposition temporelle du plan par une proposition de type $assert(attS(?x'_1..., ?x'_n) :?v_{est}, t_{est})$ est alors envisageable.

2. *si* $(t_{est} < t)$, *il existe une assertion (appelée* **lien causal***) entre l'établisseur et la proposition expliquée :* $hold(attS(?x_1, ..., ?x_n) :?v, (t_{est}, t))$.

Définition 2 *On appelle* **menace** *sur une assertion* h_P *avec* $h_P = hold(attS(?x_1, ..., ?x_n) :?v_{protégée}, (t_{début}, t_{fin}))$*:*

- *un couple* $< e, h_P >$ *où* $e = event(attS(?x'_1, ..., ?x'_n) : (?v^-, ?v^+), t)$ *tel que aucune des contraintes* $\{(?x_i \neq ?x'_i), (t < t_{début}), (t = t_{début}$ *et* $?v^+ = ?v_{protégée}), (t = t_{fin}$ *et* $?v^- = ?v_{protégée}), (t_{fin} < t)\}$ *ne peut se déduire de la base* $\mathcal{B}$ *; ou*
- *un couple* $< h, h_P >$ *où* $h = hold(attS(?x'_1, ..., ?x'_n) :?v, (t^-, t^+))$ *tel que aucune des contraintes* $\{(?x_i \neq ?x'_i), (?x_1 = ?x'_1$ *et ... et* $?x_n = ?x'_n$ *et* $?v = ?v_{protégée}), (t^+ < t_{début}), (t_{fin} < t^-)\}$ *ne peut se déduire de la base* $\mathcal{B}$ *;*

Théorème 1 *Soit* $\mathcal{B}$ *une base de connaissances temporelles* $I_{X}T_{E}T$ *. Si toutes les propositions temporelles de* $\mathcal{B}$ *sont expliquées et si* $\mathcal{B}$ *ne contient aucune menace, alors,* $\mathcal{B}$ *est nécessairement cohérente au niveau des états.*

La preuve de ce théorème est donnée dans l'annexe D.

La réciproque n'est pas toujours vraie car la notion de "proposition expliquée" définie ci-dessus impose l'existence d'un lien causal commençant exactement à partir d'un événement et se terminant exactement au premier instant de la proposition à expliquer. Ce n'est, par exemple, pas le cas pour l'assertion $hold(SUR(A) : Table, (t_1, t_2))$ sur la figure 3.1 car il n'existe pas d'assertion $hold(SUR(A) : Table, (t_0, t_1))$. La complétude du système telle qu'elle est définie au théorème 4 (complétude au sens restreint), n'est toutefois nullement affectée par cette remarque comme nous le verrons.

La vérification de la cohérence nécessaire par application directe du théorème peut s'effectuer avec une complexité polynomiale en $O(n_e^2.n_h)$ si n_e désigne le nombre maximal d'événements et n_h le nombre maximal d'assertions liées à un attribut donné.

Dans le cadre de la planification, la complexité de cette vérification est réduite par le fait que la structure d'explication est complètement explicitée et maintenue à jour de manière incrémentale (vérifier le statut expliqué/non-expliqué d'une proposition s'effectue en temps constant) et donc, qu'il ne reste à vérifier que l'absence de menace, ce qui peut être fait en $O(n_h.(n_e + n_h))$.

3.3 Le critère de disponibilité de ressource

Nous considérons uniquement dans ce paragraphe les attributs de ressource et procédons de la même manière qu'au paragraphe précédent en définissant la notion de cohérence d'une base de connaissances temporelle au niveau des ressources et en définissant et prouvant un critère local permettant de la vérifier.

3.3.1 Définition de la cohérence de ressource

Soit $\mathcal{B} = \{USE, CONS, PROD, C_V, C_T\}$ une base de connaissances temporelles IxTeT pour laquelle nous supposerons que les réseaux de contraintes C_V et C_T sont consistants. soit $\underline{\mathcal{B}} = \{USE, CONS, PROD, d_V, d_T\}$ une instance de $\mathcal{B}$.

Nous reprenons les notations définies précédemment en considérant $(t_0, t_1, t_2, ..., t_m, t_{m+1})$ la liste chronologique des instants de T imposée par l'instanciation d_T.

Soit les quatre ensembles suivants :

$USE^-(attR(X_1, ..., X_n), t_i)$ désignant l'ensemble de tous les emprunts de la ressource $attR(X_1, ..., X_n)$ sur un intervalle temporel se terminant à t_i :

$$USE^-(attR(X_1, ..., X_n), t_i) = \{use(attR(?x_1, ..., ?x_n) : q, (t_j, t_i)) \in USE \mid d_V(?x_1) = X_1, ..., d_V(?x_n) = X_n\};$$

$USE^+(attR(X_1, ..., X_n), t_i)$ désignant l'ensemble de tous les emprunts de la ressource $attR(X_1, ..., X_n)$ sur un intervalle temporel débutant à t_i :

$$USE^+(attR(X_1, ..., X_n), t_i) = \{use(attR(?x_1, ..., ?x_n) : q, (t_i, t_j)) \in USE \mid d_V(?x_1) = X_1, ..., d_V(?x_n) = X_n\};$$

$CONS(attR(X_1, ..., X_n), t_i)$ l'ensemble de toutes les consommations de la ressource $attR(X_1, ..., X_n)$ ayant lieu à l'instant t_i :

$$CONS(attR(X_1, ..., X_n), t_i) = \{consume(attR(?x_1, ..., ?x_n) : q, t_i) \in CONS \mid d_V(?x_1) = X_1, ..., d_V(?x_n) = X_n\}; \text{ et}$$

$PROD(attR(X_1, ..., X_n), t_i)$ l'ensemble de toutes les productions de la ressource $attR(X_1, ..., X_n)$ ayant lieu à l'instant t_i :

$$PROD(attR(X_1, ..., X_n), t_i) = \{produce(attR(?x_1, ..., ?x_n) : q, t_i) \in PROD \mid d_V(?x_1) = X_1, ..., d_V(?x_n) = X_n\}.$$

Ces quatre ensembles sont respectivement notés $USE^-(t_i), USE^+(t_i), CONS(t_i)$ et $PROD(t_i)$ lorsqu'il n'y a pas d'ambiguïté.

Pour un attribut de ressource instancié $attR(X_1, ..., X_n)$, on notera $\Delta(attR(X_1, ..., X_n), t_i)$ la variation de la quantité disponible de cette ressource à l'instant t_i, que l'on peut définir par :

$$\Delta(attR(X_1, ..., X_n), t_i) =$$

$$\sum_{r \in USE^-(t_i)} q(r) + \sum_{p \in PROD(t_i)} q(p) - \sum_{r \in USE^+(t_i)} q(r) - \sum_{c \in CONS(t_i)} q(c)$$

Une instance sera dite cohérente pour les ressources si et seulement si, la quantité disponible pour chaque ressource reste positive au cours du temps sachant que la ressource est initialement disponible en une quantité $Q_0(attR(X_1, ..., X_n))$.

En d'autre termes $\underline{\mathcal{B}}$ est **cohérente pour les ressources** si et seulement si :

$$\forall attR(X_1, ..., X_n), \forall t_i$$
$$Q_0(attR(X_1, ..., X_n)) + \sum_{k=1}^{k=i} \Delta(attR(X_1, ..., X_n), t_k) \geq 0$$

Une base de connaissances IXTET sera dite **nécessairement cohérente pour les ressources** si et seulement si toutes ses instances sont cohérentes pour les ressources.

3.3.2 La représentation à base de "use"

Pour avoir une représentation plus homogène des ressources et de leur utilisation, et afin de simplifier les algorithmes, nous traduisons toutes les propositions de consommation (*consume*) et de production (*produce*) de ressources en emprunts (*use*).

Il s'agit donc de trouver une transformation Θ permettant de passer d'une base $\mathcal{B} = \{USE, CONS, PROD, C_V, C_T\}$ à une base $\Theta(\mathcal{B}) = \{USE', \emptyset, \emptyset, C_V, C_T\}$ strictement équivalente à $\mathcal{B}$ au niveau de la disponibilité des ressources.

Le problème est aisé pour les consommations de ressources : la consommation d'une certaine quantité de ressource à un instant t peut en effet être considérée comme l'emprunt de cette quantité sur la demi-droite temporelle $[t, +\infty[$. On a ainsi l'équivalence :

$$consume(attR(?x_1, ..., ?x_n) : q, t) \Leftrightarrow use(attR(?x_1, ..., ?x_n) : q, (t, +\infty))$$

Pour ce qui est de la production de ressource à un instant t, on peut remarquer qu'au niveau de la disponibilité de la ressource produite, on peut considérer que celle-ci a été produite, non à l'instant t mais à l'instant 0, origine des dates dans le passé, sachant qu'elle est demeurée indisponible jusqu'à t ; en d'autres termes:

$$produce(attR(?x_1, ..., ?x_n) : q, t) \Leftrightarrow produce(attR(?x_1, ..., ?x_n) : q, 0) \wedge use(attR(?x_1, ..., ?x_n) : q, (0, t))$$

La production à l'instant 0 est alors vue, plus simplement, sous la forme d'un accroissement de la quantité initiale de la ressource[3].

Il est clair que, pour une base $\mathcal{B}$ donnée, la transformation Θ, définie en remplaçant tous les prédicats *consume* et *produce* par des *use* selon les équivalences ci-dessus, donne une base $\Theta(\mathcal{B})$ équivalente au niveau du critère de disponibilité des ressources défini dans le paragraphe précédent. En effet, chaque consommation est simplement transférée de l'ensemble $CONS$ à l'ensemble USE^+ et chaque production de $PROD$ à USE^-; ce qui ne modifie pas la valeur du critère Δ au cours du temps.

[3]Dans notre représentation la quantité initiale de chacune des ressources doit être déclarée explicitement. Aussi avons nous fait l'hypothèse, pour simplifier l'intégration au plan de l'augmentation de cette quantité initiale suite à une production de ressource, que toute production s'effectue sur un attribut de ressource totalement instancié.

Un exemple de transformation Θ est illustré sur la figure 3.3. Il s'agit de la base de connaissances associée à la tâche $TRANSFERER\_BRIQUES(?robot1, Salle1, Salle2)$ déjà décrite au chapitre 2.

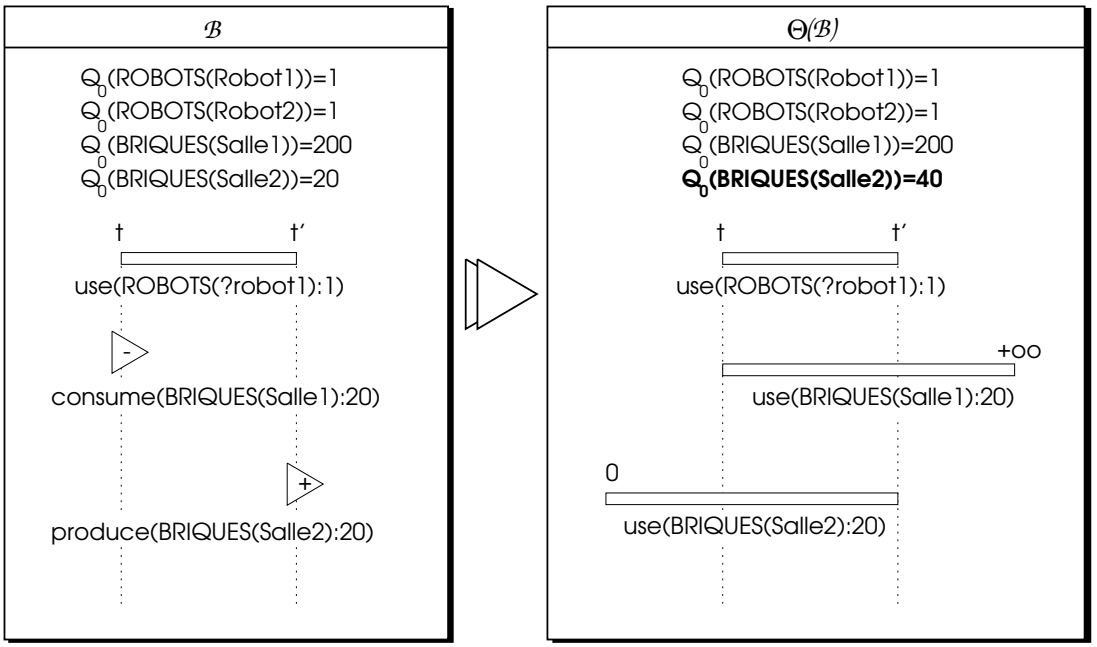


Figure 3.3: La transformation Θ

Dans la suite de ce mémoire nous supposerons, sauf mention contraire, que moyennant la transformation Θ, toutes nos bases de connaissances sont uniquement constituées de propositions d'emprunt de ressource (*use*).

3.3.3 Un critère de cohérence local

Comme au paragraphe §3.2.2, nous définissons ici un critère local pour la disponibilité des ressources et montrons que si ce critère est partout vérifié alors, la base de connaissances est nécessairement cohérente pour les ressources.

Le critère local pour vérifier la cohérence nécessaire d'une base au niveau des ressources repose sur la notion, déjà définie dans [Erschler 90] pour les ressources de type cumulatif, d'*ensemble critique minimal* (ECM).

De manière informelle, un *ensemble critique* est un conflit de ressource potentiel qui pourrait apparaître dans certaines instances de la base de données temporelles ; c'est-à-dire un ensemble d'emprunts de ressources, (1) dont les intervalles temporels pourraient globalement se chevaucher dans le temps et (2) qui pourraient porter sur une même ressource dont la quantité initiale ne permettrait pas de les supporter simultanément. Un ECM est un ensemble critique *minimal* au sens de l'inclusion ensembliste.

Nous donnons ci-dessous une définition plus formelle d'un ensemble critique minimal.

Définition 3 *On appelle* **ensemble critique minimal** *tout sous-ensemble* $U = \{use(attR(?x_{k1}, ..., ?x_{kn}) : q_k, (t_k^-, t_k^+))\}_{k\in[1,m]} \subset USE$ *vérifiant les trois conditions suivantes :*

1. *il existe une ressource* $attR(X_1, ..., X_n)$ *telle que* C_V *n'est pas incompatible avec la contrainte* $\bigwedge_{j\in[1,n]}(?x_{1j} = ... = ?x_{kj} = ... = ?x_{mj} = X_j)$ *et telle que* $\sum_{k=1}^{k=m} q_k > Q_0(attR(X_1, ..., X_n))$.
   Nous noterons $RES(U)$ *l'ensemble des ressources* $attR(X_1, ..., X_n)$ *vérifiant ce premier point (ressources possiblement conflictuelles pour* U*) ;*
2. C_T *n'est pas incompatible avec la contrainte* $\bigwedge_{k,l\in[1,m]}(t_k^- < t_l^+)$ *;*
3. $\forall U' \subset U, U' \neq U$, *U' n'est pas un ECM.*

Un exemple d'ECM est représenté sur la figure 3.4.

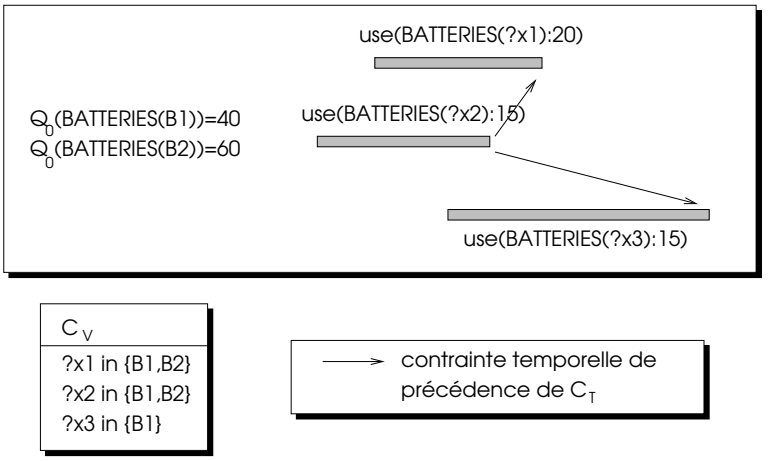


Figure 3.4: Un ensemble critique minimal

On peut alors montrer le théorème suivant (voir l'annexe D pour sa preuve) :

Théorème 2 *Soit* $\mathcal{B}$ *une base de connaissances temporelles* $I_X T_E T$ *.*$\mathcal{B}$ *est nécessairement cohérente au niveau des ressources si et seulement si* $\mathcal{B}$ *ne contient aucun ensemble critique minimal.*

Nous reviendrons plus en détail dans le chapitre 4 sur la complexité de la vérification de ce critère. Mais notons d'ores et déjà que, du fait que les ECM font intervenir un nombre a priori indéterminé de propositions d'emprunt de ressources, leur détection ne peut se résumer à un simple parcours de sous-ensembles de USE de taille fixe (comme cela est fait pour vérifier l'absence de menace en parcourant un ensemble de couples avec une complexité quadratique). Parcourir l'ensemble des sous-ensembles de USE conduirait à un algorithme en $O(2^{|USE|})$ inutilisable en pratique.

3.4 Problème de planification

Nous dirons qu'une base de connaissances temporelles $\mathcal{B}$ est **consistante** ssi:

1. C_T est consistant ;
2. C_V est consistant ;
3. $\mathcal{B}$ est nécessairement cohérente au niveau des états ; et
4. $\mathcal{B}$ est nécessairement cohérente au niveau des ressources.

Un **problème de planification** est un couple $< \mathcal{P}_0, \{\mathcal{T}_1, ..., \mathcal{T}_m\} >$ où $\mathcal{P}_0$ est un scénario initial et $\{\mathcal{T}_1, ..., \mathcal{T}_m\}$ un ensemble de bases de connaissances temporelles représentant des modèles de tâches.

Un **plan solution** est une base de connaissances $\mathcal{P}_{sol}$ telle que :

1. $\mathcal{P}_{sol}$ est consistant ;
2. $\mathcal{P}_{sol}$ supporte $\mathcal{P}_0$ ($\mathcal{P}_0 \leftarrow \mathcal{P}_{sol}$, cf. §2.6.1) ; et
3. tout changement du monde (événement ou utilisation de ressource) de $\mathcal{P}_{sol}$ non-prévu dans le scénario initial $\mathcal{P}_0$ appartient à une tâche $\mathcal{T}_i$ insérée entièrement dans le plan lors de sa génération.

3.5 L'algorithme global : défauts et résolvantes

Dans sa recherche d'un plan solution IxTeT explore un arbre de recherche dont la racine est le *scénario initial* $\mathcal{P}_0$ et les nœuds des *plans partiels* $\mathcal{P}_i$. Le passage d'un nœud $\mathcal{P}_i$ à un nœud suivant $\mathcal{P}_{i+1}$ s'effectue en résolvant un *défaut* de $\mathcal{P}_i$ par l'insertion d'une *résolvante* de ce défaut.

Soit $\mathcal{P}$ une base de connaissances IxTeT représentant un plan partiel.

Définition 4 *Un défaut Φ de $\mathcal{P}$ est soit une proposition d'état non-expliquée (NEX), soit une menace sur une protection (MNC), soit un ensemble critique minimal (ECM). Nous noterons $DEFAUTS(\mathcal{P}) = NEX(\mathcal{P}) \cup MNC(\mathcal{P}) \cup ECM(\mathcal{P})$.*

Pour résoudre un défaut Φ, quatre types de **résolvantes** ρ vont pouvoir être insérées sur le plan partiel ; ces quatre types de résolvantes sont classiques dans la littérature [Chapman 87, McAllester 91] :

- des contraintes de précédence temporelle ($t_i < t_j$) ;

- des contraintes sur les variables ($?x_i = ?x_j, ?x_i \neq ?x_j$) ;
- des liens causaux entre un établisseur déjà dans le plan partiel et une proposition nécessitant d'être expliquée ($hold(attS(?x_1, ..., ?x_n) : ?v_{protégée}, (t_{est}, t_{need}))$) ; et
- l'insertion d'un nouvel opérateur pour utiliser un de ses effets (changement d'état ou production de ressource).

De manière générale, une résolvante ρ est une conjonction d'éléments (contraintes, assertions ou tâches à insérer) que nous noterons $\rho = [\alpha_1, ..., \alpha_r]$.

Nous définissons l'opérateur $\oplus$ d'insertion d'un élément de résolvante α dans un plan partiel $\mathcal{P} = (EVT, HLD, USE, CONS, PROD, C_V, C_T)$, soit $\mathcal{P} \oplus \alpha$ de la manière suivante :

- $\mathcal{P} \oplus (t < t') = (EVT, HLD, USE, CONS, PROD, C_V, C_T \cup \{t < t'\})$
- si c(?x,?x') désigne une contrainte d'égalité ou d'inégalité entre variables :
  $\mathcal{P} \oplus (c(?x, ?x')) = (EVT, HLD, USE, CONS, PROD, C_V \cup \{c(?x, ?x')\}, C_T)$
- $\mathcal{P} \oplus (hold(attS(?x_1, ..., ?x_n) : ?s, (t^-, t^+))) =$
  $(EVT, HLD \cup \{hold(attS(?x_1, ..., ?x_n) : ?s, (t^-, t^+))\}, USE, CONS, PROD,$
  $C_V, C_T \cup \{t_{start} < t^- < t^+ < t_{end}\})$
- si $\mathcal{T} = (\overline{EVT}, \overline{HLD}, \overline{USE}, \overline{CONS}, \overline{PROD}, \overline{C}_V, \overline{C}_T)$ représente une tâche, $\mathcal{P} \oplus \mathcal{T} =$
  $(EVT \cup \overline{EVT}, HLD \cup \overline{HLD}, USE \cup \overline{USE}, CONS \cup \overline{CONS},$
  $PROD \cup \overline{PROD}, C_V \cup \overline{C}_V, C_T \cup \overline{C}_T \cup \{\forall t \in \overline{T}, t_{start} < t < t_{end}\})$
  A noter que toutes les variables de $\overline{V}$ et les instants de $\overline{T}$ sont renommés de manière à ce qu'ils n'interfèrent avec aucun nom de variable ou d'instant de $\mathcal{P}$.

L'insertion d'une résolvante $\rho = [\alpha_1, ..., \alpha_r]$ sur un plan partiel $\mathcal{P}$ produit alors un nouveau plan $\mathcal{P} \oplus \rho = \mathcal{P} \oplus \alpha_1 \oplus ... \oplus \alpha_r$[4]

Dans les trois paragraphes qui suivent, nous détaillons les résolvantes associées à chacun des trois types de défauts d'un plan partiel.

3.5.1 Résolvantes d'une proposition non-expliquée

Soit Φ une proposition non-expliquée constitué par un événement $event(attS(?x_1, ..., ?x_n) : (?v, *), t)$ ou une assertion $hold(attS(?x_1, ..., ?x_n) : ?v, (t, *))$ non-expliqué d'un plan partiel.

L'ensemble disjonctif $résolvantes(\Phi)$ des résolvantes de ce défaut est l'ensemble des liens causaux (assertions) entre tous les événements[5] (déjà contenus dans $\mathcal{P}$ ou issus d'un modèle de tâche $\mathcal{T}_i$) susceptible d'établir la proposition non-expliquée et cette dernière.

[4] A noter que l'opérateur $\oplus$ ne vérifie pas nécessairement $(\mathcal{P} \oplus \alpha) \oplus \alpha' = (\mathcal{P} \oplus \alpha') \oplus \alpha$ puisque quelquefois, α' peut nécessiter des éléments de $\mathcal{P} \oplus \alpha$ qui sont absents de $\mathcal{P}$, auquel cas $\mathcal{P} \oplus \alpha'$ n'a même pas de sens. $\mathcal{P} \oplus \alpha_1 \oplus ... \oplus \alpha_r$ est simplement une notation pour $((\mathcal{P} \oplus \alpha_1) \oplus ...) \oplus \alpha_r$. L'ordre des éléments de la conjonction ρ est donc important.

[5] Si le prédicat *assert* est géré, il faut aussi examiner les *assert* dans le plan courant et dans les modèles de tâches comme établisseurs possibles.

Plus formellement, nous dirons qu'un événement $event(attS(?x'_1...,?x'_n) : (*,?v_{est}), t_{est})$ est un établisseur possible d'une proposition $event(attS(?x_1,...,?x_n) : (?v,*),t)$ ou $hold(attS(?x_1,...,?x_n) :?v,(t,*))$ sur un plan partiel $\mathcal{P}$ si les contraintes $t_{est} < t$,$?x'_i =?x_i$ et $?v_{est} =?v$ sont compatibles avec les contraintes déjà posées sur $\mathcal{P}$.

L'ensemble des résolvantes est alors :

$$résolvantes(\Phi) = \bigcup_{e \in EVT(\mathcal{P}) |\ e\ établisseur\ possible\ de\ \Phi} \{\rho_{NEX,plan}(\Phi,e)\}$$

$$\bigcup_{\mathcal{T}} \bigcup_{e \in EVT(\mathcal{T}) |\ e\ établisseur\ possible\ de\ \Phi} \{\rho_{NEX,nouvelle\_tâche}(\Phi,\mathcal{T},e)\}$$

avec :

$$\rho_{NEX,plan}(\Phi,e) = [\ (t_{est} < t),\ (?x'_1 =?x_1),\ ...,\ (?x'_n =?x_n),\ (?v_{est} =?v),\ hold(attS(?x_1,...,?x_n) :?v,(t_{est},t)\ ]$$

$$\rho_{NEX,nouvelle\_tâche}(\Phi,\mathcal{T},e) = [\ \mathcal{T},\ (t_{est} < t),\ (?x'_1 =?x_1),\ ...,\ (?x'_n =?x_n),\ (?v_{est} =?v),\ hold(attS(?x_1,...,?x_n) :?v,(t_{est},t)\ ]$$

Le schéma 1 résume les différentes résolvantes pour une proposition d'état non-expliquée. A noter qu'avec notre formalisme, une même tâche peut éventuellement être utilisée de différentes manières pour établir une proposition à partir du moment où elle possède plusieurs événements susceptibles d'être des établisseurs de cette proposition.

Nous détaillerons davantage au paragraphe 3.7 le module d'IxTeT chargé de l'analyse des propositions non-expliquées et du calcul des résolvantes associées.

3.5.2 Résolvantes d'une menace

Soit Φ une menace ($\Phi \in MNC(\mathcal{P})$). Les résolvantes associées sont exactement les contraintes qui interviennent dans sa définition, à savoir :

- si $\Phi =< e,h_P >$ avec $e = event(attS(?x'_1,...,?x'_n) : (?v^-,?v^+),t)$ et
  $h_P = hold(attS(?x_1,...,?x_n) :?v_{protégée},(t_{début},t_{fin}))$:
  $résolvantes(\Phi) = \{[(?x'_1 \neq ?x_1)],...,[(?x'_n \neq ?x_n)],[(t < t_{début})],[(t = t_{début}),$
  $(?v^+ =?v_{protégée})],[(t = t_{fin}),(?v^- =?v_{protégée})],[(t_{fin} < t)]\}$
  en ne conservant dans cette disjonction que les contraintes qui sont compatibles avec celles de C_V et C_T du plan partiel $\mathcal{P}$.

- si $\Phi =< h,h_P >$ avec $h = hold(attS(?x'_1,...,?x'_n) :?v,(t^-,t^+))$ et
  $h_P = hold(attS(?x_1,...,?x_n) :?v_{protégée},(t_{début},t_{fin}))$:
  $résolvantes(\Phi) = \{[(?x'_1 \neq ?x_1)],...,[(?x'_n \neq ?x_n)],[(t^+ < t_{début})],[(t = t_{début}),$
  $(?v^+ =?v_{protégée})],[(?x_1 =?x'_1),...,(?x_n =?x'_n),(?v =?v_{protégée})],$
  $[(t_{fin} < t^-)]\}$

en ne conservant dans cette disjonction que les contraintes qui sont compatibles avec celles de C_V et C_T du plan partiel $\mathcal{P}$[6].

Le schéma donne les résolvantes pour une menace de type $< e, h_P >$.

3.5.3 Résolvantes d'un ensemble critique minimal

Soit $\Phi = \{r_1, ..., r_k\}$ un ensemble critique minimal sur un type de ressource $attR$ ($\Phi \in ECM(\mathcal{P})$).

Pour résoudre Φ, il suffit :

- ou bien de retirer de Φ une quelconque des r_i par ajout d'une contrainte de précédence temporelle (ordonnancement) ou d'une contrainte sur les variables (contrainte d'allocation de ressource). En effet, dans ce cas, à cause de la minimalité des ECM au sens de l'inclusion ensembliste, $\Phi \backslash \{r_i\}$ ne sera plus un ECM ;
- ou bien de produire certaines des ressources intervenant dans Φ de manière à accroître leur quantité disponible Q_0.

Nous n'examinerons pas ici dans le détail comment sont calculées les résolvantes associées à un ECM, ceci sera fait dans le chapitre 4 dédié à la gestion des ressources.

On peut néammoins voir sur le schéma 3 et ceci, pour le moment, de manière intuitive, l'ensemble des résolvantes d'un ECM en supposant que l'on dispose d'une tâche capable de recharger n'importe quelle ressource de type $attR$ d'une quantité 10. On notera comme première constatation que la prise en compte de ressources complexes (ici, une ressource partageable, individualisée et productible) entraîne un facteur de branchement élevé dans l'arbre de recherche global (l'ECM de la figure fait apparaître 10 résolvantes). Ceci explique pourquoi il sera utile de minimiser, quand cela est possible, la redondance qui peut parfois apparaître entre certaines résolvantes d'un même ECM ; nous approfondirons cette question dans le chapitre 4.

[6] A noter que si le prédicat *assert* est géré, il faudrait aussi considérer les menaces possibles entre un *assert* et un *hold*. Nous laissons au lecteur le soin de déterminer l'ensemble des résolvantes associées à ce type de menace ce qui ne représente aucune difficulté particulière.

	Proposition non-expliquée (§3.5.1)	**Menace (§3.5.2)**	**ECM (§3.5.3)**
contrainte temporelle	précédence liée à la causalité	promotion demotion	ordonnancement
contrainte sur les variables	unification des attributs	séparation fusion (<h,h>)	contrainte d'allocation
assertion	protection de la causalité		
insertion de tâche	tâche d'établissement		tâche de production

Tableau 3.1: Défauts et résolvantes associées

3.5.4 Bilan

Pour chaque type de défaut, les contraintes liées aux différentes résolvantes sont résumées dans le tableau ci-dessous.

Remarquons que, bien que leur *origine* diffère (proposition non-expliquée, menace ou ensemble critique minimal), les résolvantes sont toutes de *nature* similaire : insertion de contraintes temporelles, de contraintes sur les variables atemporelles, de nouvelle assertion et/ou de nouvelle tâche. Cette remarque est essentielle à notre approche. En effet, comme nous allons le voir dans la suite de ce chapitre, le module de contrôle du système de planification ne va pas effectuer les choix non-déterministes en fonction de l'origine des défauts et résolvantes, mais uniquement en fonction de leur nature et de la manière dont ces résolvantes vont surcontraindre le plan.

3.5.5 L'algorithme non-déterministe de contrôle

L'algorithme de recherche global implémenté dans IxTeT est présenté ci-dessous. Un nœud de l'arbre de recherche correspond à la résolution d'un défaut Φ du plan partiel courant. Autant de nœuds fils sont créés que le défaut Φ possède de résolvantes. La planification s'effectue en appelant $planifier(\mathcal{P}_0)$ où $\mathcal{P}_0$ est le scénario initial.

Algorithme $PLANIFIER(\mathcal{P})$
1. **Terminaison :** si $\mathcal{P}$ est consistant, retourner $\mathcal{P}$
2. **Analyse :** calculer
   $DEFAUTS(\mathcal{P}) = NEX(\mathcal{P}) \cup MNC(\mathcal{P}) \cup ECM(\mathcal{P})$
3. **Sélection d'un défaut :** $\Phi \in DEFAUTS(\mathcal{P})$
4. **Choix d'une résolvante :**
   4.1. Si $résolvantes(\Phi) = \emptyset$ retourner *échec*
   4.2. Sinon choisir $\rho \in résolvantes(\Phi)$
5. **Appel récursif :** retourner $PLANIFIER(\mathcal{P} \oplus \rho)$

Suivant la terminologie définie dans [Weld 95] nous employons le mot *choix* pour désigner un choix non-déterministe et par conséquent un point possible de retour-arrière par opposition à *sélection* qui correspond à un choix heuristique sur lequel un retour-arrière ne sera pas nécessaire.

La preuve des deux théorèmes fondamentaux suivants est donnée dans l'annexe D :

Théorème 3 *L'algorithme de recherche d'*I_XT_ET *est sain : tout plan retourné est un plan solution.*

Théorème 4 *L'algorithme de recherche d'*I_XT_ET *est complet au sens restreint : s'il existe une solution au problème de planification, l'algorithme s'arrêtera au bout d'un temps fini et retournera un plan solution.*

Ce dernier théorème reste valable si on utilise la définition au sens large de la cohérence nécessaire d'une base de connaissances IXTET .

La figure 3.5 présente l'architecture globale du système de planification IXTET .

Un module spécifique d'analyse est associé à chaque type de défauts et est chargé de détecter l'ensemble ou une partie des défauts de ce type et, pour chacun des défauts détectés, de calculer l'ensemble complet de ses résolvantes ainsi que la valeur de l'estimation qui permettra d'effectuer les choix non-déterministes dans l'algorithme global. Sur la base de cet ensemble de défauts et résolvantes associées, le module de contrôle sélectionne le défaut qu'il estime le plus *opportuniste* de résoudre dans le contexte courant et lui choisit une résolvante selon une stratégie de *moindre engagement*. L'insertion et la propagation de la résolvante dans le plan partiel est assurée par les gestionnaires de contraintes sur les variables temporelles et atemporelles.

A noter que dans notre implémentation, le plan partiel n'est pas totalement réanalysé à chaque nœud de l'arbre de recherche : un agenda de défauts $\mathcal{A}$ représentant un sous-ensemble des défauts du plan partiel est incrémentalement remis à jour à chaque nœud. Lorsque nous estimons que les défauts de $\mathcal{A}$ ne sont plus assez opportunistes, une analyse globale est effectuée en utilisant les modules d'analyse pour réalimenter l'agenda $\mathcal{A}$.

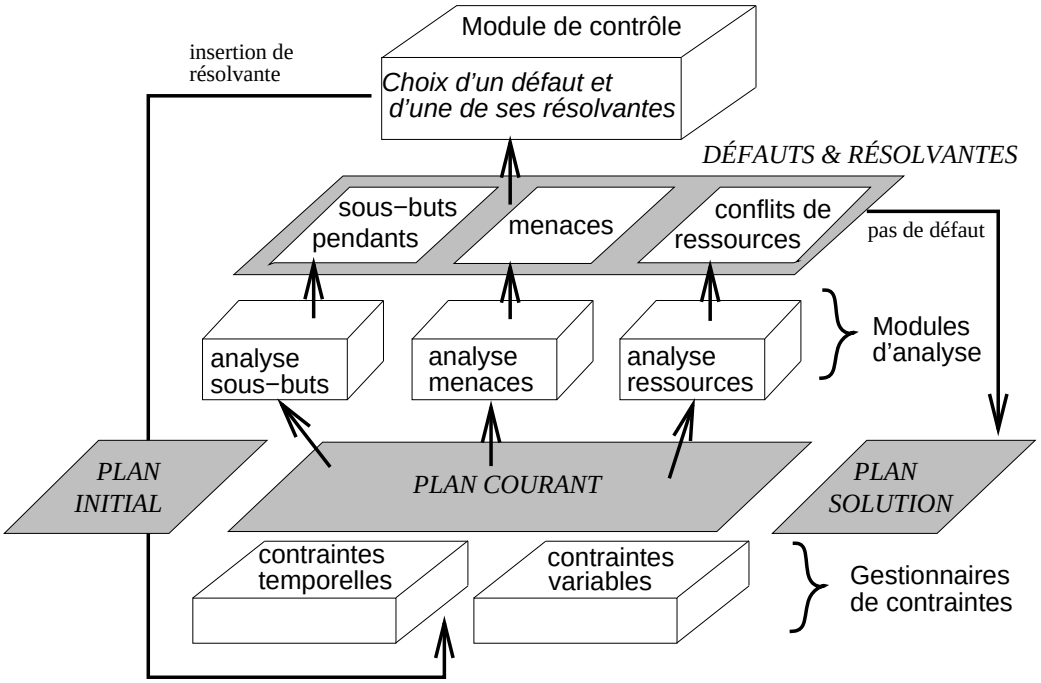


Figure 3.5: Architecture globale du système de planification

3.6 Les choix non-déterministes

Deux types de choix essentiels quant aux performances de la recherche interviennent dans l'algorithme : la sélection d'un défaut et le choix d'une résolvante pour ce défaut. Dans le but de réduire les risques de retour-arrière, le choix d'une résolvante s'effectue grâce à une stratégie de moindre engagement et la sélection d'un défaut par une stratégie opportuniste relativement à ce moindre engagement. Les paragraphes suivants précisent ces notions de *moindre engagement* et d'*opportunisme* de la recherche.

3.6.1 La stratégie de moindre engagement

Planifier consiste à choisir un chemin entre le scénario initial et un plan solution dans un graphe de recherche global $\mathcal{G}_{planif}$ dont les nœuds représentent l'ensemble des plans partiels et les arcs les résolvantes insérables sur ces plans.

Etant donné un plan partiel $\mathcal{P}$ dans $\mathcal{G}_{planif}$, nous noterons $COMPL(\mathcal{P})$ l'ensemble de tous les plans $\mathcal{P}'$ tels qu'il existe sur $\mathcal{G}_{planif}$ un chemin de résolvantes $(\rho_1, ..., \rho_n)$ entre $\mathcal{P}$ et $\mathcal{P}'$. $COMPL(\mathcal{P})$ représente donc toutes les manières de compléter $\mathcal{P}$ par des insertions de résolvantes.

Les plans partiels complétement instanciés atteignables à partir de $\mathcal{P}$ sont donc $\bigcup_{\mathcal{P}' \in COMPL(\mathcal{P})} INST(\mathcal{P}')$, ensemble en général infini non-dénombrable[7] que nous noterons

[7] Dans le cas d'IxTeT, du fait de la représentation du temps par des valeurs discrètes, on peut montrer que cet ensemble est dénombrable.

$INSTCOMPL(\mathcal{P})$.

Si $\mathcal{P}_0$ représente notre scénario initial, il est clair que tous les plans solutions de $\mathcal{P}_0$ totalement instanciés appartiennent à $INSTCOMPL(\mathcal{P}_0)$[8]. Lorsque l'on va insérer une résolvante ρ_1 sur $\mathcal{P}_0$ pour résoudre un de ses défauts Φ_1, on va réduire l'ensemble des plans totalement instanciés à $INSTCOMPL(\mathcal{P}_0 \oplus \rho_1)$ et ce faisant, éliminer de fait, toutes les solutions du problème contenues dans

$$INSTCOMPL(\mathcal{P}_0) \setminus INSTCOMPL(\mathcal{P}_0 \oplus \rho_1)$$

L'idée du moindre engagement implémentée dans IxTeT consiste à choisir, parmi les résolvantes possibles du défaut Φ_1 sélectionné, la résolvante ρ_1 qui va éliminer le moins de solutions possibles. Pour ce faire, nous faisons l'hypothèse que les solutions totalement instanciées sont uniformément réparties sur $INSTCOMPL(\mathcal{P}_0)$ et donc, que si l'on possède une mesure $\mu(INSTCOMPL(\mathcal{P}))$ de la taille de l'ensemble $INSTCOMPL(\mathcal{P})$, l'engagement lié à l'insertion de la première résolvante ρ_1 peut être estimé par :

$$engagt(\mathcal{P}_0, \rho_1) = 1 - \frac{\mu(INSTCOMPL(\mathcal{P}_0 \oplus \rho_1))}{\mu(INSTCOMPL(\mathcal{P}_0))}$$

Cette idée exposée ici pour le choix d'une résolvante au premier défaut sélectionné vaut évidemment pour l'estimation de l'engagement d'une résolvante quelconque sur un plan partiel quelconque :

$$engagt(\mathcal{P}, \rho) = 1 - \frac{\mu(INSTCOMPL(\mathcal{P} \oplus \rho))}{\mu(INSTCOMPL(\mathcal{P}))}$$

A noter qu'avec cette définition, $engagt(\mathcal{P}, \rho) \in [0, 1]$.

$engagt(\mathcal{P}, \rho) = 0$ si et seulement si la résolvante est redondante avec les contraintes du plan partiel ;

$engagt(\mathcal{P}, \rho) = 1$ si et seulement si les contraintes apportées par la résolvante sont incohérentes avec celles contenues dans le plan partiel[9].

La figure 3.6 illustre la stratégie de moindre engagement. Pour un plan partiel $\mathcal{P}_i$, l'ensemble $INSTCOMPL(\mathcal{P}_i)$ est représenté avec une répartition quasi-uniforme des plans solutions instanciés. Les plans partiels $\mathcal{P}_{i+1}^j$ sont le résultat de l'insertion sur $\mathcal{P}_i$ d'une résolvante ρ_{i+1}^j pour résoudre un défaut sélectionné. Du fait de la complétude de l'ensemble des résolvantes d'un défaut donné, les ensembles $INSTCOMPL(\mathcal{P}_{i+1}^j)$ recouvrent l'ensemble des plans solutions instanciés de $\mathcal{P}_i$. Ces ensembles ne sont par contre pas disjoints dans la mesure où

[8] Pour s'en convaincre, voir la preuve du théorème 4 (complétude au sens restreint) en annexe D.

[9] En fait, en pratique, ces deux cas ne se rencontrent pas pour nos résolvantes puisque d'après leur définition, nous nous assurons qu'elles sont utiles (puisque le défaut existe, $engagt(\mathcal{P}, \rho) \neq 0$) et qu'elles sont insérables sur le plan partiel ($engagt(\mathcal{P}, \rho) \neq 1$).

la propriété de systématicité n'est pas assurée [10]. Les résolvantes choisies en priorité sont celles qui maximisent la taille de $INSTCOMPL(\mathcal{P}^j_{i+1})$.

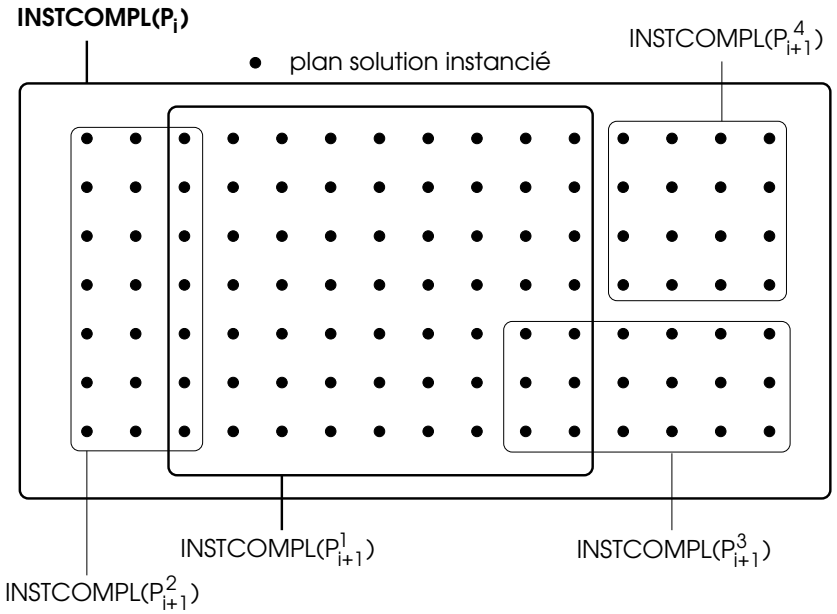


Figure 3.6: La stratégie de moindre engagement

De plus, si ρ et ρ' sont deux résolvantes (ou deux éléments de résolvante) alors:

$$engagt(\mathcal{P}, [\rho, \rho']) = \frac{\mu(INSTCOMPL(\mathcal{P})) - \mu(INSTCOMPL(\mathcal{P} \oplus \rho \oplus \rho'))}{\mu(INSTCOMPL(\mathcal{P}))} \quad (1)$$

Si ρ et ρ' sont des résolvantes de types différents, il est justifiable de faire l'hypothèse que le taux d'instances éliminées par ρ' est le même que ce soit à partir de $\mathcal{P}$ ou à partir de $\mathcal{P} \oplus \rho$. Ce qui revient à dire que $engagt(\mathcal{P} \oplus \rho, \rho') = engagt(\mathcal{P}, \rho')$ (2). De (1) et (2), on déduit aisément que

$$1 - engagt(\mathcal{P}, [\rho, \rho']) = [1 - engagt(\mathcal{P}, \rho)].[1 - engagt(\mathcal{P}, \rho')]$$

L'estimation de l'engagement $engagt(\mathcal{P}, \rho)$ s'effectue selon des critères locaux en fonction du type de résolvante ρ. Nous précisons ci-dessous le calcul de cette estimation dans le cas où la résolvante représente une contrainte de précédence temporelle, une contrainte d'égalité ou de différence entre variables et dans le cas d'un lien causal. Le cas plus difficile de l'insertion d'une nouvelle tâche au plan partiel sera examiné plus tard lors de la description du module d'analyse des sous-buts (§3.7). Insistons sur le fait qu'il ne s'agit que d'*estimations* effectuées en faisant un certain nombre d'hypothèses très fortes et basées davantage sur l'intuition que

[10] Rappelons qu'une recherche dans un graphe est dite "systématique" si tous les nœuds développés au cours de la recherche sont différents deux à deux. Une définition un peu plus contraignante des résolvantes associées à un défaut donné pourrait aisément conférer à notre algorithme cette propriété de systématicité. L'intérêt de cette propriété vis-à-vis de l'efficacité d'un algorithme de planification est toutefois largement discuté (voir [Kambhampati 93]).

sur une solide analyse probabiliste. L'objectif étant d'estimer l'engagement $engagt(\mathcal{P}, \rho)$ de chacune des résolvantes d'un défaut Φ afin d'insérer en priorité celle qui minimise cet engagement.

3.6.1.1 Engagement lié à une contrainte de précédence

Soit un plan partiel $\mathcal{P} = (EVT, HLD, USE, CONS, PROD, C_T, C_V)$ et deux instants t et t' de $\mathcal{P}$. Supposons que sur C_T, la propagation des contraintes temporelles ait réduit le domaine de $t' - t$ à l'intervalle $[d_{min}, d_{max}]$.

Pour calculer l'engagement lié à l'insertion sur $\mathcal{P}$ de la résolvante $\rho = [(t < t')]$, nous faisons l'hypothèse que sur l'ensemble des plans instanciés $INSTCOMPL(\mathcal{P})$, la variable $(t' - t)$ est uniformément répartie sur l'intervalle $[d_{min}, d_{max}]$. Comme l'insertion de ρ va éliminer l'ensemble des instances telles que $(t' - t) < 0$, nous estimons :

$$engagt(\mathcal{P}, [(t' < t)]) = \frac{min(d_{max}, 0) - min(d_{min}, 0)}{d_{max} - d_{min}}$$

Cette formule regroupe les trois cas présentés sur la figure 3.7. A noter que comme le réseau de contraintes temporelles explicite tous les intervalles $[d_{min}, d_{max}]$ pour chaque couple (t, t'), le calcul de $engagt(\mathcal{P}, [(t' < t)])$ peut être effectué en $O(1)$.

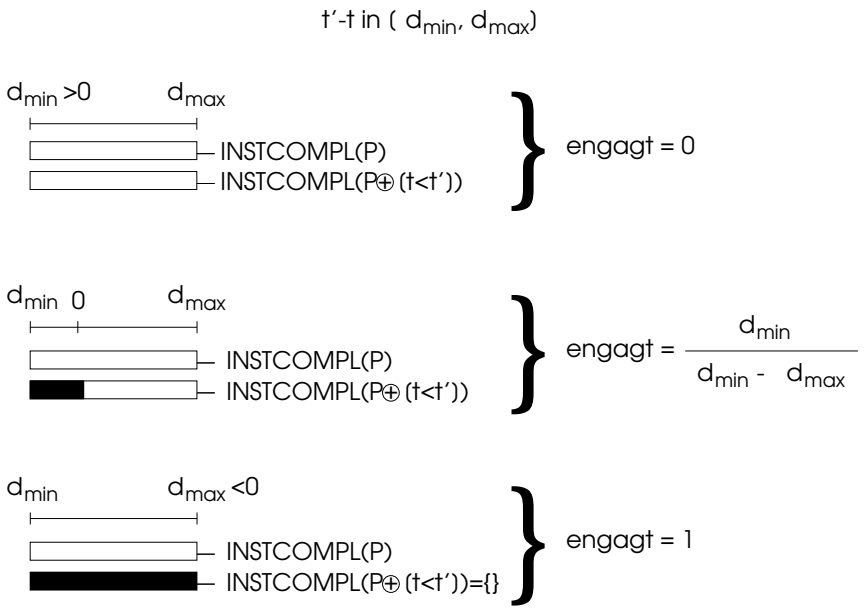


Figure 3.7: Engagement lié à une contrainte de précédence

Par exemple, si nous prenons sur la figure 2.5 le cas de l'engagement lié à l'insertion d'une résolvante $\rho = [(t_2 < t_4)]$, comme après propagation on a, sur le réseau de contraintes temporelles, $(t_4 - t_2) \in [-1, 3]$, on estime: $engagt(\mathcal{P}, [(t_2 < t_4)]) = (0 + 1)/(3 + 1) = 0.25$.

3.6.1.2 Engagement lié à une contrainte de différence

Soit un plan partiel $\mathcal{P} = (EVT, HLD, USE, CONS, PROD, C_T, C_V)$ et deux variables atemporelles $?x$ et $?y$ de C_V. Supposons que sur C_V, la propagation des contraintes ait réduit les domaines des variables $?x$ et $?y$ respectivement à $D_{?x}$ et $D_{?y}$.

L'engagement lié à l'insertion sur $\mathcal{P}$ de la résolvante $\rho = [(?x \neq ?y)]$ est estimé en supposant, de manière similaire au cas d'une contrainte de précédence, que sur $INSTCOMPL(\mathcal{P})$, les valeurs des variables $?x$ et $?y$ se répartissent uniformément sur $D_{?x}$ et $D_{?y}$. Dans ce cas là, les instances éliminées par l'insertion de la contrainte supplémentaire $?x \neq ?y$ sont, parmi le produit cartésien $D_{?x} \times D_{?y}$ tous les couples de valeurs $(?x, ?y)$ tels que $?x = ?y$, ensemble de taille $card(D_{?x} \cap D_{?y})$.

D'où l'estimation :

$$engagt(\mathcal{P}, [(?x \neq ?y)]) = \frac{card(D_{?x} \cap D_{?y})}{card(D_{?x}).card(D_{?y})}$$

Comme sur le réseau C_V des contraintes sur les variables, tous les domaines sont explicités, la complexité du calcul de l'engagement est celle du calcul de l'intersection des domaines. Sur IxTeT , cette complexité est linéaire en fonction de la taille du domaine des variables.

Sur la figure 2.8 par exemple, l'engagement lié à l'insertion de la résolvante $?x_1 \neq ?x_5$ est $engagt(\mathcal{P}, [(?x_1 \neq ?x_5)]) = card(\{c, d\} \cap \{b, c\})/card(\{c, d\}).card(\{b, c\}) = 0.25$.

3.6.1.3 Engagement lié à une contrainte d'égalité

Le cas d'une contrainte d'égalité est le dual de celui d'une contrainte de différence, l'ensemble des instances éliminées par l'une de ces contraintes est exactement l'ensemble des instances conservé par l'autre et vice-versa. Nous estimons donc :

$$engagt(\mathcal{P}, [(?x = ?y)]) = 1 - engagt(\mathcal{P}, [(?x \neq ?y)])$$

3.6.1.4 Engagement lié à l'insertion d'un lien causal

En reprenant les notations du paragraphe §3.5.1, insérer une résolvante ρ_{CL} de type lien causal pour expliquer une proposition consiste en :

1. une contrainte temporelle de précédence entre t_{est} et t ;
2. un ensemble de contraintes d'égalité sur les variables $?v_{est} = ?v$ et $?x'_i = ?x_i$; et
3. une assertion protégeant l'attribut d'état valué $attS(?x_1, ..., ?x_n) : ?v$ sur l'intervalle temporel (t_{est}, t).

Une estimation de l'engagement peut facilement être associée aux deux premières constituantes de la résolvante (cf. paragraphes ci-dessus). L'engagement lié à l'insertion de la protection dépend essentiellement de deux facteurs :

1. la longueur temporelle minimale de cette protection qui va empêcher par la suite toute présence d'événement sur $attS$ entre t_{est} et t. Soit $dyn_{attS}(d) \in [0,1]$ une fonction mesurant la dynamicité de l'attribut d'état $attS$ c'est-à-dire, étant donné un intervalle de temps de durée d, la probabilité d'existence, dans un plan, d'au moins un événement sur $attS$ ayant lieu dans cet intervalle ; et
2. la manière dont la valeur protégée de l'attribut $attS$ exclut les autres valeurs possibles. Ceci est estimé en comparant la taille des domaines de $D_{?v_{est}} \cap D_{?v}$ (domaine de la valeur protégée) et D_{attS} le domaine général des valeurs possibles de l'attribut $attS$.

On estime alors :

$$\begin{aligned} 1 - engagt(\mathcal{P}, \rho_{CL}) \quad = \quad & [1 - engagt(\mathcal{P}, (t_{est} < t))]. \\ & [1 - engagt(\mathcal{P}, (?v_{est} = ?v))]. \prod_{i \in [1,n]} [1 - engagt(\mathcal{P}, (?x'_i = ?x_i))]. \\ & [1 - dyn_{attS}(d)]. \frac{card(D_{?v_{est}} \cap D_{?v})}{card(D_{attS})} \end{aligned}$$

où, si sur le plan $\mathcal{P}$ on a $t - t_{est} \in [d_{min}, d_{max}]$, $d = max(0, d_{min})$.

La notion de dynamicité d'un attribut introduite ici est relativement intuitive : sur un chantier de construction automatisé, on peut par exemple s'attendre à ce que l'attribut décrivant la position d'un robot soit plus dynamique (c'est-à-dire évolue statistiquement plus souvent) que la localisation d'une bétonnière et donc, qu'à durée égale, une protection sur la position du robot soit plus contraignante qu'une protection sur la localisation de la bétonnière. Dans notre implémentation, cette notion de dynamicité dépendant des attributs n'a pas été codée. Nous utilisons simplement une estimation $dyn(d) = d/D$ si D est la durée totale minimale du plan partiel.

Sur l'exemple de lien causal présenté sur la figure 3.8, on va ainsi estimer :

$$1 - engagt(\mathcal{P}, \rho_{CL}) = [1 - 1/5].[1 - 3/4].[1 - 1/2].[1 - 0].(1/2)$$

Puisqu'ici, $d = 0$ et de manière générale, $Dyn_{attS}(0) = 0$.

On obtient donc $engagt(\mathcal{P}, \rho_{CL}) = 0.95$.

3.6.2 Une recherche opportuniste

Nous avons vu dans les paragraphes précédents comment est effectué le choix d'une résolvante pour un défaut donné (point 4 de l'algorithme global), reste un second choix à éclairer : celui du défaut qui va être résolu.

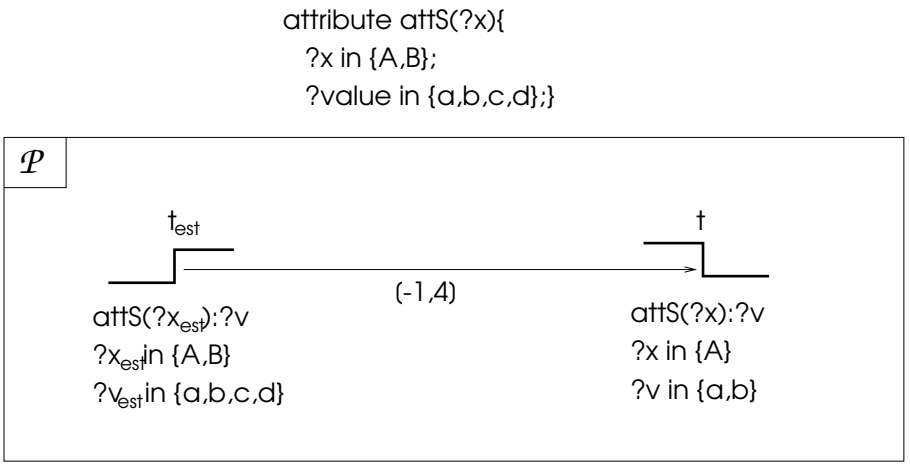


Figure 3.8: Engagement lié à l'insertion d'un lien causal

En l'absence de critère de choix plus informé [11], IXTET utilise une stratégie opportuniste en sélectionnant toujours les défauts pour lesquels le choix de la résolvante à insérer sera le plus évident.

Un cas trivial est celui des défauts ne possédant qu'une seule résolvante (que nous appelons défauts **déterministes**). Il est clair que l'on a tout intérêt à sélectionner ces défauts en priorité puisqu'ils ne font pas apparaître de branchement supplémentaire sur l'arbre de recherche. De manière plus générale, une heuristique consisterait à choisir en priorité le défaut ayant le nombre le plus faible de résolvantes. Une telle heuristique est décrite dans [Yang 92] pour le planificateur WATPLAN. Des heuristiques du même type sont couramment utilisées lors de l'exploration de la recherche d'une solution à un CSP ; il s'agit alors de choisir d'instancier la variable dont le domaine de valeurs est le plus petit. L'idée principale derrière ce type d'heuristique est d'effectuer en priorité les choix qui sont les plus faciles à effectuer et d'attendre d'être dans un contexte plus informé, plus complet pour effectuer les choix plus délicats. Dans IXTET , nous lions donc l'opportunité de la résolution d'un défaut à la facilité de choisir une de ses résolvantes. Un critère $opp(\mathcal{P}, \Phi)$ mesure cette opportunité en estimant dans quelle mesure la meilleure résolvante est isolée des autres quant à l'engagement associé :

$$\frac{1}{opp(\mathcal{P}, \Phi)} = \sum_{\rho \in résolvantes(\Phi)} \frac{1}{1 + engagt(\mathcal{P}, \rho) - engagt(\mathcal{P}, \rho_{min})}$$

On a toujours $opp(\mathcal{P}, \Phi) \in]0, 1]$.

Différents défauts classés par ordre d'opportunité décroissante sont représentés sur la figure 3.9 avec la répartition de l'engagement associé à chacune de leurs résolvantes.

Dans le cas où le même engagement est associé à toutes les résolvantes d'un défaut, $opp(\mathcal{P}, \Phi)$ est égal à l'inverse du nombre de résolvantes de ce défaut. Dans le cas d'un défaut déterministe, l'opportunité est maximale : $opp(\mathcal{P}, \Phi) = 1$.

[11] Nous verrons, au chapitre 5, comment une hiérarchisation de la recherche peut permettre de mieux informer ce choix.

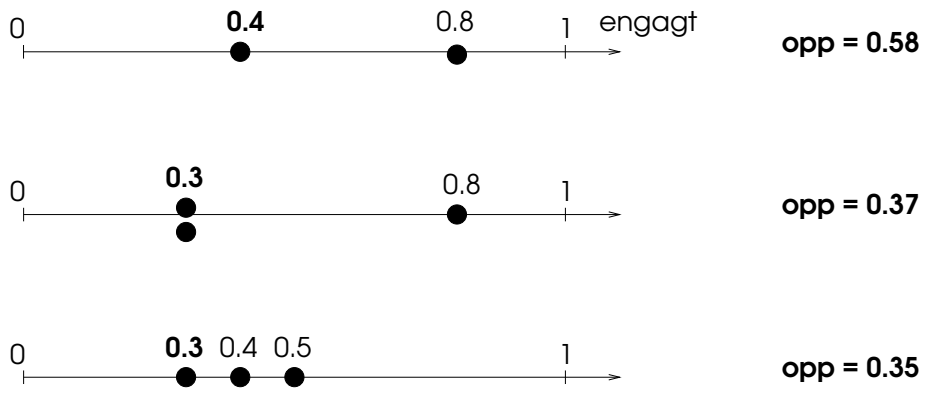


Figure 3.9: Opportunité d'un défaut

3.7 Analyse des sous-buts

Le module d'analyse des sous-buts est chargé de détecter les propositions non-expliquées et, pour chacune de celles-ci, de calculer l'ensemble des résolvantes associées et leur intérêt relatif en terme d'engagement.

Dans notre représentation d'un plan partiel, toutes les propositions non-expliquées sont explicitées, leur détection ne pose donc aucun problème.

Pour une proposition non-expliquée donnée, les résolvantes consistant en des liens causaux entre un événement déjà dans le plan et la proposition sont calculées en parcourant tous les événements sur l'attribut $attS$ susceptibles d'être des établisseurs de la proposition. Si n_e est le nombre maximal d'événements liés à un attribut donné et d_V la taille maximale des domaines des variables, cette étape est d'une complexité en $O(n_e.d_V)$ pour une proposition non-expliquée donnée.

Une deuxième étape[12] consiste à calculer les résolvantes passant par l'insertion d'une nouvelle tâche au plan partiel. Pour estimer l'intérêt d'une tâche quant à établir la proposition non-expliquée, nous effectuons une procédure de "forward checking" consistant à estimer le travail supplémentaire qui devra être fourni pour expliquer les nouveaux sous-buts insérés avec la tâche. Cette procédure, décrite plus en détail dans [Laruelle 94], §5.7, consiste en une planification qui ignore les interactions (positives ou négatives) entre opérateurs. Un graphe ET/OU est développé jusqu'à une profondeur p paramétrable dont les nœuds ET correspondent à la résolution de l'ensemble des sous-buts introduits par une tâche et les nœuds OU l'ensemble des tâches susceptibles d'établir un sous-but donné. Un exemple d'un tel graphe est donné sur la figure 3.10 dans le cadre de la planification de la production d'objets possédant des propriétés particulières (forme, couleur, ...) dans un atelier. Le résultat de cette procédure est une pondération des tâches susceptibles d'établir la proposition non-expliquée en fonction de l'estimation de la complexité entrainée par leur insertion dans le plan.

[12]Cette étape est effectuée même s'il existe dans le plan partiel des événements susceptibles d'être établisseurs.

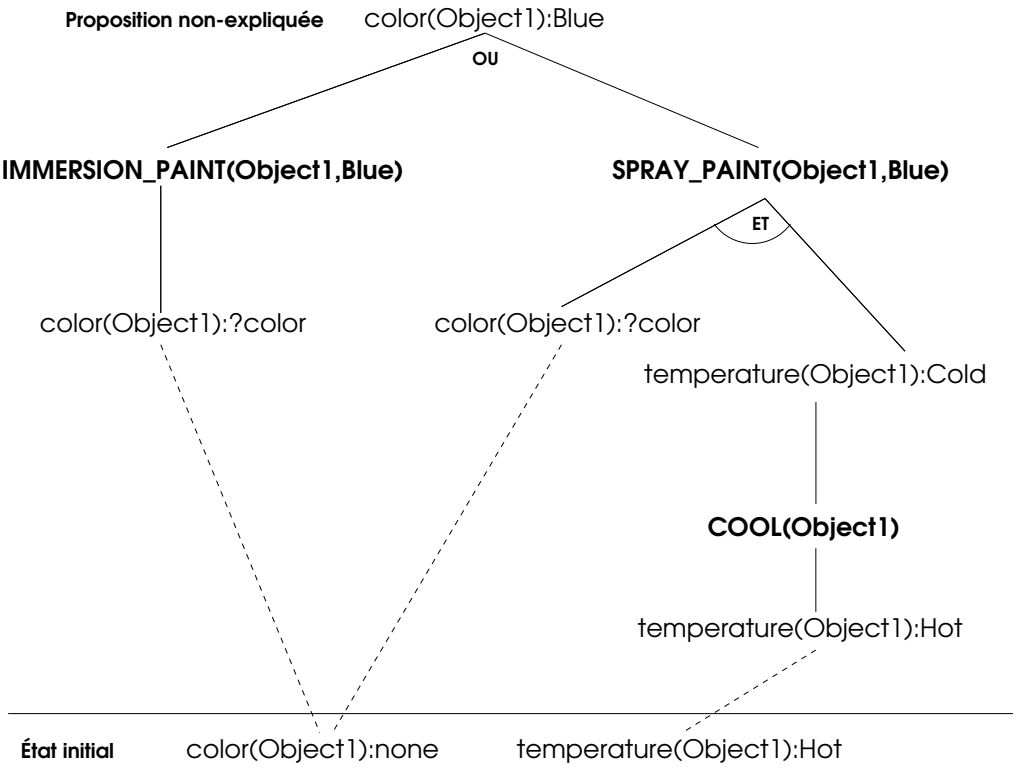


Figure 3.10: Graphe ET/OU pour l'analyse d'une proposition non-expliquée

Si τ désigne le nombre maximal de tâches susceptibles d'établir une proposition sur un attribut $attS$ donné et β le nombre maximal de sous-buts introduits par une tâche (c'est-à-dire le nombre de propositions d'état *event* et *hold* de la tâche), la complexité de cette procédure pour une proposition non-expliquée donnée est de l'ordre de $O(\tau.(\beta.\tau)^p)$.

Nous avons constaté en pratique qu'une grande valeur de p pénalise beaucoup trop le temps d'analyse des sous-buts par rapport au gain en terme de précision de l'estimation de l'engagement. La valeur optimale de p se situe en général dans l'intervalle $\{0, 1, 2\}$.

3.8 Analyse des menaces

L'analyse des menaces consiste à parcourir les couples $(e, h_P) \in EVT_{attS} \times HLD_{attS}$ et $(h, h_P) \in HLD^2_{attS}$ à la recherche de menaces.

On s'aperçoit toutefois qu'il n'est pas nécessaire de parcourir cet espace en totalité. Pour un plan partiel donné, EVT_{attS} peut se décomposer en quatre sous-ensembles selon que l'événement considéré est expliqué ou non et selon qu'il sert à expliquer une proposition ou non. De même, les assertions HLD_{attS} peuvent se diviser en quatre groupes selon qu'elles sont ou non expliquées et selon qu'elles se terminent ou non par un événement. Les menaces potentielles peuvent donc être réparties en trente deux ensembles disjoints selon la caractéristique des deux propositions qui la constituent. Ces ensembles sont présentés sur la figure 3.11.

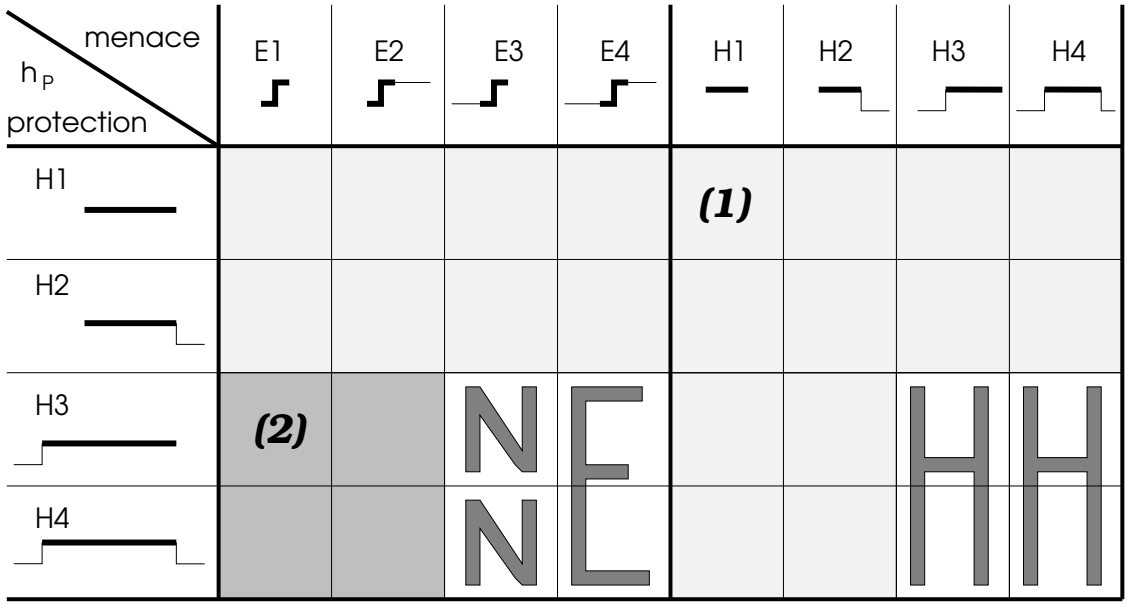


Figure 3.11: Les différents types de menaces

Les types d'assertions H1, H2, H3 et H4 désignent respectivement les assertions non-expliquées ne se terminant pas par un événement, les assertions non-expliquées se terminant par un événement, les assertions expliquées ne se terminant pas par un événement et les assertions expliquées se terminant par un événement. De manière similaire, E1, E2, E3 et E4 désignent respectivement les événements non-expliqués et n'expliquant aucune proposition temporelle, les événements non-expliqués expliquant une proposition, les événements expliqués n'expliquant aucune proposition et les événements expliqués expliquant une proposition. Les zones apparaissant en grisé sur la figure sont des conflits qu'il n'est pas *nécessaire* de gérer pour conserver la complétude de l'algorithme de planification (ce qui ne veut absolument pas dire que leur gestion n'améliore pas les performances globales du système).

Les conflits des zones (1) et (2) peuvent ne pas être gérés dans la mesure où ils font intervenir au moins une proposition non-expliquée et que cette proposition sera nécessairement expliquée en cours de planification, faisant ainsi basculer le conflit dans une autre zone.

Dans le cas (1), il s'agit de conflits faisant intervenir au moins une assertion non-expliquée. Dans ces cas-là, on peut avoir intérêt à attendre que l'assertion ait été expliquée puisqu'elle aura alors atteint son étendue temporelle maximale (cf. fig 3.12).

Le cas (2) est très similaire pour une menace par un événement.

Les deux cases N sur la figure 3.11 doivent nécessairement être gérées car l'événement menaçant risque de ne jamais expliquer une autre proposition (s'il se situe en fin de plan) et sa nouvelle valeur peut entrer en conflit avec une protection. Selon que l'on considère une approche de la gestion des menaces plutôt basée sur les événements ou sur les assertions, il sera aussi nécessaire de gérer soient les deux cases E, soient les quatre cases H.

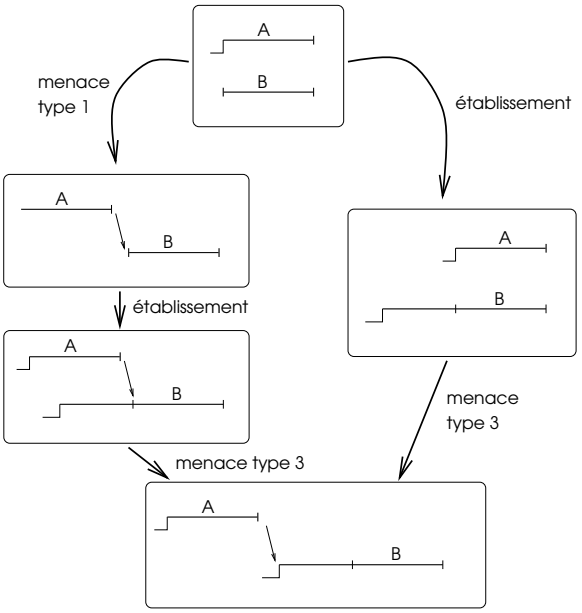


Figure 3.12: Conflit faisant intervenir une assertion non-expliquée

3.9 Filtrage des résolvantes

Nous avons déjà noté un certain nombre d'analogies entre notre représentation du plan sous la forme d'un ensemble de défauts auxquels sont associés des ensembles de résolvantes et les problèmes de satisfaction de contraintes. Le rapprochement défaut/variable et résolvante/valeur est évident. Dans ce paragraphe, à partir de l'approche décrite dans [Yang 92] pour le planificateur WATPLAN, nous proposons une manière de filtrer le "CSP" défauts/résolvantes afin de le simplifier avant de le résoudre.

3.9.1 L'approche utilisée dans WATPLAN

Partant de la constatation que l'ordre de la gestion des conflits a un effet très important sur les performances globales de la planification, l'auteur montre la nécessité d'une approche globale utilisant des techniques de filtrage afin d'orienter efficacement la recherche et de détecter précocement les voies sans issues.

Plus précisément, à partir d'une représentation des conflits d'un plan partiel Π sous la forme d'un ensemble $\{Conf_1, ..., Conf_n\}$, chaque conflit $Conf$ disposant d'un ensemble de méthodes de résolutions $M(Conf) = \{R_1, ..., R_k\}$ (conjonction de contraintes de précédence temporelle, égalité ou différence sur des variables atemporelles), deux relations sont définies :

- R_1 subsume R_2 dans un plan Π (noté $S_\Pi(R_1, R_2)$) si et seulement si l'insertion de R_1 sur Π (que nous noterons $\Pi \oplus R_1$) entraîne de manière nécessaire la vérification de R_2 sur ce nouveau plan partiel ;

- R_1 est inconsistant avec R_2 sur Π (noté $I_\Pi(R_1, R_2)$) si et seulement si on ne peut pas les insérer simultanément sur Π.

Deux types de filtrage sont alors proposés : l'élimination de méthodes de résolutions pour un conflit donné et l'élimination de conflits redondants.

Soient par exemple deux conflits $Conf$ et $Conf'$ tels que :

$$M(Conf) = \{[x = y, b < c], ...\}$$
$$M(Conf') = \{[x \neq y], [c < b]\}$$

Il est clair que comme $I(x = y, x \neq y)$ et $I(b < c, c < b)$, la contrainte $[x = y, b < c]$ ne pourra pas être utilisée pour lever le conflit $Conf$. Ceci correspond au filtrage par consistance d'arc utilisé dans les CSP classiques.

Propriété 1 *Pour chaque couple de conflit $(Conf, Conf')$, on peut éliminer de $M(Conf)$ toutes les méthodes R vérifiant : $\forall R' \in M(Conf'), I_\Pi(R, R')$.*

Cette propriété peut être étendue pour éliminer encore plus de résolvantes :

Propriété 2 *Soient $Conf, Conf'$ deux conflits d'un plan partiel Π. Supposons que $\exists R' \in M(Conf)$ tel que $\forall R \in M(Conf)$*

1. *soit $I_\Pi(R, R')$,*
2. *soit $\exists R'' \in M(Conf')$ tel que $R' \neq R''$ et $S_\Pi(R, R'')$*

alors, R' peut être éliminé de l'ensemble des méthodes de résolutions de $Conf'$.

En effet, dans ce cas-là, il est facile de voir que la contrainte R choisie pour résoudre $Conf$ vérifiera soit le point (1) et dans ce cas, R' est inutile car inconsistante avec le plan partiel, soit le point (2) et alors, $Conf'$ est déjà résolu par R.

La propriété suivante permet d'éliminer la redondance au niveau des conflits :

Propriété 3 *Soient $Conf, Conf'$ deux conflits d'un plan partiel Π. Si $\forall R \in M(Conf), \exists R' \in M(Conf')$ tel que $S_\Pi(R, R')$ alors $Conf'$ peut être éliminé de l'ensemble des conflits à résoudre.*

La justification de cette propriété est évidente : en résolvant $Conf$ par l'insertion d'une de ses méthodes de résolution R, le nouveau plan partiel $\Pi \oplus R$ contiendra une contrainte R' et donc, $Conf'$ sera, de fait, aussi résolu.

L'application des propriétés ci-dessus permet de simplifier le CSP afin de pouvoir faire un choix prenant en compte des considérations globales d'interdépendance entre les méthodes de résolutions. Le surcoût de la vérification de ces propriétés en terme de complexité est toutefois loin d'être négligeable.

3.9.2 Filtrage des résolvantes sur IXTET

Bien que cela n'ait pas encore été implémenté sur IXTET , toutes les propriétés décrites dans le paragraphe précédent peuvent être envisagées en remplaçant la notion de "conflit" par la notion, plus générale, de "défaut" et le terme de "méthode de résolution" par "résolvante".

La complexité de la vérification de chacune des propriétés est donnée dans le tableau ci-dessous.

Propriété	Complexité
propriété 1	$O(d^2.r^2.I)$
propriété 2	$O(d^2.r^2.(I + r.S))$
propriété 3	$O(d^2.r^2.S)$

Où d désigne le nombre de défauts, r le nombre de résolvantes par défaut, et I et S les complexités respectives de vérification des relations $I_{\mathcal{P}}(\rho, \rho')$ et $S_{\mathcal{P}}(\rho, \rho')$.

En général, la vérification des relations I et S entre deux résolvantes sur un plan partiel demande d'insérer effectivement et de propager une des deux résolvantes sur le plan partiel $\mathcal{P}$. Il est clair que vu le nombre de fois où une telle vérification est effectuée, l'application telle quelle des propriétés n'est pas envisageable en pratique.

La complexité peut toutefois être réduite de deux manières différentes si l'on accepte de ne pas éliminer du CSP tout ce qui pourrait l'être par une vérification à la lettre des propriétés :

1. réduire le paramètre d en ne filtrant pas tous les défauts mais seulement un sous-ensemble de ceux-ci. Les propriétés (1) et (2) peuvent par exemple être utilisées lorsque l'on a déjà sélectionné un défaut pour éliminer certaines de ses résolvantes en le filtrant par rapport aux $d - 1$ autres défauts ;

2. réduire les paramètres I et S en ne vérifiant pas ces relations sur les réseaux de contraintes C_T et C_V du plan $\mathcal{P}$ lui-même, mais sur des sous-réseaux plus petits, moyennant la remarque que si une relation $I_{\mathcal{P}}(\rho, \rho')$ ou $S_{\mathcal{P}}(\rho, \rho')$ est vraie sur un sous-réseau, elle l'est aussi sur le réseau total. Le cas le plus simple est celui où ces sous-réseaux sont vides ; dans ce cas, I et S sont en $O(1)$. Une autre possibilité, un peu plus coûteuse, peut être de n'extraire de $\mathcal{P}$ que le sous-réseau correspondant aux variables (temporelles ou atemporelles) utilisées dans les deux résolvantes.

3.10 Exploration de l'arbre de recherche

L'arbre de recherche global est exploré grâce à un algorithme A_ϵ [Ghallab 83]. Notre objectif pour la recherche d'une solution est double : trouver une solution en un minimum de temps de calcul et trouver une solution de "bonne" qualité. Ces deux points ne sont toutefois pas incompatibles : dans le cadre d'une stratégie de moindre engagement (visant à améliorer les performances du système), les résolvantes choisies en priorité sont celles qui surcontraignent

le moins le plan partiel et, ce faisant, on aboutit à une solution peu contrainte et très souple dans le sens où elle représente une très grande variété d'instances possibles, et ceci correspond bien, en général, à la conception intuitive d'un "bon" plan.

L'estimation d'un nœud (f) correspond à la combinaison entre l'engagement lié à l'insertion de toutes les résolvantes depuis le nœud racine (g) et une estimation heuristique (h) de l'engagement minimal qu'il resterait à effectuer sur le plan partiel pour arriver à une solution. Cette heuristique h est calculée en cumulant l'engagement lié aux meilleures résolvantes des défauts non-résolus du plan partiel.

Plus formellement, si $\rho(N)$ désigne la résolvante insérée pour passer du nœud $Parent(N)$ à N et si l'on confond N avec le plan partiel représenté à ce nœud, on a:

$$g(N) = \sum_{M \in Ascendants(N)} engagt(Parent(M), \rho(M))$$

$$h(N) = \sum_{\Phi \in DEFAUTS(N)} engagt(N, \rho_{min}(\Phi))$$

où $\rho_{min}(\Phi)$ est la résolvante de Φ de moindre engagement.

Si les défauts étaient indépendants les uns des autres, h serait une estimation exacte. En pratique, les défauts sont largement interdépendants et h n'est donc qu'une estimation ; d'autre part, du fait des interactions positives entre les résolvantes (une même résolvante peut très bien résoudre plusieurs défauts), h n'est pas nécessairement une heuristique minorante. On constate toutefois, de manière empirique, que cette heuristique est relativement efficace pour guider l'exploration de l'arbre de recherche et conduit à des plans solutions de bonne qualité.

3.11 Conclusion

A partir de la formalisation de la notion de *plan solution* associé à un problème de planification donné, nous avons dans ce chapitre proposé une *algorithmique* générale permettant de générer un tel plan. Cette algorithmique a été prouvée *saine* et *complète au sens restreint*. Une des originalité de notre approche est la gestion en *concomitance* et selon une stratégie de *moindre engagement* de tous les défauts du plan partiel courant, indépendamment du fait qu'il s'agisse de propositions non-expliquées, de menaces ou d'ensembles critiques minimaux. Un des interêts de l'approche est de pouvoir prendre en compte, très tôt dans la recherche, certains défauts très déterministes qu'un ordre plus rigide sur la résolution des défauts aurait peut être relégué en fin du processus. La représentation du plan sous la forme d'un *CSP défauts/résolvantes* présente aussi d'intéressantes perspectives quant au filtrage

de celui-ci. Enfin, notre approche de moindre engagement combinée à des heuristiques adéquates permet d'aboutir, de manière générale, à des plans solutions peu contraints et très flexibles ; qualités qui font en général partie des critères définissant un "bon" plan solution.

Le chapitre suivant détaille le module de gestion des ressources, c'est-à-dire le problème de la détection des ensembles critiques minimaux et de la génération d'un ensemble minimal de résolvantes associées.

Chapitre 4

Gestion de ressources partageables

4.1 Introduction

Nous traitons essentiellement dans ce chapitre la *recherche d'ensembles critiques minimaux* sur un plan partiel et le calcul des *résolvantes associées* ; résolvantes que nous avons déjà évoquées au paragraphe 3.5.3.

Nous restreignons dans un premier temps le problème à celui de la gestion de ressources *banalisées*. Nous donnons dans ce cadre-là un algorithme de recherche des ECM qui est prouvé *sain*, *systématique* et *complet*. Notre algorithme est basé sur une représentation originale des contraintes temporelles du plan partiel sous la forme d'un *graphe d'intersections possibles* entre propositions d'emprunt de ressources [Laborie 94, Laborie 95]. Nous montrons d'autre part, grâce à une analyse de complexité en pire cas, l'efficacité de l'algorithme proposé.

Pour ce qui est du calcul des résolvantes, après avoir développé un bref corpus théorique sur la *minimisation de contraintes temporelles disjonctives*, nous appliquons nos résultats à la minimisation du nombre de résolvantes suffisantes pour résoudre un ECM donné. Cette élimination de la redondance au sein des résolvantes d'un ECM permet de diminuer le facteur de branchement dans l'arbre de recherche global tout en conservant la propriété fondamentale de *complétude au sens restreint* de notre algorithme de planification.

Les restrictions posées préalablement sont discutées dans les deux derniers paragraphes de ce chapitre où nous montrons comment l'allocation et la production de ressources sont prises en compte dans notre approche. Les points principaux abordés dans ce chapitre sont résumés sur la figure 4.1.

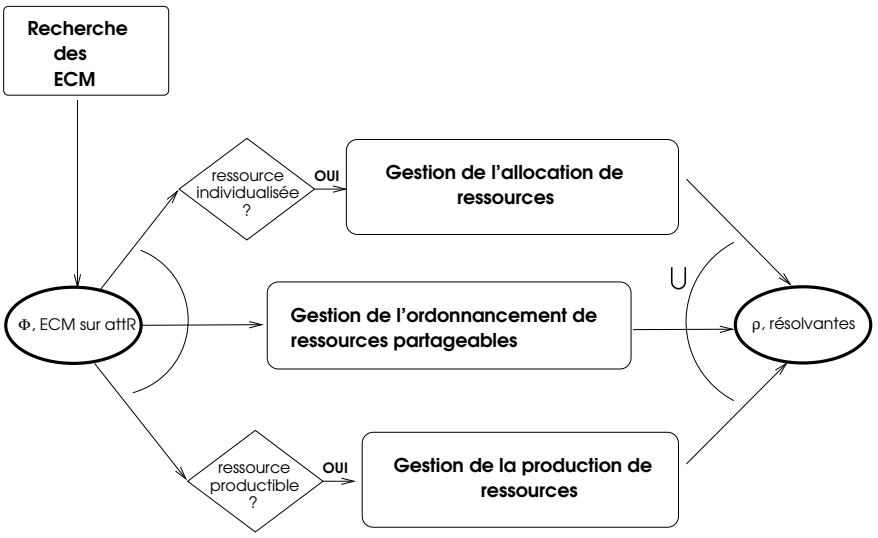


Figure 4.1: Le module d'analyse des ressources

4.2 Le problème de base

Nous avons donné en §3.3.3, Définition 3, la définition d'un ensemble critique minimal.

Dans le début de ce chapitre, nous allons traiter uniquement le cas de ressources partageables *banalisées* (ou complètement agrégées), c'est-à-dire dont le type associé ne contient qu'une ressource. Il n'y a donc pas d'argument dans l'attribut de ressource. Par exemple, la puissance électrique maximale : `resource PUISSANCE_ELEC(){capacity()=100;}`.

Dans ce cas-là, la définition d'un ECM sur une telle ressource $attR$ est la suivante :

Définition 5 *On appelle* **ensemble critique minimal** *tout sous-ensemble* $U = \{use(attR() : q_k, (t_k^-, t_k^+))\}_{k\in[1,m]} \subset USE$ *vérifiant les trois conditions suivantes :*

1. $\sum_{k=1}^{k=m} q_k > Q_0(attR())$;
2. C_T *n'est pas incompatible avec la contrainte* $\bigwedge_{k,l\in[1,m]}(t_k^- < t_l^+)$; *et*
3. $\forall V \subset U, V \neq U,$ V *n'est pas un ECM.*

Nous faisons cette restriction pour des raisons de simplification de la présentation de l'algorithme mais aussi parce que la majorité des ressources partageables sont banalisées puisque souvent, ce sont des ressources que l'on a agrégées pour ne pas gérer explicitement leur allocation (voir la notion d' "aggregate resource" dans [Muscettola 93]). Nous verrons toutefois dans le dernier paragraphe comment étendre notre approche aux ressources partageables quelconques.

Si l'on s'en tient à l'analyse des ressources, le problème de planification revient alors à résoudre l'ensemble des ECM afin que le plan partiel courant représente une classe d'instanciations sans conflit.

Ci-dessous sont définies deux relations temporelles portant sur des sous-ensembles d'intervalles qui vont permettre la détection des ECM sur le plan partiel courant.

4.3 Graphes d'intersections possibles

Gérer des ressources consiste essentiellement à gérer des intervalles utilisateurs de ressources [Tessier-Badie 88] [Parrod 92] plutôt que des instants indépendants modifiant la disponibilité de celles-ci. La représentation des contraintes temporelles sous la forme de relations de l'algèbre d'intervalles générale [Allen 83a] présente toutefois des désavantages puisque la simple vérification de la cohérence du réseau est un problème NP-complet [Vilain 86] ; c'est pourquoi la majorité des systèmes de planification ou d'ordonnancement gérant le temps de manière explicite reposent sur une représentation des contraintes temporelles au niveau des instants (graphe PERT, treillis temporel [Ghallab 89a]) qui ne permet pas d'exprimer des contraintes temporelles disjonctives mais présente l'avantage d'assurer une vérification de cohérence et une propagation en temps polynômial. Cela correspond en terme d'intervalles à ne gérer qu'un sous-ensemble de l'algèbre d'intervalles : l'algèbre d'intervalles restreinte qui est équivalent à l'algèbre d'instants. D'autres restrictions polynômiales de l'algèbre d'intervalles ont été proposées dans [Golumbic 92] comme par exemple le sous-ensemble $\Delta_1 = \{b, b', \cap, b \vee \cap, \cap \vee b', b \vee \cap \vee b'\}$ que nous avons évoqué au chapitre 2 et qui est représenté sur la figure 4.2.

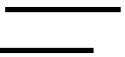
mnem	relation	interprétation	algèbre d'instants
b	I before J		I+ < J-
b′	I after J		J+ < I-
∩	I intersect J		I- < J+ et J- < I+
b∩	I intersect before J		I- < J+
∩ b′	I intersect after J		J- < I+
b∩ b′	I don't care J		

Figure 4.2: Le sous-ensemble Δ_1

Pour la gestion de ressources, les contraintes temporelles les plus importantes sont celles qui concernent l'intersection et la non-intersection temporelle entre intervalles puisque les capacités finies des ressources ont pour effet de restreindre le parallélisme entre activités. Or la nature même du problème d'analyse des conflits de ressources oblige le système à partir d'une représentation des contraintes temporelles qui soit commune avec celle de la gestion

des états (qui utilise l'algèbre d'instants). Ceci justifie l'adoption d'une représentation intermédiaire qui permette d'adapter la généricité de la représentation du plan à la spécificité des algorithmes de gestion de ressources.

Du point de vue de la gestion des ressources, la représentation des connaissances temporelles sous la forme d'un graphe dont les sommets représentent les propositions temporelles d'emprunt de ressources (USE)[1] et les arcs les connaissances temporelles (C_T) exprimées sur le domaine Δ_1 présente plusieurs avantages ; dont en particulier,

- une traduction facile permettant de passer de Δ_1 à l'algèbre d'instants et vice-versa ; et
- une représentation naturelle de la notion d'intersection entre intervalles temporels tout en conservant une expressivité suffisante pour détecter les conflits de ressources potentiels (intersection possible entre n intervalles) et permettant d'avoir une idée sur leur degré de déterminisme.

Etant donné un réseau de contraintes temporelles exprimées sur l'algèbre d'instants par un ordre partiel $\leq$, on appellera $\Delta_1 - graphe$ associé à ce réseau, le graphe orienté représentant la traduction de ces contraintes dans le domaine Δ_1, les arêtes étant minimalement étiquetées.

Soit G un $\Delta_1 - graphe$, considérons G' le sous-graphe non-orienté de G constitué uniquement des relations contenant $\cap$ (c'est-à-dire $\{\cap, b \vee \cap, \cap \vee b'$ et $b \vee \cap \vee b'\}$). Ce graphe capturant l'idée d'intervalles pouvant s'intersecter deux à deux dans une des instanciations cohérentes du réseau de contraintes temporelles, sera nommé **graphe d'intersections possibles (GIP)** associé au réseau initial.

Un exemple de graphe de GIP associé à un réseau de contraintes temporelles est présenté sur la figure 4.3.

La propriété suivante sur la structure topologique des GIP est facile à montrer[2]:

Propriété 4 (caractérisation des GIP) *Les graphes d'intersections possibles sont exactement les graphes complémentaires des* graphes de comparabilité. *En particulier, les graphes d'intersections possibles sont des* graphes faiblement triangulés.

En effet, si G est un GIP et G^c son complémentaire, l'existence d'une arête $\{I, J\}$ sur G^c signifie que l'intersection entre les intervalles temporels I et J est impossible c'est-à-dire que l'on a nécessairement (au niveau des intervalles) IbJ ou $Ib'J$; relation qui induit sur les sommets de G^c une orientation représentant une relation d'ordre.

[1] A noter que pour des raisons de clarté, nous identifierons dans ce paragraphe une proposition d'emprunt de ressources à l'intervalle temporel pendant lequel elle à lieu.

[2] Pour plus de précisions sur les graphes cités dans ce chapitre, on se reportera à l'annexe B.

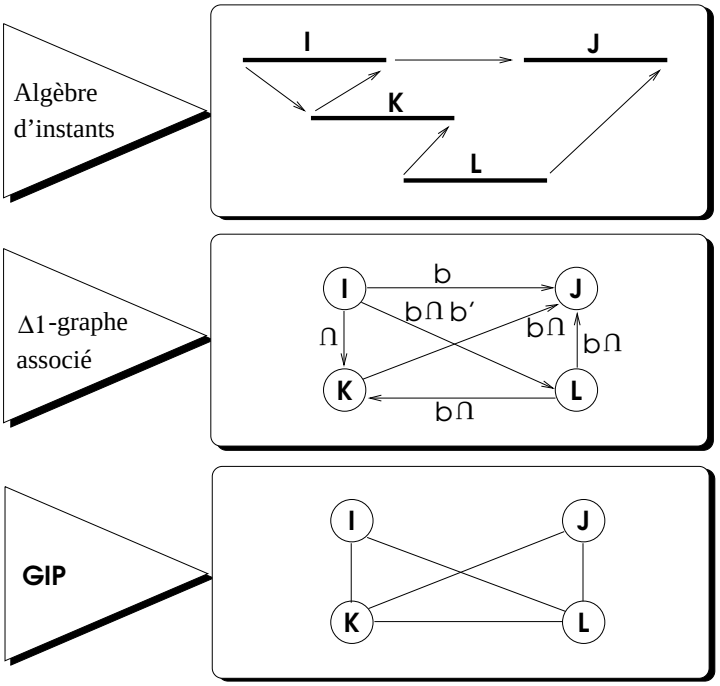


Figure 4.3: Un exemple de GIP

Définition 6 (Relation d'intersection possible) *Etant donné un réseau de contraintes temporelles C_T donné, un sous-ensemble d'intervalles $U = \{I_i\}_{i\in[1,n]}$ sera dit en intersection possible (sur C_T) s'il existe une instanciation temporelle d_T des contraintes de C_T qui vérifie:*

$$\bigcap_{i\in[1,n]} d_T(I_i) \neq \emptyset$$

où l'intersection est celle des intervalles de $\mathbb{R}$. On notera alors $\Diamond \cap_{C_T}(U)$ (ou plus simplement $\Diamond \cap (U)$ lorsqu'il n'y a pas d'ambiguïté). $\neg\Diamond \cap (U)$ désigne la négation de $\Diamond \cap (U)$ (intersection impossible).

Définition 7 (Relation d'intersection nécessaire) *Etant donné un réseau de contraintes temporelles C_T donné, un sous-ensemble d'intervalles $U = \{I_i\}_{i\in[1,n]}$ sera dit en intersection nécessaire (sur C_T) si et seulement si quelle que soit l'instanciation temporelle d_T des contraintes de C_T on a :*

$$\bigcap_{i\in[1,n]} d_T(I_i) \neq \emptyset$$

où l'intersection est celle des intervalles de $\mathbb{R}$. On notera alors $\Box \cap_{C_T}(U)$ (ou plus simplement $\Box \cap (U)$ lorsqu'il n'y a pas d'ambiguïté).

Sur l'exemple de la figure 4.3, on a ainsi les relations : $\Diamond \cap (I, K, L)$, $\Box \cap (I, K)$.

Propriété 5 *Soit U un sous-ensemble d'intervalles, $\Box \cap (U) \Rightarrow \Diamond \cap (U)$.*

Propriété 6 (Décomposabilité de la relation d'intersection nécessaire) *soit U un sous-ensemble d'intervalles :*

$$\Box \cap (U) \Leftrightarrow \forall V \subset U, \Box \cap (V) \Leftrightarrow \forall \{I, J\} \subset U, \Box \cap (\{I, J\})$$

Propriété 7 (Décomposabilité de la relation d'intersection possible) *soit U un sous-ensemble d'intervalles :*

$$\Diamond \cap (U) \Leftrightarrow \forall V \subset U, \Diamond \cap (V) \Leftrightarrow \forall \{I, J\} \subset U, \Diamond \cap (\{I, J\})$$

La preuve de la première propriété est immédiate ; celle de la suivante utilise le fait que sur $\mathbb{R}$, comme les intervalles sont connexes, l'intersection globale équivaut à l'intersection deux à deux. La propriété 7 est un peu plus délicate à montrer (cf Annexe D).

Les deux dernières propriétés sont essentielles car elles permettent de considérer des relations n-aires comme un ensemble de relations binaires sur le plan partiel courant. En terme de graphe, la dernière propriété signifie qu'un sous-ensemble d'intervalles s'intersecte possiblement si et seulement s'il représente une *clique* sur le graphe d'intersection possible associé au plan courant.

Remarque : De manière générale, les contraintes temporelles ne sont pas constituées uniquement de contraintes symboliques de précédence mais aussi de contraintes numériques de la forme $d(t2) - d(t1) \in [d_{min}, d_{max}]$. On peut toutefois déduire du réseau de contraintes numériques l'ensemble des contraintes symboliques (les couples (t1, t2) pour lesquels $d_{min} > 0$) et le graphe d'intersections possibles associé à ces contraintes. Dans le cas où toutes les contraintes numériques sont des contraintes contrôlables [Ghallab 95b], on peut montrer que la notion d'intersection possible vis à vis des contraintes symboliques déduites équivaut à la notion d'intersection possible vis à vis des contraintes numériques (à condition bien entendu que la propagation des contraintes numériques soit complète et permette de déduire l'ensemble des contraintes symboliques).

Au terme de ce paragraphe, nous pouvons donc reformuler le problème de la détection d'ECM sur un plan partiel comme celui de la recherche de *cliques sur-consommatrices minimales* sur un GIP. Le paragraphe qui suit propose un algorithme pour effectuer cette recherche.

4.4 Recherche d' ensembles critiques minimaux

Pour une ressource $attR$ donnée, la recherche des ECM par le module de gestion des ressources s'effectue dans un arbre de recherche dont chaque nœud N représente une clique $\Pi(N)$ du graphe d'intersections possibles associé.

Dans le paragraphe suivant, nous définissons la structure d'un nœud de l'arbre de recherche et donnons une série de conditions sur le développement d'un nœud qui suffisent à assurer la *consistance*, la *systématicité* et la *complétude* de la recherche des ECM. Nous présentons

ensuite deux procédures de développement d'un nœud qui vérifient ces conditions et étudions la complexité de la plus prometteuse d'entre elles, laquelle a été implémentée sur le planificateur IXTET . La question de l'exploration de l'arbre de recherche en fonction de la requête posée au module de gestion des ressources est abordée dans un troisième paragraphe.

4.4.1 L'arbre de recherche des ECM

Dans ce paragraphe, nous décrivons l'espace de recherche des ECM. Plutôt que de décrire un algorithme particulier et de démontrer certaines de ses propriétés (correction, systématicité, complétude), nous décrivons l'espace de recherche des ECM par un certain nombre de caractéristiques de cet arbre et montrons que *tous les algorithmes* respectant ces caractéristiques possèdent les propriétés ci-dessus.

Le paragraphe 4.4.1.1 donne le principe de la recherche des ECM. Le paragraphe 4.4.1.2 formalise de manière plus précise les caractéristiques que l'arbre de recherche doit vérifier afin d'assurer ces propriétés.

Etant donné un nœud N dans un arbre de recherche, nous prendrons les notations suivantes (cf. figure 4.4) :

- $Parent(N)$ désigne le nœud parent de N ;
- $Ascendants(N) = \{N, Parent(N), Parent^2(N), ...\}$ désigne l'ensemble des nœuds ascendants du nœud N (N compris) ;
- $Enfants(N)$ désigne l'ensemble ordonné des enfants de N ; par analogie avec un arbre généalogique, on pourra dire qu'un enfant $N1$ d'un nœud N donné est plus jeune qu'un autre enfant $N2$ s'il a été développé avant ; on notera alors $plus\_jeune(N1, N2)$;
- $Cadets(N)$ est l'ensemble des nœuds M frères de N et tels que $plus\_jeune(M, N)$.

4.4.1.1 Principe de la recherche des ECM

L'idée principale de la recherche est d'examiner des cliques $\Pi(N)$ de taille croissante le long d'un chemin de l'arbre de recherche. Si $(N_0, N_1, ..., N_k)$ est un chemin de la racine (clique vide) vers les feuilles de l'arbre on a donc une suite de cliques emboîtées $\Pi(N_0) = \emptyset \subset \Pi(N_1) \subset \ ... \ \subset \Pi(N_k)$.

A un nœud N donné, on note $\Delta(N)$ la contribution du nœud N à l'agrandissement de la clique, c'est-à-dire que $\Pi(N) = \Pi(Parent(N)) \cup \Delta(N)$. Dans ce contexte, développer un nœud N consiste à créer ses enfants, c'est-à-dire à agrandir la clique $\Pi(N)$ dans différentes directions en lui rajoutant certains sommets du GIP. Pour un nœud N, nous noterons $P(N)$ l'ensemble des sommets du GIP qui seront utilisés à l'agrandissement de la clique $\Pi(N)$

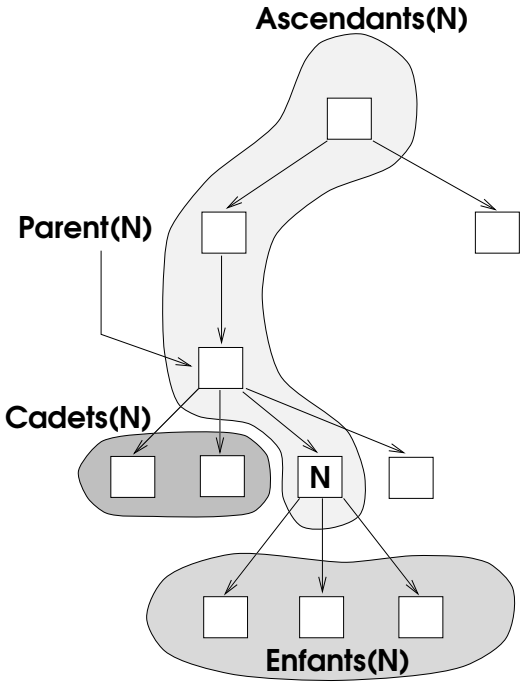


Figure 4.4: Quelques notations sur un arbre de recherche

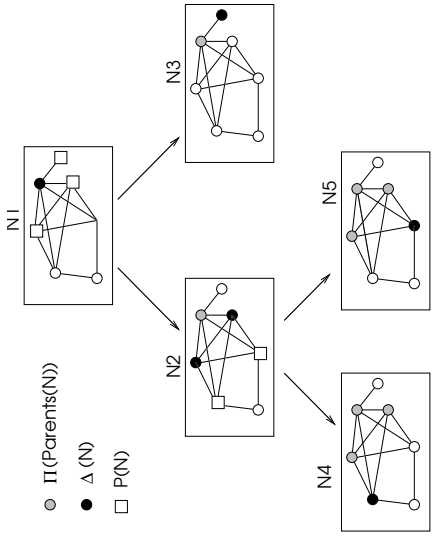


Figure 4.5: Principes de la recherche des ECM

dans les nœuds enfants de N. En d'autre termes : $P(N) = \cup_{M \in Enfants(N)} \Delta(M)$ (voir la figure 4.5).

Enfin, nous associons à un nœud N l'ensemble $\Phi(N)$ des ensembles critiques minimaux appartenant à la clique $\Pi(N)$ et contenant au moins une proposition d'emprunt de ressources de chacun des ensembles $\Delta(M)$ de l'ascendance de N.

4.4.1.2 Cadre formel de la recherche des ECM

Oublions maintenant la sémantique associée aux ensembles $\Pi(N)$, $P(N)$ et $\Delta(N)$ introduite au paragraphe précédent (4.4.1.1) et qui n'avait d'autre objectif que d'éclairer le lecteur sur le principe de la recherche.

Un nœud N de l'arbre de recherche des ECM est représenté par un triplet $< \Delta(N), P(N), \Phi(N) >$; le nœud racine étant $< \emptyset, USE_{attR}, \emptyset >$.

On notera:

$$\Pi(N) = \cup_{M \in Ascendants(N)} \Delta(M)$$

$$\Sigma(N) = \cup_{M \in Cadets(N)} \Delta(M)$$

Voici, de manière formelle, les conditions que nous imposons sur les ensembles caractérisant les noeuds[3] de l'arbre de recherche des ECM:

C1: $\{\Delta(M)\}_{M \in Enfants(N)}$ est une partition en clique de $P(N)$;

C2: $P(N) = \{u \in \Sigma(N) \mid \forall u' \in \Delta(N), \{u, u'\}$ est une arête du GIP$\}$;

C3: $\Phi(N)$ est l'ensemble des sous-ensembles $U \subset \Pi(N)$ sur-consommateurs minimaux qui vérifient: $\forall M \in Ascendants(N), U \cap \Delta(M) \neq \emptyset$.

La condition [C1] permet de caractériser les ensembles $\Delta(N)$ associés à un noeud N.

Etant donné des ensembles $\Delta(N)$ vérifiant [C1], les conditions [C2] et [C3] permettent de construire les ensembles $P(N)$ et $\Phi(N)$.

On peut montrer facilement que [C1] et [C2] confèrent la propriété [P1] suivante:

P1: $\forall N,\ \Delta(N) \cap P(N) = \emptyset$;

Afin de montrer la complétude de nos algorithmes de recherche d'ECM, nous aurons aussi besoin d'imposer la condition suivante:

C4: si U clique de $P(N)$, $\exists M \in Enfants(N) \mid U \subset (\Delta(M) \cup P(M))$ et $U \cap \Delta(M) \neq \emptyset$;

Ces caractéristiques ainsi que les différents ensembles qu'elles mettent en jeu sont représentées sur la figure 4.6.

On peut alors montrer les trois théorèmes suivants portant sur la recherche d'ensembles critiques minimaux sur la base d'un tel arbre (leur preuve est donnée en annexe D). Ces théorèmes justifient a posteriori le principe de la recherche présenté au paragraphe 4.4.1.1.

Soit T un arbre de recherche d'ECM vérifiant les propriétés [C1 à C3].

[3] A l'exception du noeud racine qui ne vérifie pas C2.

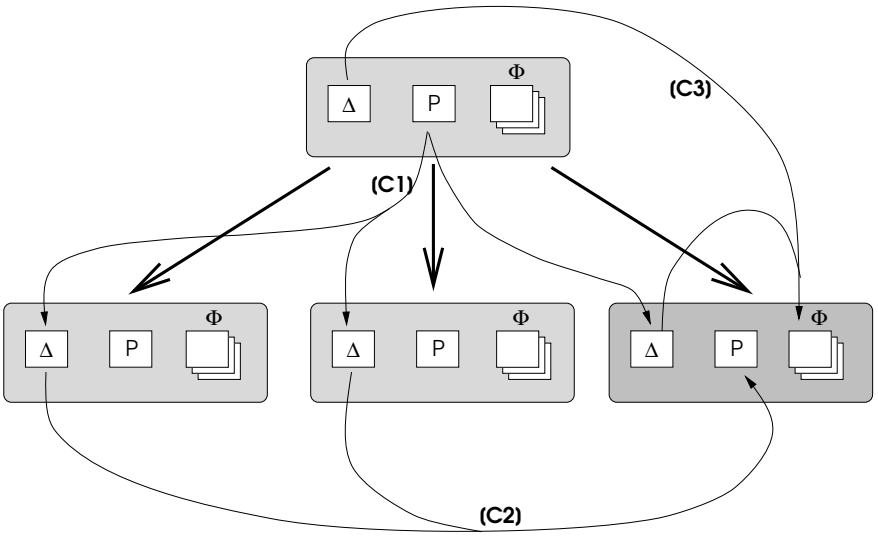


Figure 4.6: Propriétés de l'arbre de recherche des ECM

Théorème 5 (Consistance de la recherche) *Pour chaque nœud N de T , les éléments de $\Phi(N)$ sont des ensembles critiques minimaux.*

Théorème 6 (Systématicité de la recherche) *Les ensembles de conflits de ressources minimaux $\Phi(N)$ associés aux nœuds de T sont tous disjoints.*

Soit T un arbre de recherche d'ECM vérifiant les propriétés [C1 à C4].

Théorème 7 (Complétude de la recherche) *Supposons que T soit entièrement développé (c'est-à-dire que chaque feuille N vérifie $P(N) = \emptyset$) ; alors si U est un ECM, il existe un nœud N de T tel que $U \in \Phi(N)$.*

A noter que les propriétés [C1 à C4] représentent uniquement un certain nombre de contraintes sur le développement de l'arbre de recherche permettant d'assurer la consistance, la systématicité et la complétude de la recherche. Le point principal qui n'est pas précisé dans ces propriétés est la manière de partitionner, dans [C1], l'ensemble $P(N)$ associé à un nœud N donné en un ensemble de cliques $\{\Delta(M)\}_{M \in Enfants(N)}$; puisqu'une fois une partition trouvée, les ensembles $P(M)$ peuvent être calculés à partir de la propriété [C2] et $\Phi(M)$ à partir de [C3]. Nous avons simplement montré que tous les algorithmes de partitionnement de $P(N)$ vérifiant les propriétés [C1 à C4] sont consistants, systématiques et complets. Dans le paragraphe qui suit, nous présentons deux de ces algorithmes.

4.4.2 Procédures de développement d'un nœud

4.4.2.1 Une variante systématique de l'algorithme glouton

Supposons ordonné l'ensemble des propositions d'emprunt de ressources de USE_{attR} : $USE_{attR} = \{u_1, u_2, ..., u_n\}$.

La première solution qui vient à l'esprit pour partitionner $P(N)$ en cliques est de le décomposer en l'ensemble de ses singletons. Ceci conduit à un arbre de recherche complet tel que celui schématisé sur la figure 4.7.

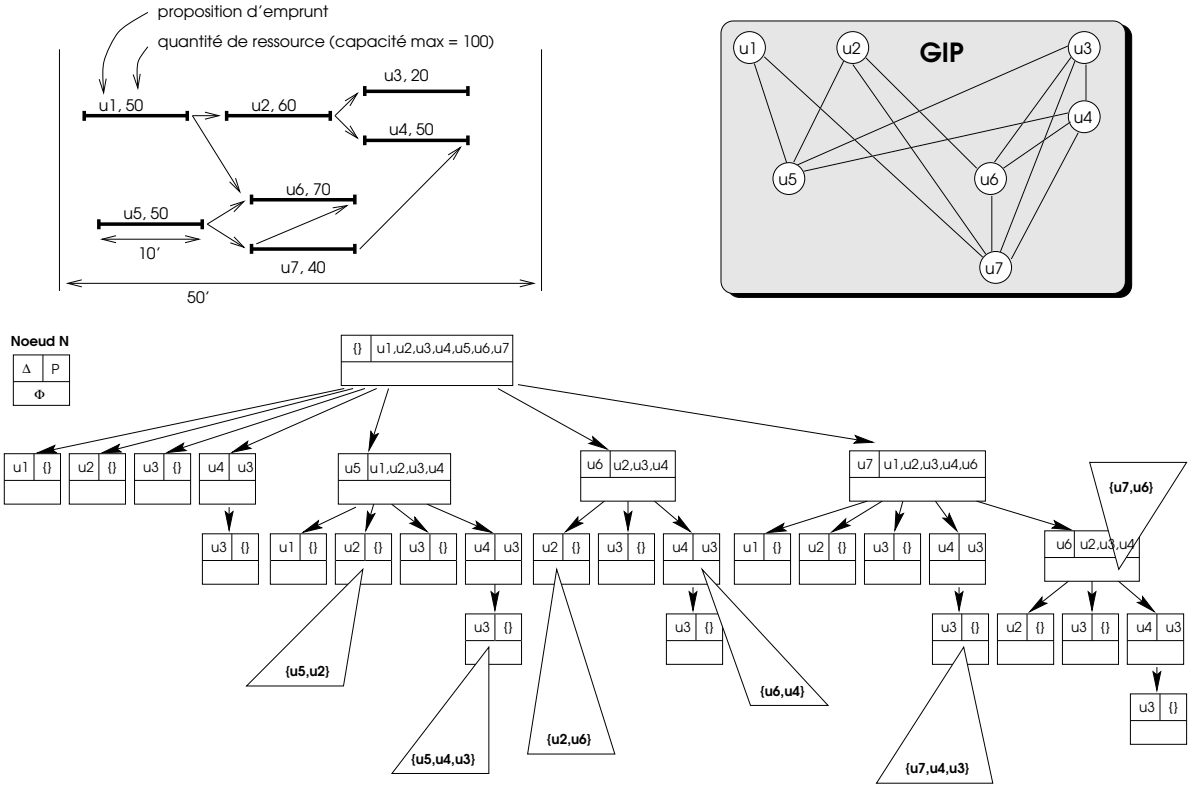


Figure 4.7: Arbre de recherche des ECM : Algorithme glouton

La procédure de développement d'un nœud $N =< \Delta(N), P(N), \Phi(N) >$ est alors la suivante (nous supposerons, pour un nœud N donné et à une renumérotation des propositions d'emprunt de ressources près, que $P(N) = \{u_1, u_2, ..., u_k\}$) :

Algorithme $DEVELOPPEMENT(N)$

1. Créer k enfants $M_1, ..., M_k$ à N

2. Pour $i = 1$ jusqu'à k faire :

2.1. $\Delta(M_i) = \{u_i\}$

2.2. $P(M_i) = \{u_j \mid j < i$ et $\{u_i, u_j\}$ arête du GIP $\}$

2.3. soit $U = \cup_{K \in Ascendants(M_i)} \Delta(K)$

2.3.1. Si U est sur-consommateur minimal alors $\Phi(M_i) = \{U\}$

2.3.2. Sinon $\Phi(M_i) = \emptyset$.

Il est immédiat de vérifier, dans ce contexte-là, que l'algorithme conserve les propriétés [C1,C2,C3] de l'arbre de recherche.

Reste à établir [C4] : si U est une clique de $P(N)$, soit l l'indice maximal des assertions u_i de $P(N)$ et posons $M = M_l$. On a alors $U \cap \Delta(M) = \{u_l\} \neq \emptyset$ et, par choix de l, $U \subset (\Delta(M) \cup P(M))$. D'où le résultat.

Les résultats établis dans le paragraphe précédent montrent donc que la recherche des ECM induite par cet algorithme est *consistante*, *systématique* et *complète*.

Intuitivement, on obtient un algorithme glouton qui consiste à agrandir à chaque nœud N la clique courante $\Pi(N)$ par un sommet unique u du GIP vérifiant la propriété que $\Pi(N) \cup \{u\}$ est une clique (cf. exemple sur la figure 4.7).

4.4.2.2 DECLIC : Vers une contraction de l'espace de recherche

Il y a un double intérêt à augmenter la taille des ensembles $\Delta(N)$ intervenant dans l'arbre de recherche : cela diminue la largeur de l'arbre du fait de la propriété [C1] et sa profondeur puisque, d'après [C2], les ensembles $P(N)$ seront plus contraints et donc, plus petits.

Nous présentons ici une procédure de développement nommée DECLIC (pour "DEcomposition en CLIques" des ensembles $P(N)$) qui, contrairement à la procédure précédente génère des ensembles $\Delta(N)$ de cardinalité supérieure à 1.

L'algorithme DECLIC de développement de tous les fils d'un nœud N est donné ci-dessous ; il repose sur une indexation σ des propositions de $P(N)$ correspondant à l'algorithme du calcul d'une *combinaison parfaite* sur un graphe *triangulé* donné en annexe B.

Un arbre de recherche complet pour le même exemple que celui de la figure 4.7 mais développé par l'algorithme DECLIC est présenté sur la figure 4.8. On notera sur cette figure la diminution de la taille de l'arbre de recherche.

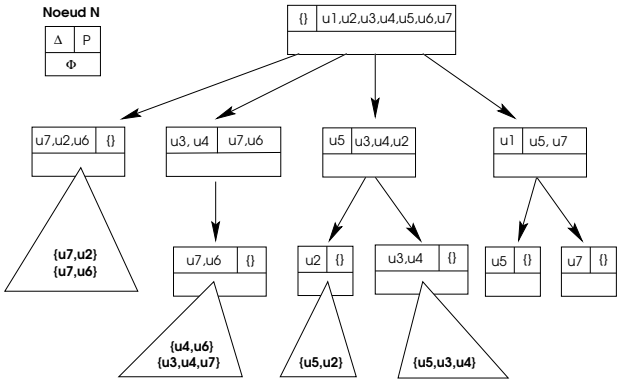


Figure 4.8: Arbre de recherche des ECM : Algorithme DECLIC

Dans l'algorithme, $Adj(u)$ désigne l'ensemble des sommets du GIP adjacents à u par une arête.

Algorithme $DECLIC(N)$

Etape 1 : Construction de σ.
1.1. $U \leftarrow P(N)$
1.2. $V \leftarrow \emptyset$
1.2. $indice\_courant \leftarrow 0$
1.4. tant que $U \neq \emptyset$ faire:
1.4.1. $indice\_courant \leftarrow indice\_courant + 1$
1.4.2. choisir u une assertion de U qui maximise $card(V \cap Adj(u))$
1.4.3. $U \leftarrow U \backslash \{u\}$
1.4.4. $\sigma(u) = indice\_courant$
1.4.5. $V \leftarrow V \cup \{u\}$

Etape 2 : Construction des enfants de N.
(on renommera u_i la proposition d'emprunt u telle que $\sigma(u) = i$ et
on notera $P(N) = \{u_1, u_2, ..., u_k\}$)
2.1. $C(u_1) \leftarrow \{u_1\}$
2.2. $\Delta \leftarrow \{u_1\}$
2.3. $P \leftarrow \emptyset$
2.4. pour $i = 1$ à k faire:
2.4.1. si $i \neq k$: $C(u_{i+1}) \leftarrow \{u_{i+1}\} \cup [Adj(u_{i+1}) \cap \{u_1, u_2, ..., u_i\}]$
2.4.2. sinon, si $i = k$: $C(u_{i+1}) \leftarrow \emptyset$
2.4.3. si $C(u_i) \subset C(u_{i+1})$: $\Delta \leftarrow \Delta \cup \{u_i\}$
2.4.4. sinon:
2.4.4.1 $P \leftarrow C(u_i) \backslash \Delta$
2.4.4.2 création d'un fils M de N tel $\Delta(M) = \Delta$ et $P(M) = P$
2.4.4.3 calcul de $\Phi(M)$ par la propriété $[C4]$
2.4.4.4 $\Delta \leftarrow \{u_{i+1}\}$
2.4.4.5 $P \leftarrow \emptyset$

Le choix de la prochaine proposition d'emprunt à indexer (ligne 1.4.2.) peut s'effectuer selon divers critères ; le critère que nous avons implémenté est de toujours indexer en priorité les propositions d'emprunt les plus "gourmandes" en ressources. Ainsi, sur l'exemple de la figure 4.7, l'indexation associée au nœud racine de l'arbre de recherche des ECM donne : $(u_6, u_2, u_7, u_4, u_3, u_5, u_1)$.

La propriété suivant de l'algorithme est montrée en annexe D :

Propriété 8 *L'algorithme DECLIC conserve les propriétés [C1 à C4] de l'arbre de recherche des ECM. En conséquence, il conduit à une recherche des ECM qui est* **consistante**, **systématique** *et* **complète**.

4.4.3 Exploration de l'arbre de recherche

Selon le type de requête posée au module d'analyse des ressources, un type particulier de parcours de l'arbre de recherche que nous venons de définir sera effectué. Nous décrivons ici

deux types de requêtes selon que l'on cherche à montrer la *nécessaire consistance* de la base de connaissances au niveau des ressources ou selon que l'on recherche des *ECM particuliers* dans celle-ci.

4.4.3.1 Test de consistance nécessaire de la base

Il s'agit ici de montrer s'il existe ou non des ECM sur la base de connaissances. L'idée principale consiste à rechercher dans l'arbre le nœud M qui maximise $q(M) = \Sigma_{u \in \Pi(M)} q(u)$ où $q(u)$ désigne la quantité de ressource empruntée par la proposition u.

Deux cas vont alors se présenter :

- soit $q(M) > Q_0(attR)$ et dans ce cas, nécessairement $\Phi(M) \neq \emptyset$, donc il existe des ECM. En fait, si au cours de la recherche d'un tel nœud M on trouve un nœud N tel que $q(N) > Q_0(attR)$, on est d'ores et déjà sûr qu'il existe des ECM et on peut alors répondre à la requête (inconsistance possible de la base) sans continuer l'exploration de l'arbre ;
- soit $q(M) \leq Q_0(attR)$ et alors, si U était un ECM de la base associé à un nœud N, on aurait nécessairement $\Sigma_{u \in U} q(U) \leq q(N) \leq q(M)$, la dernière inégalité provenant simplement de la définition même de M ; c'est-à-dire $\Sigma_{u \in U} q(U) \leq Q_0(attR)$ ce qui est incohérent avec le fait que U est un ECM. Dans ce cas, il est donc clair qu'il n'existe pas d'ECM sur la ressource $attR$ et donc que la base est nécessairement consistante vis-à-vis de cette ressource.

Pour rechercher un tel nœud M, nous attachons à chaque nœud N de l'arbre de recherche une heuristique $f(N) = g(N) + h(N)$ avec $g(N) = q(N)$ et $h(N) = \Sigma_{u \in P(M)} q(u)$. L'heuristique $f(N)$ majore clairement les quantités $\Sigma_{u \in U} q(U)$ de tous les ECM susceptibles d'être associés à un nœud N' descendant de N ; en effet, si $N \in Ascendants(N')$ et si $U \in \Phi(N')$ alors comme $U \subset \Pi(N') \subset (\Pi(N) \cup P(N))$, on a $\Sigma_{u \in P(M)} q(u) \leq q(N') \leq f(N)$.

Le parcours de l'arbre de recherche consiste alors en un algorithme A^* [Farreny 87] cherchant à trouver un nœud M qui maximise $f(M)$. A noter que jamais, au cours de cette recherche, nous n'avons à expliciter les ensembles $\Phi(N)$ associés au nœud. Pour ce type de requête, on peut donc faire abstraction de la ligne 2.4.4.3 de l'algorithme DECLIC.

A noter que des recherches récentes en théorie des graphes font état d'algorithmes de plus en plus performants pour les problèmes de recherche de cliques particulières sur les graphes faiblement triangulés. Dans [Raghunathan 89] par exemple est décrit un algorithme polynômial - en $O(n^5)$ si n est le nombre de sommets du graphe - de recherche de cliques maximum pondérées ; algorithme qui pourrait être utilisé dans le cas de la vérification de la consistance nécessaire d'un plan partiel. Il suffit d'associer à un noeud donné un poids correspondant à la quantité de ressource empruntée ; le poids d'une clique maximum (au sens de la sommation du poids de chacun de ses sommets) correspond alors à la quantité maximale de

ressource susceptible d'être utilisée à un même instant donné. Le test de consistance revient alors simplement à comparer le poids de la clique maximum à la capacité maximale de la ressource.

4.4.3.2 Recherche d'ECM particuliers dans IXTET

Pour des raisons stratégiques dans le parcours de l'arbre de recherche de planification, il est intéressant, comme nous l'avons vu au chapitre 3 de sélectionner en priorité les défauts du plan partiel pour lesquels l'ensemble des résolvantes est le plus petit possible. Il est clair que plus un ECM fait intervenir de propositions d'emprunt de ressources, plus nombreuses seront ses résolvantes potentielles. Il est donc a priori avantageux de détecter en priorité les ECM les plus petits. C'est à ce type de requêtes que nous nous intéressons dans ce paragraphe.

Notons tout d'abord que si $d(N) = card(Ascendants(N))$ désigne la profondeur d'un nœud N dans l'arbre de recherche des ECM, du fait du dernier point de la propriété [C4] : $\forall U \in \Phi(M)$ avec $N \in Ascendants(M), d(N) \leq card(U)$. En d'autres termes, tous les ECM associés à un nœud descendant de N sont de taille supérieure ou égale à $d(N)$.

On peut donc rechercher les ECM de taille minimale en parcourant l'arbre par une recherche de type *largeur d'abord* et en s'arrêtant au premier nœud N tel que : $\exists M$, nœud déjà exploré, et $\exists U \in \Phi(M) \mid card(U) \leq d(N)$. On est alors sûr que U est un ECM de taille minimale et si $d = card(U)$, l'arbre de recherche n'a pas été développé au-delà de la profondeur d.

Ici encore, une adaptation possible de l'algorithme décrit dans [Raghunathan 89] pourrait être envisagée pour la recherche de telles cliques. Nous n'avons toutefois pas investigué dans cette direction pensant qu'une construction incrémentale des ECM telle qu'elle est effectuée par notre algorithme présente l'avantage de pouvoir fournir rapidement un ECM quelconque, quitte à poursuivre plus en avant la recherche pour obtenir un ECM plus particulier.

4.4.3.3 Complexité de la recherche avec l'algorithme DECLIC

Nous faisons dans cette partie une brève étude de la complexité de la recherche d'un ensemble d'ECM de taille minimale avec l'algorithme DECLIC.

Soit $N =< \Delta(N), P(N), \Phi(N) >$ un nœud ; étudions tout d'abord la complexité de l'application de l'algorithme DECLIC à ce nœud pour construire l'ensemble de ses enfants.

Soit $\delta(N) = card(\Delta(N)), p(N) = card(P(N)), d(N) = card(Ascendants(N))$ la profondeur du nœud N et δ_{max} la taille maximale des ensembles Δ de l'arbre de recherche.

En utilisant le même algorithme et la même représentation que dans [Gavril 72] (voir l'annexe B), la construction de l'indexation σ des éléments de $P(N)$ peut s'effectuer en $O(p(N) + deg(P(N)))$ où $deg(P(N))$ est la largeur du sous-graphe du GIP induit par les sommets de $P(N)$.

Pour ce qui est de la seconde étape de l'algorithme, on voit que l'on fait une première boucle sur chaque élément de $P(N)$ (ligne 2.4), et à l'intérieur de celle ci, les points 2.4.1 à 2.4.4.2 peuvent être effectués en $O(p(N))$ tandis que $\Phi(M)$ peut être calculé en $O(\delta_m^{d(N)})$. Ceci donne une borne maximale en $O(p(N).(p(N)+\delta_m^{d(N)})$ pour cette seconde étape.

La complexité maximale du développement d'un nœud N par DECLIC est donc bornée par $O(p(N)^2 + p(N).\delta_m^{d(N)})$.

Si k est la taille du plus petit ECM et si l'on recherche tous les ECM de taille k, il faudra développer l'arbre de recherche jusqu'à la profondeur $k-1$. Soit pour une profondeur i donnée dans l'arbre de recherche des ECM, $p_m(i)$ la taille maximale des ensembles $P(N)$ pour tous les nœuds N de profondeur i. Etant donné qu'un nœud N peut faire apparaître au maximum $p(N)$ nouveaux enfants, la complexité de la recherche sera donc bornée par $C(k)$ avec :

$$\begin{aligned} C(k) \;=\; & [p_m(0)^2 + p_m(0).\delta_m] \\ & + p_m(0).[p_m(1)^2 + p_m(1).\delta_m^2] \\ & + p_m(0).p_m(1).[p_m(2)^2 + p_m(2).\delta_m^3] + \;\ldots \\ & + p_m(0).p_m(1).....p_m(k-3)[p_m(k-2)^2 + p_m(k-2).\delta_m^{k-1}] \end{aligned}$$

Il est clair que $p_m(i)$ décroît avec i du fait que l'on a la propriété $\forall N \in \mathit{Ascendants}(M)$, $P(M) \subset P(N)$. On peut majorer très grossièrement chacun des $p_m(i), i \geq 1$ par $p_m(1)$. A noter que $p_m(1)$ est lui même majoré par le nombre maximal de propositions d'emprunt de ressources u pouvant intersecter temporellement une autre proposition de ressource u_0 donnée.

On vérifie alors, sachant que $p_m(0) = n$ est le nombre de propositions d'emprunt de $attR$ sur le plan partiel :

$$C(k) \leq n^2 + n.\frac{p_m(1)^k - p_m(1)}{p_m(1) - 1} + n.\frac{p_m(1)^{k-1}.\delta_m^k - \delta_m}{p_m(1).\delta_m - 1}$$

Soit de manière encore plus large, une complexité en pire cas majorée en $O(n^2 + n.[p_m(1).\delta_m]^{k-1})$. Dans tous les problèmes que nous avons codés sur IXTET , les valeurs de k restent de l'ordre de 2 ou 3.

Sur l'exemple de la figure 4.8, on a $n = 7$, $p_m(1) = 3$, et $\delta_m = 3$.

Deux conclusions peuvent être tirées de cette analyse de complexité maximale :

1. Si la taille du plus petit ECM est k, l'ensemble des plus petits ECM peut être calculé avec une complexité polynômiale de degré toujours inférieur à k ;

2. Si au cours de sa construction, le plan partiel croît de manière linéaire plutôt que de manière parallèle (cf. figure 4.9), on peut montrer que les paramètres $p_m(1)$ et δ_m restent constant ; dans ce cas la complexité de la recherche d'ECM croît en $O(n^2)$

avec la taille du plan à analyser. A noter qu'en cours de planification, généralement, le plan croît tout d'abord plutôt de manière parallèle à cause de l'insertion des premières tâches pour expliquer les buts du problème - mais alors n est petit - puis, ensuite, plutôt de manière linéaire pour expliquer récursivement les sous-buts induits par ces tâches.

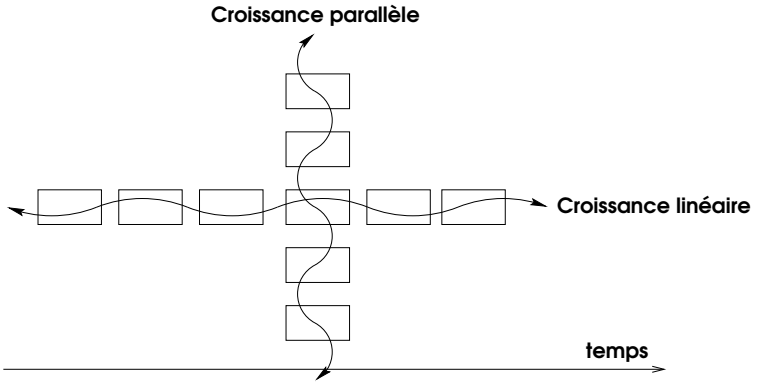


Figure 4.9: Deux types de croissance d'un plan partiel

4.5 Minimisation des résolvantes d'un ECM

Etant donné un ECM U détecté grâce à la procédure que nous venons de décrire, le problème que nous abordons dans ce paragraphe est celui de la constitution d'un ensemble minimal de résolvantes qui soit suffisant pour envisager toutes les manières possibles de résoudre le conflit potentiel. Nous supposons pour l'instant des ressources non-productibles si bien que les seules résolvantes envisagées sont des contraintes d'ordonnancement.

De manière générale, lorsque l'on doit résoudre un conflit et que l'on dispose pour celà d'une disjonction de contraintes possibles, il faut se poser la question de savoir si cette disjonction de contraintes est nécessaire et suffisante pour résoudre le conflit et ce, afin d'éliminer un maximum de redondance au sein de celle-ci. Nous allons dans un premier temps définir de manière générale et indépendamment du contexte "gestion de ressources" ce que l'on entend par le terme de *minimalité d'une contrainte temporelle disjonctive*. Nous appliquerons ensuite nos résultats à un type particulier de contraintes temporelles disjonctives : les contraintes de non-intersection, qui sont celles qui nous intéressent dans le cadre du module de gestion de ressources.

4.5.1 Notion de minimalité d'une contrainte temporelle

Soit $T = \{t\}$ un ensemble d'instants et C_T un ensemble de contraintes temporelles de précédence sur T qui va induire par transitivité une relation d'ordre $<$.

Définition 8 (Contraintes temporelles)
Une **contrainte de précédence (c.p.)** *entre deux instants t et t' sera représentée par $\alpha = (t \prec t')$.*
Une **contrainte temporelle conjonctive (c.t.c.)** *représente un ensemble conjonctif de contraintes de précédence. On notera $\beta = [\alpha_1, \alpha_2, ..., \alpha_p]$.*
Une **contrainte temporelle disjonctive (c.t.d.)** *représente un ensemble disjonctif de contraintes temporelles conjonctives. On notera $\gamma = \{\beta_1, \beta_2, ..., \beta_q\}$.*

Les contraintes α, β, γ décrites ci-dessus ne sont pas (nécessairement) contenues dans C_T ; ceci explique pourquoi nous adoptons la notation $\prec$ dans les contraintes, alors que nous utilisons $<$ pour la relation d'ordre issue de C_T.

A noter que l'ordre partiel courant $<$ associé aux contraintes de C_T peut être vu comme une contrainte temporelle conjonctive que nous noterons $\beta_<$.

On définit de manière naturelle les opérations de conjonction et de disjonction internes à l'ensemble des contraintes disjonctives :

$$\gamma \vee \gamma' = \{\beta_1, \beta_2, ..., \beta_q, \beta'_1, \beta'_2, ..., \beta'_r\}$$

$$\gamma \wedge \gamma' = \{\beta_i \wedge \beta'_j\}_{i \in [1,q]; j \in [1,r]}$$

Définition 9 (Cohérence d'une contrainte temporelle)
Une contrainte conjonctive $\beta = [\alpha_1, \alpha_2, ..., \alpha_p]$ sera dite **cohérente** *ssi elle permet d'induire, par fermeture transitive une relation d'ordre sur les instants sur lesquels elle porte.*
Une contrainte disjonctive $\gamma = \{\beta_1, \beta_2, ..., \beta_q\}$ sera dite **cohérente** *ssi toutes ses constituantes β_i sont cohérentes.*
Une contrainte disjonctive $\gamma = \{\beta_1, \beta_2, ..., \beta_q\}$ sera dite **cohérente par rapport à** $C_T <$ *ssi la contrainte disjonctive $\gamma \wedge \{\beta_<\}$ est cohérente.*

Ainsi, si nous reprenons les contraintes exprimées sur la figure 4.3 entre les intervalles I, J, K et L, nous voyons que :

- $\{[L^+ \prec I^-], [L^+ \prec K^-], [I^+ \prec L^-]\}$ est cohérente par rapport aux contraintes déjà posées ;
- $\{[K^+ \prec I^-], [L^+ \prec K^-]\}$ ne l'est pas.

Nous supposons dans ce qui suit que toutes les contraintes que nous manipulons sont cohérentes par rapport au réseau de contraintes courant $<$.

Définition 10 (Relation de dominance entre contraintes temporelles)
Soient β et β' deux c.t.c. On dira que β' **domine** *β ssi toute c.p. de β est déductible par transitivité des c.p. de β'. On notera $\beta \leftarrow \beta'$.*

Soient γ et γ' deux c.t.d. On dira que γ' **domine** *γ ssi $\forall \beta' \in \gamma', \exists \beta \in \gamma \mid \beta \leftarrow \beta'$. On notera $\gamma \leftarrow \gamma'$.*
Soient γ et γ' deux c.t.d. On dira que γ' **domine** *γ relativement à C_T ssi $\gamma \wedge \{\beta_<\} \leftarrow \gamma' \wedge \{\beta_<\}$. On notera alors $\gamma \overset{<}{\leftarrow} \gamma'$.*

Ainsi, sur l'exemple précédent $\gamma' = \{[L^+ \prec I^-], [L^+ \prec J^-]\}$ domine $\gamma = \{[L^+ \prec K^-], [I^+ \prec L^-], [L^- \prec J^-]\}$ par rapport aux contraintes déjà posées $<$ puisque l'on a : $[L^+ \prec K^-] \overset{<}{\leftarrow} [L^+ \prec I^-]$ et $[L^- \prec J^-] \overset{<}{\leftarrow} [L^+ \prec J^-]$.

Propriété 9 *La relation $\leftarrow$ (ainsi que $\overset{<}{\leftarrow}$) définit un préordre sur l'ensemble des contraintes temporelles disjonctives.*

Définition 11 (Relation d'équivalence entre contraintes temporelles)
Soient γ et γ' deux contraintes disjonctives ; γ et γ' seront dites **équivalentes sur un ordre courant** *$<$ ssi on a $\gamma \overset{<}{\leftarrow} \gamma'$ et $\gamma' \overset{<}{\leftarrow} \gamma$. On notera $\gamma \overset{<}{\longleftrightarrow} \gamma'$.*

On vérifie de manière évidente que, puisque $\overset{<}{\leftarrow}$ est un préordre, $\overset{<}{\longleftrightarrow}$ est une relation d'équivalence sur les contraintes temporelles disjonctives.

Etant donné un réseau de contraintes temporelles induisant un ordre partiel $<$ sur les instants et une contrainte temporelle disjonctive γ, le centre de notre propos est de simplifier au maximum γ afin de trouver une contrainte γ_0, plus élémentaire, qui lui soit équivalente sur $<$. Les définitions suivantes formalisent cette notion de *minimalité d'une contrainte temporelle*. A noter que pour cela, nous introduisons une nouvelle relation d'ordre entre contraintes temporelles qui est l'inclusion ensembliste $\subset$. On pourra remarquer par ailleurs que :

- au niveau des contraintes conjonctives : $(\beta \subset \beta') \Rightarrow (\beta \leftarrow \beta')$;
- au niveau des contraintes disjonctives : $(\gamma \subset \gamma') \Rightarrow (\gamma' \leftarrow \gamma)$.

Définition 12 (Minimalité d'une contrainte temporelle)
Soit β une contrainte temporelle conjonctive, on dira que β est **minimale sur un ordre courant** *$<$ ssi $\forall \beta' \subset \beta$, $\neg(\beta \overset{<}{\longleftrightarrow} \beta')$.*

Soit γ une contrainte temporelle disjonctive, on dira que γ est **minimale sur un ordre courant** *$<$ ssi (1) $\forall \beta \in \gamma$, β est minimale sur $<$ et (2) $\forall \gamma' \subset \gamma$, $\neg(\gamma \overset{<}{\longleftrightarrow} \gamma')$.*

On voit ainsi sur l'exemple que $\gamma = \{[L^+ \prec I^-], [L^+ \prec K^-], [I^+ \prec L^-]\}$ n'est pas minimale sur l'ordre courant puisque l'on a $\gamma \overset{<}{\longleftrightarrow} \{[L^+ \prec K^-], [I^+ \prec L^-]\}$

On peut alors montrer le théorème suivant dont la preuve est reportée en annexe D :

Théorème 8 (Unicité des contraintes minimales) *Soit γ une contrainte temporelle disjonctive. Il existe une* **unique** *contrainte disjonctive γ_0 telle que : (1) γ et γ_0 sont* **équivalentes** *sur $<$ et (2) γ_0 est* **minimale** *sur $<$.*

Nous décrivons aussi dans cette même annexe un algorithme polynômial permettant de calculer γ_0 à partir de n'importe quelle contrainte disjonctive γ.

Dans le paragraphe qui suit, nous appliquons ce résultat d'unicité aux contraintes temporelles qui nous intéressent plus particulièrement dans le cadre de la résolution des ECM : les contraintes de non-intersection.

4.5.2 Application aux contraintes de non-intersection

Supposons trouvé un ECM U ; il s'agit maintenant de définir la contrainte temporelle disjonctive minimale sur le réseau de contraintes courant associée à la résolution de ce conflit.

Nous définissons tout d'abord la contrainte de non-intersection associée à U comme la disjonction de toutes les contraintes conjonctives β qui, si elles étaient insérées sur le réseau courant, suffiraient à lever l'intersection possible entre les intervalles de U. Plus formellement:

Définition 13 (Contrainte de non-intersection)
Soit U un ensemble d'intervalles temporels en intersection possible sur un réseau de contraintes $<$ ($\Diamond \cap_< (U)$). On appellera **contrainte de non-intersection associée à** *U la contrainte temporelle disjonctive $\Gamma(U)$ définie comme la disjonction de toutes les contraintes conjonctives β qui vérifient : (1) β est cohérente sur $<$ et (2) si $\prec$ désigne la nouvelle relation d'ordre définie par $\beta_< \wedge \beta$, alors $\neg\Diamond \cup_\prec (U)$.*

En appliquant le théorème 8 à la contrainte $\Gamma(U)$, on obtient l'existence et l'unicité d'une contrainte minimalement disjonctive $\Gamma_{min}(U)$ qui soit équivalente à $\Gamma(U)$ sur $<$.

Un algorithme bien plus performant que la combinaison des deux algorithmes classiques donnés en annexe D peut être appliqué pour calculer $\Gamma_{min}(U)$:

Algorithme $\Gamma_{min}(<, U)$
Supposons $U = \{u_1, u_2, ..., u_m\}$
1. $\Gamma' = \{[u_i^+, u_j^-] \mid u_j^- \text{ et } u_i^+ \text{ sont non comparables par } <\}$
2. Pour chaque (u_i^+, u_j^+) tel que $u_i^+ < u_j^+$
  2.1. Ôter de Γ' toutes les c.t.c. de la forme $[u_j^+, *]$
3. Pour chaque (u_i^-, u_j^-) tel que $u_i^- < u_j^-$
  3.1. Ôter de Γ' toutes les c.t.c. de la forme $[*, u_i^-]$
4. Retourner Γ'

La contrainte disjonctive Γ' calculée à la ligne 1 est clairement équivalente à $\Gamma(U)$. La partie consistant à minimiser Γ' (lignes 2 et 3) est simplement basée sur le fait que si on a par exemple $u_i^+ < u_j^+$, alors pour tout instant t, $[u_i^+, t] \stackrel{\leq}{\leftarrow} [u_j^+, t]$ et que, d'autre part, si $[u_j^+, t] \in \Gamma'$ alors $[u_i^+, t] \in \Gamma'$.

On peut vérifier que si m est la taille de l'ECM, la complexité de l'algorithme est en $O(m^4)$. En pratique m reste de l'ordre de quelques unités et excède très exceptionnellement 4 ou 5.

Exemple d'application :

Reprenons maintenant l'exemple de la figure 4.7 et supposons que l'on s'intéresse à déterminer les résolvantes de l'ECM $U = \{u_3, u_4, u_7\}$.

En appliquant l'algorithme ci-dessus, on crée d'abord (ligne 1) :

$$\Gamma' = \{[u_3^+, u_4^-], [u_4^+, u_3^-], [u_3^+, u_7^-], [u_7^+, u_3^-], [u_7^+, u_4^-]\}$$

Lors de la minimisation de Γ' à la ligne 2, on s'aperçoit que l'on a sur le réseau de contraintes $u_7^+ < u_4^+$; cela va conduire à éliminer de Γ' toutes les contraintes conjonctives de la forme $[u_4^+, *]$ (c'est-à-dire en fait $[u_4^+, u_3^-]$) puisqu'une telle contrainte, comme elle dominera $[u_7^+, *]$ est inutile et sur-contraignante. On aboutit donc à :

$$\Gamma_{min}(U) = \{[u_3^+, u_4^-], [u_3^+, u_7^-], [u_7^+, u_3^-], [u_7^+, u_4^-]\}$$

4.6 Allocation de ressources partageables

Dans le cas où les ressources gérées $attR$ ne sont pas banalisées mais font aussi apparaître un choix d'allocation (ressources individualisées), ces contraintes d'allocation de ressources (qui se traduisent par des contraintes sur des variables atemporelles) viennent se greffer sur l'analyse des contraintes temporelles telle qu'elle a été étudiée dans le début de ce chapitre.

Rappelons qu'un attribut sur une ressource individualisée est de la forme $attR(?x_1, ..., ?x_n)$ où chaque $attR(X_1, ..., X_n)_{(X_1,...,X_n) \in D_1 \times D_2 \times ... \times D_n}$ est une ressource susceptible d'être utilisée. Nous envisageons dans ce paragraphe l'extension des algorithmes de recherche d'ECM et de simplification des résolvantes décrits précédemment au cas de ressources individualisées identifiées par une seule variable en argument $(attR(?x), ?x \in D_x)$. Le cas général d'une identification de la ressource par un vecteur de n variables peut être traité de manière similaire (avec toutefois des contraintes sur les variables plus complexes) mais est peu utile en pratique.

Un premier problème posé par ces ressources est que, comme les capacités des ressources $attR(X)$,$attR(Y)$, ..., de type $attR$ peuvent être différentes, l'algorithme DECLIC ne pourra pas se baser sur une *capacité identique* pour toutes les ressources de type $attR$, ce qui complexifie lourdement l'algorithme. Une manière aisée de se ramener au cas où toutes les

ressources de même type ont même capacité est de considérer la capacité maximale des ressources de ce type : $Q_0(attR) = max_{X \in D_x}(Q_0(attR(X)))$ et, pour les ressources X ayant une capacité réelle $Q_0(attR(X)) < Q_0(attR)$, de rajouter un emprunt (virtuel) de cette ressource sur l'intervalle $]-\infty, +\infty[$ en quantité $Q_0(attR) - Q_0(attR(X))$. Un exemple d'une telle transformation est donné sur la figure 4.10 pour le cas de deux batteries de capacités différentes déjà évoqué au chapitre 2.

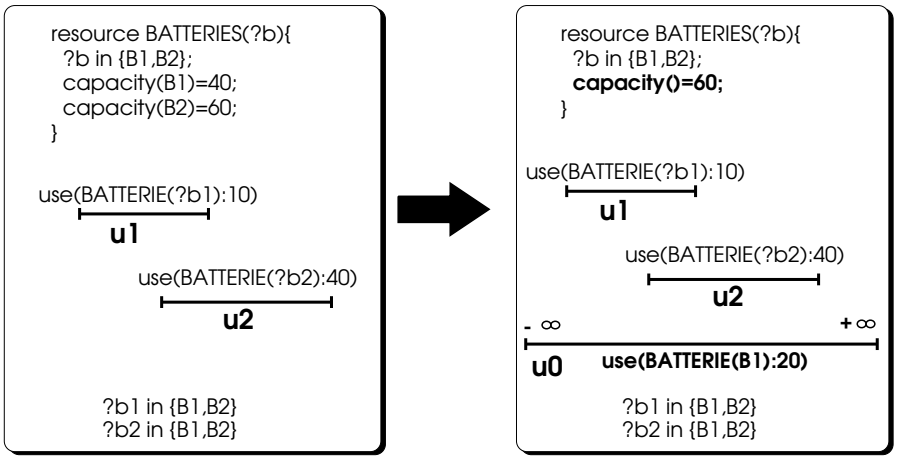


Figure 4.10: Gestion de ressources de même type à capacités différentes

Si nous supposons maintenant que toutes les ressources d'un même type $attR$ ont des capacités maximales identiques $Q_0(attR)$ alors il est possible, simplement en transformant la propriété [C3] définissant les ensembles $\Phi(N)$, de conserver tout le cadre algorithmique que nous avons décrit plus haut.

C3bis: $\Phi(N)$ est l'ensemble des sous-ensembles $U \subset \Pi(N)$ sur-consommateurs minimaux *qui peuvent se voir allouer la même ressource* et qui vérifient : $\forall M \in Ascendants(N), U \cap \Delta(M) \neq \emptyset$.

Vérifier si les propositions d'emprunt d'un ensemble $U = \{use(attR(?x_i) : q_i, (*,*))\}_{i \in [1,n]}$ peuvent se voir allouer la même ressource revient à tester si la contrainte $?x_1 = ?x_2 = ... = ?x_n$ est consistante avec le réseau C_V des contraintes sur les variables atemporelles. Comme le gestionnaire permet la représentation de contraintes de différence ($\neq$) et de dépendance ($\Rightarrow$) entre variables, nous savons que répondre à ce type de requête est un problème NP-complet.

Comme pour les requêtes sur une simple possibilité d'unification entre deux variables $?x_1 = ?x_2$, nous nous contentons, pour ce type de requête d'une réponse qui sera peut être un peu trop pessimiste (au sens où l'existence de conflits pourra être détectée alors qu'il n'y a pas lieu) en testant uniquement (1) les domaines des variables après propagation du réseau et (2) les contraintes d'inégalité qui ont été directement exprimées entre deux variables $?x_i$ et $?x_j$. A noter toutefois que, si cette "approximation" de la requête peut affecter les performances du système, elle ne remet en cause ni la consistance, ni la complétude globale de la recherche.

Plus formellement, nous répondons par l'affirmative à la requête si et seulement si:

- $\mathcal{D}_{e(?x_1)} \cap \mathcal{D}_{e(?x_2)} \cap ... \cap \mathcal{D}_{e(?x_n)} \neq \emptyset$ et;
- $\forall i, j \in [1, n]$, aucune contrainte $?x_i \neq ?x_j$ n'a été *exprimée*.

Le calcul des résolvantes associées à un ECM s'effectue alors en ajoutant, en plus de toutes les résolvantes d'ordonnancement discutées au paragraphe 4.5, toutes les contraintes de différence entre deux quelconques des variables $?x_i$ et $?x_j$ intervenant comme des ressources possiblement en conflit.

Ainsi, sur l'exemple du plan partiel de la figure 4.10, un seul ECM est détecté : $U = \{u_0, u_1, u_2\}$ et l'ensemble de ses résolvantes est:
$résolvantes(U) = \{[u_1^+ < u_2^-], [u_2^+ < u_1^-], [?b_1 \neq B1], [?b_2 \neq B2], [?b_1 \neq ?b_2]\}$.

A noter que les contraintes d'allocation sont, à l'image des contraintes temporelles d'ordonnancement, susceptibles d'être minimisées en examinant les contraintes déjà posées sur le plan partiel. Le principe restant le même : on peut ôter de $résolvantes(U)$ toutes les contraintes $?x_i \neq ?x_j$ telles que sur C_V, l'insertion de $?x_i \neq ?x_j$ entraînerait la vérification de $?x_k \neq ?x_l$, une autre résolvante appartenant à $resolvantes(U)$. Ceci peut par exemple être le cas lorsque C_V contient les contraintes d'égalités $?x_i = ?x_k$ et $?x_j = ?x_l$.

Un cas particulier de ressources individualisées est celui des ressources non-partageables c'est-à-dire pour lesquelles chacune des ressources du même type est de capacité unitaire ; on peut par exemple penser à une flottille de robots, chacun ne pouvant être alloué qu'à une tâche à un instant donné. Il est clair que dans ce cas, tous les ECM sont des paires $\{u, u'\}$ de propositions d'emprunt de la ressource. Vu l'extrême importance en pratique de ce type de ressources, nous avons codé dans IxTeT un algorithme spécifique de complexité quadratique pour détecter et proposer des résolvantes aux ECM sur ce type de ressources. L'algorithme est très similaire à celui gérant les menaces (cf. §3.8) ; nous ne le détaillerons pas ici.

4.7 Cas des ressources productibles

Dans le cas de ressources productibles, il faut aussi envisager, lors du calcul des résolvantes pour un ECM, l'insertion d'une ou plusieurs tâches productrices de ressources.

Pour des raisons de clarté de l'exposé, nous supposons que chacun des modèles de tâches susceptible de produire une ressource ne contient qu'un seul instant de production de cette ressource. Si ce n'est pas le cas, il faut alors raisonner plus localement au niveau des instants où la ressource est produite ; ce qui n'est pas plus complexe mais surchargerait les notations.

Etant donné un ECM $\Phi = \{r_1, ..., r_k\} \in ECM(\mathcal{P})$ avec $r_i = use(attR(?x_i) : q_i, (t_i^-, t_i^+))$, la quantité de ressource devant nécessairement être produite pour résoudre le conflit est :

$$SURCONS(\Phi) = \sum_{r \in \Phi} q(r) - Q_0(attR)$$

Si d'autre part $RES(\Phi)$ désigne l'ensemble $\{attR(X)\}$ des ressources possiblement conflictuelles dans Φ (cf. définition 3) :

$$RES(\Phi) = \{attR(X) \mid X \in \mathcal{D}_{e(?x_1)} \cap \mathcal{D}_{e(?x_2)} \cap ... \cap \mathcal{D}_{e(?x_k)}\}$$

Alors, sont envisagées toutes les combinaisons minimales $\mathcal{E}$ de tâches susceptibles de produire une ressource de $RES(\Phi)$ en quantité supérieure ou égale à $SURCONS(\Phi)$. Nous noterons $Combinaisons(AttR(X), SURCONS(\Phi))$ l'ensemble de ces combinaisons minimales. Ces combinaisons $\mathcal{E}$ sont des multi-ensembles de tâches dans la mesure où l'insertion de plusieurs instances d'une même tâche peut être nécessaire pour produire la ressource en quantité suffisante. Toutes les tâches de $\mathcal{E}$ doivent être insérées avant un des instant t_i^-.

Remarquons que si on a une contrainte de précédence $t_j^- < t_i^-$ sur le plan partiel analysé, il serait inutile d'envisager d'insérer des tâches de production avant t_j^- puisqu'il est suffisant de les insérer avant t_i^- et que cette dernière possibilité est déjà envisagée parmi les autres résolvantes. Cette minimalisation des résolvantes de production de ressources est bien entendu le pendant des autres minimisations que nous avons évoquées plus haut pour les résolvantes d'ordonnancement et de contraintes d'allocation de ressources.

Finalement, les résolvantes liées à la production de ressources par l'insertion de nouvelles tâches dans le plan sont les suivantes :

$$production(\Phi) = \bigcup_{t_i^- | (\nexists t_j^- | t_i^- < t_j^-)} production\_avant(\Phi, t_i^-)$$

$$production\_avant(\Phi, t_i^-) = \bigcup_{attR(X) \in RES(\Phi)} production\_instance\_avant(AttR(X), \Phi, t_i^-)$$

$$production\_instance\_avant(AttR(X), \Phi, t_i^-) = \bigcup_{\mathcal{E} \in Combinaisons(AttR(X), SURCONS(\Phi))} \{\rho_{prod}(\mathcal{E}, t_i^-)\}$$

La résolvante liée à l'insertion d'un multi-ensemble de tâches de production de ressource $\mathcal{E} = \{\mathcal{T}\}$ avant l'instant t_i^- est alors :

$$\rho_{prod}(\mathcal{E}, t_i^-) = [(\mathcal{T})_{\mathcal{T} \in \mathcal{E}} \ , \ ((t < t_i^-), (?x = X))_{produce(attR(?x):q,t) \in \mathcal{T} \in \mathcal{E}}]$$

Il est à remarquer le fait que l'insertion de tâches de production de ressources à un nœud donné de l'arbre de recherche global peut très bien lever des conflits de ressources qui ont déjà été résolus plus haut dans le processus de planification. Ceci a pour effet de conserver dans le plan certaines contraintes qui ne sont plus nécessaires pour assurer sa consistance et peut donc conduire à des plans solution surcontraints (cf. exemple sur la figure 4.11).

Ce phénomène, bien que ne remettant en cause, ni la consistance, ni la complétude de la recherche peut toutefois être gênant quant aux performances de la planification puisqu'il risque de conduire à des retours-arrière non nécessaires. Une solution (que nous n'avons pas implémentée à ce jour) consisterait, chaque fois qu'à un nœud de l'arbre de recherche un

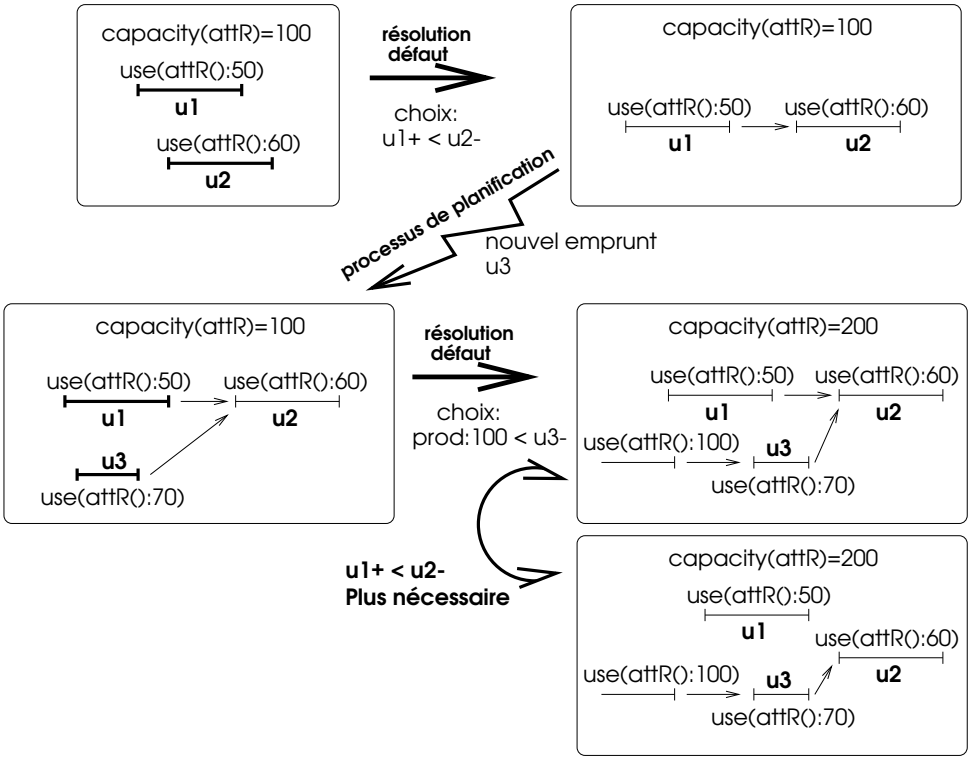


Figure 4.11: Un plan partiel surcontraint suite à une production de ressources

ECM est résolu par l'insertion de tâches de production de ressources, à réexaminer tous les ECM sur cette ressource qui ont été résolus plus haut dans l'arbre de recherche et à reconstruire le plan partiel courant en évitant de rajouter les résolvantes non nécessaires du fait des nouvelles tâches de production.

4.8 Bilan et conclusion

L'architecture du module d'analyse des ressources est résumée sur la figure 4.12.

Dans notre implémentation, nous ne recherchons pas un seul ECM puisque le module d'analyse des ressources est appelé pour réalimenter complétement l'agenda des défauts ; nous ne recherchons pas non plus l'ensemble des ECM car la taille de cet ensemble évolue de façon exponentielle avec la taille du plan. Nous avons effectué un compromis en recherchant un ensemble d'ECM de *taille minimale* et de cardinalité n'excédant pas *nb* qui est un paramètre de contrôle de notre algorithme de planification.

La courbe 4.13 illustre les performances en moyenne du module d'analyse de ressources sur des problèmes générés aléatoirement. Il est clair que pour un problème de taille donnée la densité des contraintes temporelles qui portent sur celui-ci est un paramètre déterminant pour les performances de l'analyse des ressources. Ainsi, sur un plan très peu contraint, il existe en général un très grand nombre d'ECM et la recherche d'un sous-ensemble de ceux-ci est en général assez facile. A l'opposé, lorsque la densité des contraintes est élevée,

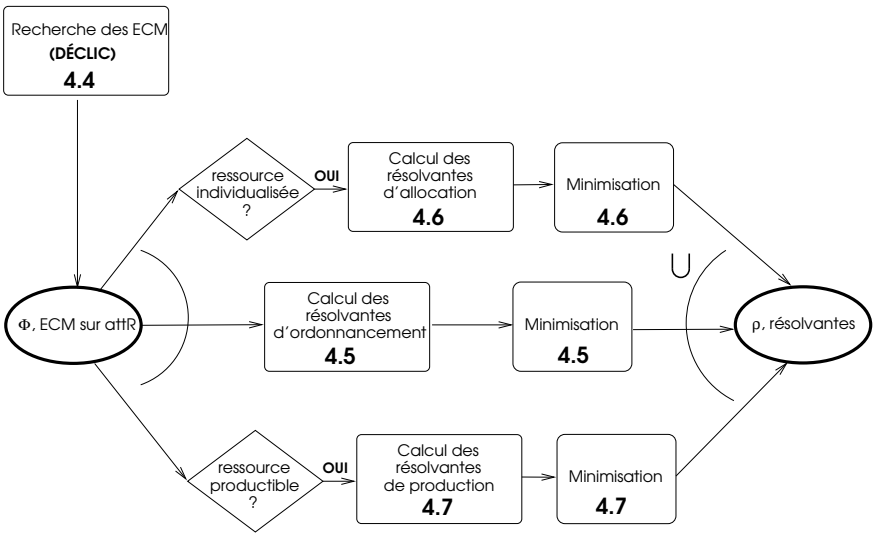


Figure 4.12: Architecture du module d'analyse des ressources et paragraphes concernés

elle impose, à la limite, un ordre total sur les instants et dans ce cas on peut montrer que le GIP possède une structure de graphe d'intervalles (cf. annexe B) et que notre recherche d'ECM est strictement équivalente à l'algorithme de recherche de cliques maximales sur un graphe d'intervalles. La profondeur maximale de l'arbre de recherche est alors de deux et les performances de l'algorithme DECLIC excellentes.

Chaque point de la courbe représente une moyenne sur 50 échantillons de problèmes de la taille donnée et pour la densité de contraintes temporelles la plus défavorable vis-à-vis de notre algorithme (près du point de transition de phase entre problèmes sous-contraints et problèmes sur-contraints). On notera principalement le comportement quasi-linéaire en fonction de la taille du problème et le fait que les performances se dégradent peu selon que l'on recherche un seul ou plusieurs ECM de taille minimale.

Les performances du module d'analyse des ressources sur des problèmes réels se sont montrées suffisamment intéressantes pour que ce module ne constitue pas le point critique de notre approche globale ; ainsi, le temps passé à l'analyse des ressources est en général du même ordre de grandeur que celui passé à l'analyse des menaces pour lequel on dispose pourtant d'un algorithme de complexité quadratique. Ces performances sont liées au fait que la profondeur de l'arbre de recherche des ECM reste, en pratique, très faible.

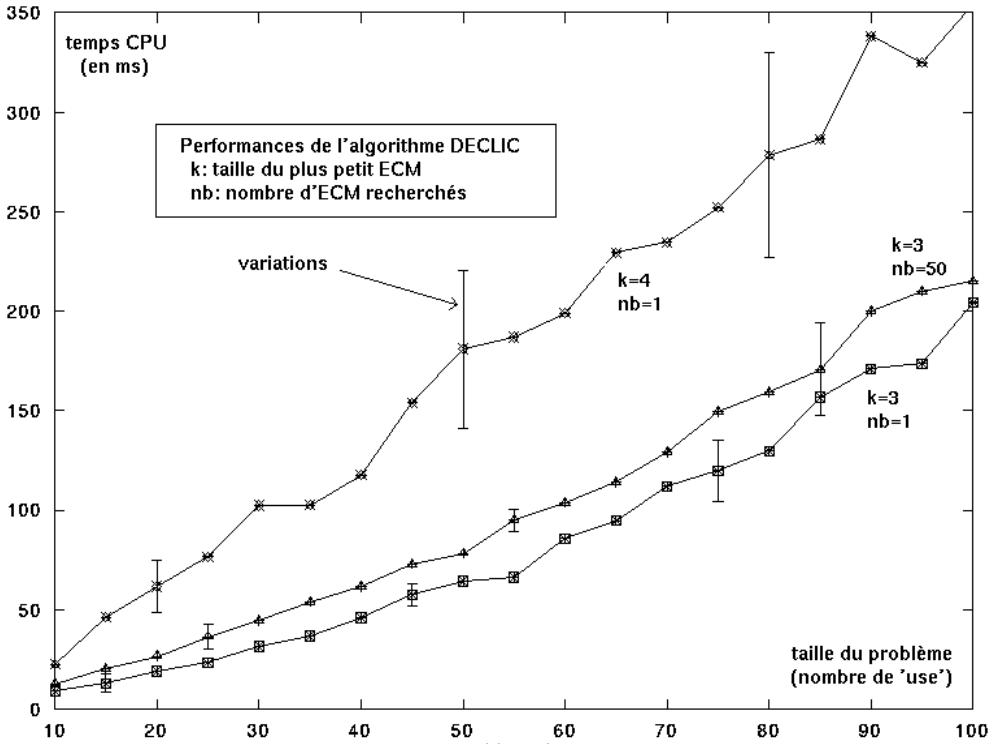


Figure 4.13: Performances de l'analyse des ressources

Chapitre 5

Hiérarchisation de la recherche

5.1 Introduction

L'utilisation de techniques de hiérarchisation se révèle en général très fructueuse lorsqu'il s'agit de gérer des problèmes de grande taille.

De manière générale, la hiérarchisation de la recherche consiste à d'abord résoudre le problème à un niveau d'abstraction élevé puis à affiner progressivement la solution dans des niveaux moins abstraits jusqu'à aboutir, en final, à une solution concrète au problème.

Dans les approches hiérarchiques classiques de la planification [Sacerdoti 74] [Yang 90] [Knoblock 94], la hiérarchie doit être complètement définie hors-ligne, soit par l'utilisateur, soit automatiquement à partir d'une analyse syntaxique des opérateurs de planification. Au cours du processus de planification, le contrôle de la recherche n'a alors plus aucun moyen d'action sur cette hiérarchie (notion de *hiérarchie statique*). L'originalité de l'approche hiérarchique développée sur IXTET tient en particulier dans le fait que nous ne nous engageons pas à une hiérarchie complètement spécifiée hors-ligne et imposée au contrôle. A un domaine donné, nous n'associons pas une seule hiérarchie mais un jeu de contraintes permettant de caractériser un *ensemble de hiérarchies admissibles* vis-à-vis d'un critère donné. Le choix final de la hiérarchie effectivement utilisée est alors contrôlé par le processus de planification lui-même (notion de *hiérarchie dynamique*).

D'autre part la hiérarchisation utilisée dans IXTET , du fait de l'intégration entre synthèse de plans et gestion de ressources, permet d'englober et d'entrelacer de manière naturelle ces deux aspects tout au long de la chaîne allant du niveau le plus abstrait au niveau le plus concret.

Ce chapitre décrit notre approche, ses extensions possibles et un certain nombre de résultats empiriques montrant son intérêt.

5.2 Hiérarchies d'abstraction

Le problème de la sélection du sous-but à établir ou, de manière plus générale, du prochain défaut du plan partiel à résoudre est une composante de la stratégie de l'algorithmique de contrôle qui doit être définie pour chaque système de planification. Récemment, de nombreux travaux ont été effectués visant à formaliser et gérer des *hiérarchies d'abstraction* en planification linéaire aussi bien que non-linéaire, avec pour objectif de définir une procédure intelligente de sélection des défauts [Yang 90, Knoblock 91b, Knoblock 94]. Plusieurs résultats expérimentaux ont montré le fait que l'utilisation d'une hiérarchie d'abstraction pouvait permettre d'augmenter de manière substancielle les performances de la planification [Wilkins 88, Knoblock 94].

Le principe des hiérarchies d'abstraction utilisées en planification, initialement dû à Sacerdoti dans le cadre du planificateur ABSTRIPS [Sacerdoti 74] et réintroduit plus tard dans des systèmes comme ABTWEAK [Yang 90] ou PRODIGY [Knoblock 94] est très simple. Un entier naturel, souvent appelé *niveau d'abstraction* ou *criticité* est associé à chaque motif de sous-but potentiel du domaine de planification considéré. Généralement, en partant du niveau de criticité le plus élevé, le problème de planification est résolu à chaque niveau i en ne considérant à ce niveau-là que les sous-buts ayant une criticité supérieure ou égale à i. Cette approche fait apparaître deux types de problématiques : (1) comment associer des niveaux d'abstraction à chaque sous-but potentiel et (2) comment gérer ces niveaux dans la procédure de recherche d'un plan solution.

Tous les résultats sur les hiérarchies d'abstraction ont jusqu'alors été obtenus dans le cadre d'une représentation de type STRIPS. La version du formalisme STRIPS que nous adoptons dans ce chapitre est, pour des raisons de clarté un peu moins détaillée que celle utilisée en §2.7. Nous supposons qu'une action est représentée par un opérateur α comme une paire ($pre(\alpha)$,$effect(\alpha)$) où $pre(\alpha)$ et $effect(\alpha)$ sont des listes de littéraux, c'est-à-dire des formules atomiques (positives ou négatives) de la logique du premier ordre décrivant respectivement les préconditions et les effets de chaque action. Les sous-buts sont alors représentés par des littéraux auxquels, dans le contexte d'une hiérarchie d'abstraction, seront associés des niveaux.

Etant donné un domaine de planification et un problème à résoudre, plusieurs hiérarchies d'abstraction peuvent être envisagées. Afin de caractériser ces hiérarchies, divers critères peuvent être définis. Les deux critères les plus étudiés sont respectivement la propriété de *raffinement descendant (Downward Refinement Property ou DRP)* [Bacchus 91, Bacchus 94] et la propriété de *monotonicité ordonnée (Ordered Monotonicity Property ou OMP)* [Knoblock 90] décrites informellement ci-dessous.

Dans ces définitions, un plan abstrait est un plan solution du problème de planification à un niveau d'abstraction i donné. Un plan P_{i-1} est un *raffinement* d'un plan abstrait P_i si P_{i-1} est un plan abstrait qui ne diffère de P_i que par l'addition d'opérateurs et/ou de contraintes ayant été insérées pour satisfaire des préconditions introduites de niveau $i-1$.

Propriété 10
Raffinement descendant (DRP): *S'il existe une solution au problème global, chaque plan abstrait peut être raffiné en une solution.*
Monotonicité ordonnée (OMP): *Pour chaque plan abstrait, aucun raffinement de ce plan n'affectera les littéraux déjà établis dans le plan abstrait.*

Il est clair que la DRP est la propriété la plus intéressante qu'une hiérarchie puisse satisfaire. Cette propriété réduit le coût de la recherche de manière exponentielle [Knoblock 91a]. Néanmoins, il s'agit d'un critère très fort et donc, qui ne peut être garanti que dans des cas très particuliers. L'OMP est plus réaliste et donc, plus intéressante pour des problèmes généraux qui se posent en pratique. Lorsque cette propriété est vérifiée, beaucoup de sources de retour-arrière sont éliminées puisqu'aucune interaction directe entre les niveaux d'abstraction n'est possible. Dans ce cas, l'efficacité de la recherche peut être grandement améliorée. De plus, il a été montré que de telles hiérarchies d'abstraction vérifiant l'OMP pouvaient facilement être générées automatiquement, ce qui rend cette propriété d'autant plus attractive [Knoblock 94].

Pour construire automatiquement une hiérarchie d'abstraction, Knoblock a proposé différentes conditions suffisantes permettant de garantir l'OMP. Ces conditions correspondent à un ensemble de contraintes sur le niveau des littéraux du domaine ; contraintes qui peuvent être directement extraites de la description syntaxique des opérateurs et du problème :

Propriété 11 (Conditions suffisantes pour l'OMP)
Soit O l'ensemble des opérateurs du domaine. $\forall \alpha \in O$, $\forall p \in pre(\alpha)$ *et* $\forall q, q' \in effect(\alpha)$,

1. $niveau(q') = niveau(q)$
2. $niveau(p) \leq niveau(q)$

Le principe de ces conditions est simple : elles imposent que chaque littéral qui pourrait être modifié par le processus de résolution d'un autre littéral ait un niveau d'abstraction inférieur ou égal à ce dernier. Ceci assure que tout opérateur inséré pour satisfaire, directement ou indirectement un littéral, laissera invariants les littéraux de plus haut niveau d'abstraction.

A partir de l'ensemble des conditions suffisantes obtenues en analysant le domaine, il est aisé de générer une hiérarchie d'abstraction vérifiant l'OMP. Pour cela, il suffit de trouver une valuation des littéraux qui satisfasse l'ensemble des contraintes. La solution proposée dans ALPINE [Knoblock 94] consiste à construire un graphe orienté dont les nœuds représentent des littéraux (ou des ensembles de littéraux de même niveau) et les arcs, les contraintes de niveau hiérarchique entre les littéraux représentés dans les nœuds. Un tri topologique du graphe est alors effectué qui définit un ordre total sur les littéraux ; ordre total conduisant à une hiérarchie d'abstraction qui vérifie nécessairement l'OMP. Le choix de l'ordre total

construit à partir du graphe est crucial relativement à l'efficacité de la planification. Dans [Bäckström 95b], les auteurs ont montré qu'un mauvais choix pouvait conduire non pas à une amélioration des performances mais bien à une détérioration exponentielle de celles-ci ; nous verrons plus loin comment l'approche que nous proposons aide à résoudre ce problème en ne s'engageant pas au choix d'un ordre total [Garcia 95].

Il peut être avantageux de prendre aussi en compte le problème à résoudre lors de la génération de la hiérarchie d'abstraction. L'intérêt étant que les hiérarchies générées seront moins surcontraintes puisque les contraintes entre niveaux d'abstraction qui ne sont pas relevantes vis-à-vis du problème ne seront pas prises en compte. La solution adoptée par Knoblock consiste à ne considérer comme effets possibles d'une action, lors de la génération automatique des contraintes sur les niveaux, que ceux qui sont *relevants* pour la résolution des buts du problème.

Une autre extension possible revient à faire la distinction entre *effets principaux* et *effets secondaires* d'un opérateur. Classiquement, les effets principaux spécifient la justification réelle de la présence d'une action dans un plan tandis que les effets secondaires sont des effets de bord de l'action. Durant le processus de planification, les opérateurs sont sélectionnés pour satisfaire un sous-but uniquement sur la base de leurs effets principaux ; ceci permet une nette amélioration des performances suite à une réduction du facteur de branchement dans l'arbre de recherche [Fink 95b]. Dans les conditions suffisantes pour vérifier l'OMP, seuls les effets primaires nécessitent alors d'être pris en compte. Ici encore, l'intérêt est d'obtenir un réseau plus souple de contraintes sur les niveaux d'abstraction.

La dernière extension est liée à l'utilisation de schémas d'opérateurs. En effet, la majorité des systèmes de planification utilise des shémas d'opérateurs plutôt que des opérateurs totalement instanciés. La solution proposée dans ALPINE est d'associer un *type* à chacune des constantes pouvant apparaître dans les arguments d'un littéral et de construire, à partir de cette typologie des constantes (ou objets) du domaine, une typologie des littéraux instanciés. Les niveaux d'abstraction sont alors associés aux différents types de littéraux plutôt qu'aux littéraux eux-mêmes.

5.3 La hiérarchie de moindre engagement d' IxTeT

Pour les raisons décrites ci-dessus, l'utilisation d'une hiérarchie d'abstraction pour contrôler la recherche au sein du planificateur IxTeT est très intéressante. Dans la suite de ce chapitre, nous proposons : (1) une extension au formalisme IxTeT du paradigme de planification avec hiérarchie d'abstraction et (2) un nouveau principe de génération automatique et de gestion de la hiérarchie qui étend au paradigme de hiérarchie d'abstraction l'approche de moindre engagement.

5.3.1 Hiérarchie d'abstraction pour IXTET

Nous avons vu dans le chapitre 3 qu'une composante essentielle du contrôle de la recherche consiste à sélectionner le prochain défaut à résoudre dans le plan partiel courant.

Pour ce défaut, un ensemble minimal de résolvantes est calculé à partir duquel sont créés les nœuds successeurs dans l'arbre de recherche. Ainsi, la généralisation des travaux décrits plus haut sur le formalisme STRIPS nous conduit à associer un niveau d'abstraction à chacun des différents défauts du plan. Dans IXTET , les défauts sont de trois types :

- sous-but pendant : par exemple, $hold(POSITION(?robot) : SalleA, (t1, t2))$;
- menace : entre $event(ETAT\_ROBOT(robot1) : (VIDE, CHARGE), t)$ et $hold(ETAT\_ROBOT(robot1) : CHARGE, (t1, t2))$ par exemple ; ou
- conflit de ressource, ainsi un ECM composé de $use(BATTERIES(B2) : 60, (t1, t2))$ et $use(BATTERIES(B2) : 70, (t3, t4))$.

Pour chaque instance de défaut, un seul *nom d'attribut* est présent (ici : $POSITION$, $ETAT\_ROBOT$ ou $BATTERIES$). Suite à cette observation et au fait que cela complexifie peu le mécanisme de contrôle, nous avons choisi, dans une première approche, de caractériser le niveau d'abstraction d'un défaut par le *nom de l'attribut* sur lequel il porte.

Hypothèse 1 (Un unique niveau par nom d'attribut) *: Un niveau d'abstraction représente un ensemble de défauts. Deux défauts portant sur le même nom d'attribut ont nécessairement même niveau d'abstraction.*

Au niveau du contrôle, la gestion d'une hiérarchie d'abstraction consiste donc à résoudre les défauts dans les différents niveaux d'abstraction, partant des défauts les plus abstraits vers les défauts les moins abstraits. Nous avons choisi de définir des hiérarchies vérifiant une propriété similaire à l'OMP étendue à notre formalisme :

Propriété 12 (Monotonicité ordonnée pour IXTET) *: Pour chaque plan partiel courant possible, aucun raffinement envisageable de ce plan partiel suite à la résolution de défauts ne fera apparaître de nouveau défaut qui soit d'un niveau strictement supérieur au niveau d'abstraction courant.*

Planifier dans le contexte d'une telle hiérarchie signifie que les seules causes de retour-arrière à partir d'un plan partiel sont l'absence de résolvante pour un défaut du niveau d'abstraction courant ou l'inconsistance des réseaux de contraintes sur les variables (temporelles ou atemporelles). L'efficacité d'une telle approche a été empiriquement prouvée dans de nombreux domaines.

Malgré l'extrême simplicité de ce critère, différents types de hiérarchies qui le vérifient sont possibles. Tous les systèmes de planification hiérarchiques (ABSTRIPS [Sacerdoti 74], ABTWEAK [Yang 90], PRODIGY[Knoblock 94]) imposent un ordre total pré-établi sur les niveaux d'abstraction ; plus précisément, la hiérarchie qu'ils utilisent est une partition calculée hors-ligne de l'ensemble des défauts possibles d'un plan partiel, chaque sous-ensemble de cette partition correspondant à un niveau. Dans ces approches, une telle décomposition s'effectue dans un premier temps en recherchant un ordre partiel sur les niveaux satisfaisant l'OMP (cf. propriété 11) puis, dans un deuxième temps, par la linéarisation de cet ordre partiel en fonction d'une stratégie définie hors-ligne [Knoblock 94, Bacchus 94]. Cette approche soulève un important problème : comme cette stratégie est statique et décorrélée du processus de planification, elle conduit à un engagement a priori qui peut être préjudiciable aux performances de l'algorithme de planification et résulter en l'augmentation des risques de retours-arrière entre niveaux d'abstraction. Ce risque est clairement exprimé sur la base d'un exemple dans [Bäckström 95b].

C'est pourquoi nous proposons une solution originale : dans IXTET , la hiérarchie ne sera totalement définie qu'au cours du processus de planification. Les différents niveaux d'abstraction sont construits en ligne en fonction de l'évolution de la recherche. Si L représente le niveau d'abstraction courant, c'est-à-dire l'ensemble des défauts qui sont couramment analysés sur ce plan partiel, l'idée principale est de ne pas attendre que tous les défauts de L aient été résolus avant de faire évoluer le niveau d'abstraction de L à L' comme cela est fait dans les approches classiques.

5.3.2 Une approche de moindre engagement

La question qui se pose maintenant est de savoir comment construire automatiquement une telle hiérarchie pour IXTET . A la manière d'ALPINE, notre approche repose sur la définition de conditions suffisantes extraites de la description syntaxique des tâches du domaine. Ces conditions s'expriment par des contraintes sur les niveaux d'abstraction et les hiérarchies d'abstraction possibles. Elles sont représentées par une relation d'ordre partiel sur les différents noms d'attributs.

Dans la définition qui suit, nous supposons que toutes les sous-tâches d'une tâche donnée ont déjà été récursivement développées si bien qu'une tâche ne contient que des ensembles d'événements, d'assertions, d'utilisations de ressources et des contraintes d'instantiations sur les variables, temporelles ou non, qu'elle contient. Nous notons att_p le nom de l'attribut sur lequel porte une proposition temporelle p (qu'il s'agisse d'un événement, d'une assertion ou d'une utilisation de ressource) et att_Φ le nom de l'attribut sur lequel porte un défaut quelconque Φ.

Définition 14 (Contraintes $\prec$ sur les noms d'attributs) *: Pour chaque tâche T du domaine, si e est un événement (event) ou une proposition de production de ressource*

(produce)[1] *et si p est une proposition temporelle quelconque de T, alors* $att_p \dot{\prec} att_e$. *Nous noterons* $\prec$ *la fermeture transitive de* $\dot{\prec}$.

La présence de la contrainte $att \prec att'$ interdit au cours du processus de planification, de résoudre un défaut sur un nom d'attribut att dans un niveau plus abstrait (i.e. avant) qu'un défaut portant sur un attribut att'. En d'autres termes, pour pouvoir résoudre un défaut sur att, il sera nécessaire d'attendre que tous les défauts sur att' aient été résolus. A partir de l'ensemble des contraintes $\prec$, on peut définir une relation $<$ entre ensembles de noms d'attributs qui sont $\prec$-équivalents (nous identifions ici $<$ et sa fermeture transitive) :

Définition 15 ($<$) *: Soit* $\widehat{att} = \{att' \mid att \prec att' \wedge att' \prec att\}$. *Alors* $\widehat{att} < \widehat{att}'$ *si et seulement si* $\widehat{att} \neq \widehat{att}'$ *et* $att \prec att'$.

Cet ordre partiel $<$ peut être représenté par un graphe de contraintes orienté et acyclique $\mathcal{G}$ dont les nœuds sont des ensembles de noms d'attributs $\prec$-équivalents et les arcs représentent les contraintes de $<$ sur les niveaux d'abstractions.

Pour illustrer ces définitions, considérons la tâche DEPOSER_BRIQUES décrite sur la figure 2.14. Les contraintes sur les niveaux d'abstraction induites par cette tâche sont :

POSITION $\prec$ ETAT ;
ROBOTS $\prec$ ETAT ;
POSITION $\prec$ BRIQUES ;
ROBOTS $\prec$ BRIQUES ;
ETAT $\prec$ BRIQUES et
BRIQUES $\prec$ ETAT.

L'examen de cette seule tâche conduirait donc à mettre $BRIQUES$ et $ETAT$ dans la même classe de noms d'attributs $\prec$-équivalents.

La figure 6.2 présente un graphe $\mathcal{G}$ complet dans un domaine lié à la planification d'expériences de biologie à bord d'une station spatiale de type Columbus[2]. Sur le graphe, un arc entre deux classes $\widehat{att}$ et $\widehat{att}'$ signifie que tous les défauts sur des noms d'attributs de $\widehat{att}$ doivent nécessairement être résolus avant ceux portant sur un nom d'attribut de $\widehat{att}'$, c'est-à-dire : $\widehat{att}' < \widehat{att}$.

Une fois les contraintes sur l'ordre des noms d'attributs exprimées, nous devons expliciter une hiérarchie d'abstraction vérifiant la propriété de monotonicité ordonnée d' IXTET .

Dans IXTET , nous gérons directement la hiérarchie partiellement ordonnée au sein du processus de planification. C'est dans cette perspective que nous définissons la notion d'*état d'abstraction* :

[1] Si l'on gère le prédicat *assert*, il faut aussi considérer e comme pouvant être une proposition temporelle de type *assert*.

[2] Cet exemple sera davantage détaillé au chapitre 6.

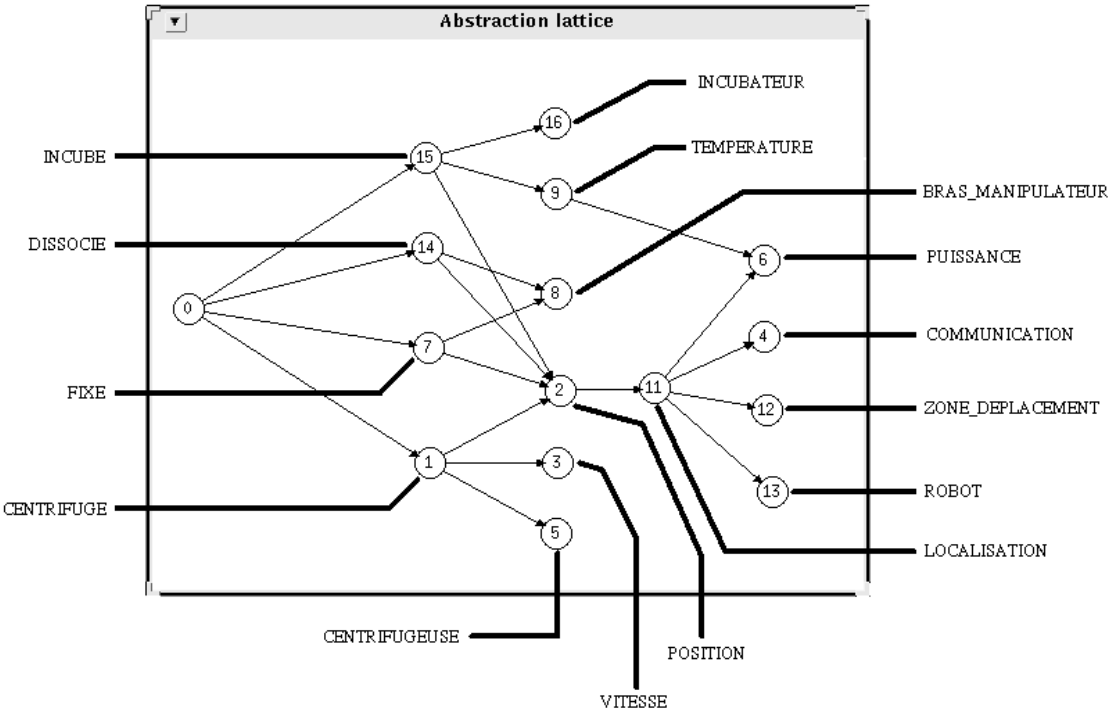


Figure 5.1: Graphe $\mathcal{G}$ pour le domaine COLUMBUS

Définition 16 (Etat d'abstraction) *: Un état d'abstraction est un couple* (R, C)[3] *d'ensembles de noms d'attributs, où* $R = \{att_{s1}, ..., att_{sn}\}$ *représente l'ensemble des attributs ayant été complétement résolus jusqu'ici, et* $C = \{att_{c1}, ..., att_{cm}\}$*, l'ensemble des attributs qui sont en cours de résolution.*

Un état d'abstraction décrit, pour un nœud donné de l'arbre de recherche, le degré d'abstraction du plan courant. A un nœud donné, seuls les défauts sur des noms d'attributs de C sont examinés. L'ensemble C de l'état d'abstraction courant caractérise donc le niveau d'abstraction courant L :

$$L = \{\phi \mid att_\phi \in C\}$$

Moyennant ces définitions, définir une hiérarchie d'abstraction se ramène donc à définir la manière dont évoluent les états d'abstraction le long des nœuds de l'arbre de recherche.

Dans le nœud racine correspondant au scénario initial, l'état d'abstraction est défini comme (R_0, C_0), où $R_0 = \emptyset$, et $C_0 = \{att \mid \{att' \mid \widehat{att} < \widehat{att}'\} = \emptyset\}$ c'est-à-dire les noms d'attributs les plus abstraits au sens de l'ordre partiel $<$ (éléments maximaux). Supposons alors que durant l'étape de résolution des défauts de L_0, un ensemble $\widehat{att}$, avec $att \in C_0$, de noms d'attributs $\prec$-équivalents ait été complétement résolu. Cela signifie que le plan partiel ne contient plus aucun défaut sur des noms d'attributs de $\widehat{att}$. Nous sommes sûrs, d'autre part,

[3]Nous notons R pour attributs Résolus et C pour attributs en Cours de résolution.

que de tels défauts n'apparaîtront pas dans la suite du processus ; en effet, la définition de C_0 interdit l'existence de défauts Φ de L_0 dont la résolution conduirait à l'apparition de nouveaux défauts ϕ', avec $att_{\phi'} \in \widehat{att}$.

Puisque aucun défaut sur les noms d'attributs $\prec$-équivalents de $\widehat{att}$ ne pourra apparaître, ces attributs peuvent être ôtés de l'ensemble C de l'état d'abstraction courant. L'approche hiérarchique classique consisterait à attendre que l'ensemble C soit complétement vide avant de calculer un nouvel ensemble C' et un nouveau niveau d'abstraction L'. En fait, cela n'est pas nécessaire et nous pouvons immédiatement ajouter à C certains nouveaux noms d'attributs à traiter à partir du moment où ceci ne viole pas les contraintes $\prec$ qui assurent l'OMP.

La règle générale de mise à jour du niveau d'abstraction courant (R, C) à un nœud de l'arbre de recherche est la suivante :

Définition 17 (Mise à jour des niveaux d'abstraction)
Lorsqu'un ensemble $\widehat{att}_{résolu}$ de noms d'attributs $\prec$-équivalents s'avère être complètement résolu, $\widehat{att}_{résolu}$ est ôté de C, rajouté à R et C est recalculé à partir du nouvel ensemble d'attributs résolus R par :

$$R' = R \cup \widehat{att}_{résolu}$$
$$C' = C - \widehat{att}_{résolu} \cup \{att_{nouveau} \mid \widehat{att}_{nouveau} < \widehat{att}_{résolu} \wedge \{att \mid \widehat{att}_{nouveau} < \widehat{att}\} \subset R'\}$$

Un exemple de mise à jour d'un état d'abstraction au cours du processus de planification est montré sur la figure 5.2 correspondant au noeud de l'arbre de recherche où le dernier défaut sur l'attribut $CENTRIFUGE$ a été résolu.

Comme pour l'approche classique, on montre le théorème suivant (preuve en annexe D) :

Théorème 9 *: Les hiérarchies d'abstraction utilisées dans IXTET vérifient la propriété de monotonicité ordonnée : pour chaque plan partiel courant possible, aucun raffinement envisageable de ce plan partiel suite à la résolution de défauts ne fera apparaître de nouveau défaut qui soit d'un niveau strictement supérieur au niveau d'abstraction courant.*

La différence principale avec l'approche classique tient dans le fait que les niveaux que nous générons ne forment pas une partition de l'ensemble des défauts : généralement, deux niveaux d'abstraction consécutifs s'intersectent. L'avantage de notre approche réside dans l'accroissement de l'opportunisme du contrôle qui ne prend en compte que les contraintes d'abstraction entre attributs qui sont directement issues du graphe $\mathcal{G}$. Il est ainsi possible de développer en priorité les branches parallèles du graphe $\mathcal{G}$ les plus opportunes pour la recherche du plan et d'attendre d'être dans un contexte plus complet pour avancer dans les branches moins opportunes.

Nous définissons la profondeur d'abstraction $d_{\mathcal{G}}$ comme le nombre de sommets du plus long chemin sur le graphe $\mathcal{G}$ et $p_{\mathcal{G}}$ le degré de parallélisme maximal[4] de $\mathcal{G}$. Soit $n_{\mathcal{G}}$ le nombre de

[4]Ce degré étant défini comme la taille d'un ensemble indépendant maximal pour l'ordre partiel $<$.

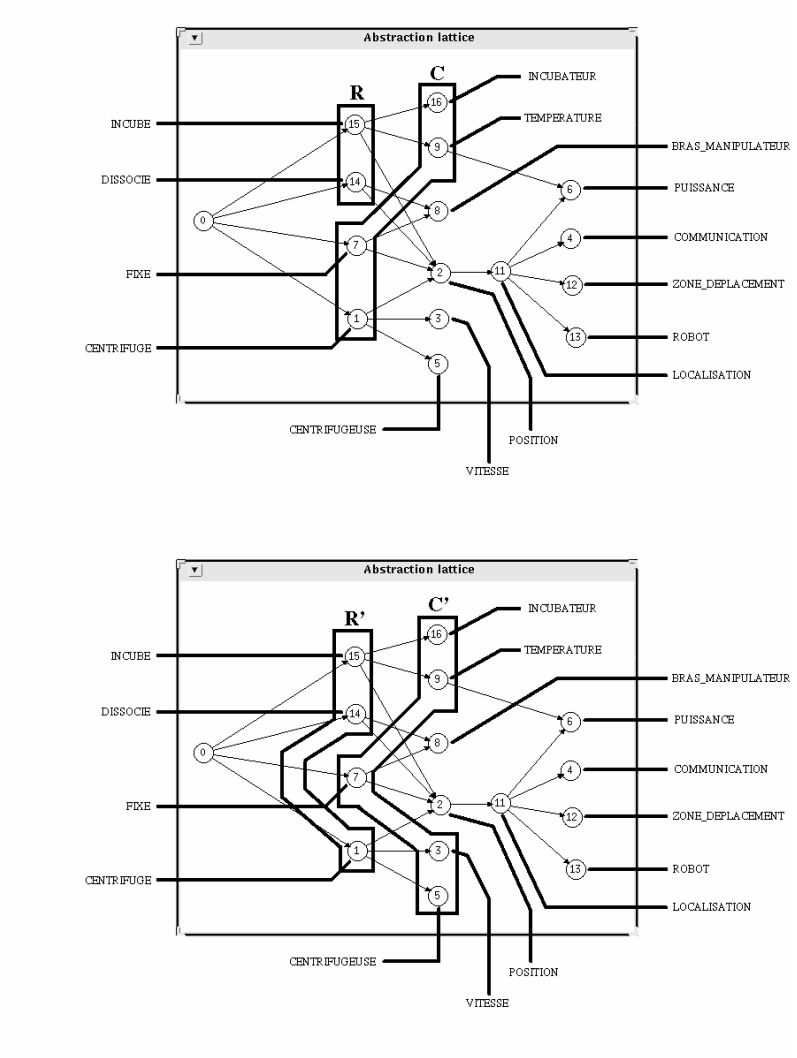


Figure 5.2: Deux états d'abstraction consécutifs

nœuds de $\mathcal{G}$. Le résultat suivant, facile à montrer, caractérise les hiérarchies d'abstraction indépendamment du problème spécifique traité :

Propriété 13 *: Le nombre maximal l des différents niveaux d'abstraction parcourus au cours de la recherche depuis le scénario initial $\mathcal{P}_0$ jusqu'à un plan solution $\mathcal{P}$ est tel que $d_{\mathcal{G}} \leq l \leq n_{\mathcal{G}}$.*

5.4 Quelques améliorations de la hiérarchie

5.4.1 Effets principaux et secondaires

Pour les mêmes raisons que celle décrites par Knoblock [Knoblock 94] et davantage formalisées dans [Fink 95b], il peut être intéressant de faire la distinction entre *effets principaux* et *effets secondaires* d'une tâche. Dans le contrôle du système de planification IxTeT , une

tâche peut être insérée au plan partiel dans le but de résoudre une proposition d'état non expliquée ou de produire une ressource susceptible d'intervenir dans un conflit (voir tableau 3.1). La tâche va permettre de résoudre le défaut si elle contient un effet, qu'il s'agisse d'un évènement (*event*) ou une production de ressource (*produce*)[5], adéquat. Souvent, néammoins, une tâche donnée ne pourra être insérée dans le plan que pour un sous-ensemble de ses effets. Par exemple la tâche $TRANSFERER\_BRIQUES$ décrite au chapitre 2 a comme effet que le robot sera dans la salle d'arrivée après exécution de la tâche (attribut d'état $POSITION$) ; toutefois, il semble inutile d'envisager l'insertion dans le plan d'une telle tâche pour établir le fait que le robot se trouve dans une salle donnée. Nous considérerons donc l'attribut $POSITION$ comme un attribut secondaire de la tâche, tandis que l'attribut de ressource $BRIQUES$ sera, lui, principal. En distinguant de cette manière attributs principaux et attributs secondaires, il sera possible d'éliminer de $\mathcal{G}$ certaines contraintes qui n'ont pas raison d'être étant donnée notre manière de gérer les effets des tâches au cours du processus de planification.

L'avantage de cette distinction est alors double : en plus d'une réduction du facteur de branchement dans l'arbre de recherche (il y aura moins de résolvantes pour les défauts de type "proposition non-expliquée" et ECM), cela permettra d'obtenir une hiérarchie moins contrainte ce qui donnera plus de liberté au contrôle de la recherche pour appliquer sa stratégie de recherche opportuniste. La nouvelle définition de l'ordre hiérarchique partiel $\prec$ devient alors :

Définition 18 (Contraintes $\prec_{P/S}$ sur les noms d'attributs) [6]
Pour chaque tâche T du domaine, si e est un événement (event) ou une proposition de production de ressources (produce)[7] **intervenant comme effet principal de** *T et si p est une proposition temporelle quelconque de T, alors $att_p \dot{\prec}_{P/S} att_e$. Nous noterons $\prec_{P/S}$ la fermeture transitive de $\dot{\prec}_{P/S}$.*

5.4.2 Prise en compte du problème à résoudre

Dans [Knoblock 94], l'auteur montre que de meilleures hiérarchies d'abstraction peuvent être obtenues en prenant en compte le problème de planification que l'on cherche à resoudre lors du calcul du graphe de contraintes $\mathcal{G}$. Le principe est le même que dans le cas de la distinction effet principal/effet secondaire que nous venons de décrire : il s'agit d'éviter d'insérer sur le graphe $\mathcal{G}$ des contraintes inutiles et d'obtenir ainsi une caractérisation plus générale des hiérarchies admissibles au sens de l'OMP. Nous montrons dans ce paragraphe comment on peut généraliser cette approche au formalisme IxTeT . Nous noterons toutefois que si les prédicats *assert* et *produce* ne sont pas gérés [8], la prise en compte du problème n'entraine aucune amélioration de la hiérarchie.

[5] Ou une proposition de type *assert* si ce type de prédicat est géré.
[6] Nous notons P/S pour Principal/Secondaire.
[7] Si l'on gère le prédicat *assert*, il faut aussi considérer e comme pouvant être une proposition temporelle de type *assert*.
[8] En d'autre termes, si l'on ne gère que *event*, *hold* et *use*.

Soit e une proposition temporelle de type *event*, *assert*, ou *produce* appartenant à une tâche $\mathcal{T}$. Nous dirons que e est **relevante** pour le scénario initial $\mathcal{P}_0$, et nous noterons $e \in relevant(\mathcal{T}, \mathcal{P}_0)$, si la proposition e peut justifier l'insertion de la tâche $\mathcal{T}$ dans le plan partiel pour résoudre un défaut du scénario initial ou un nouveau défaut que pourrait engendrer l'insertion dans le plan d'une tâche $\mathcal{T}'$, avec $relevant(\mathcal{T}', \mathcal{P}_0) \neq \emptyset$.

Nous dirons alors qu'un attribut *att* est **relevant pour le scénario initial** s'il existe une tâche $\mathcal{T}$ et une proposition e dans $\mathcal{T}$ de type *event*, *assert* ou *produce*, portant sur l'attribut *att* et telle que $e \in relevant(\mathcal{T}, \mathcal{P}_0)$.

Il est clair, d'après cette définition récursive des attributs relevants pour le scénario initial, qu'au cours de la résolution les seules tâches qui pourront être insérées dans le plan sont telles que $relevant(\mathcal{T}, \mathcal{P}_0) \neq \emptyset$.

La façon naturelle de définir les conditions suffisantes pour que la hiérarchie vérifie l'OMP devient alors :

Définition 19 (Contraintes $\prec_{SI}$ sur les noms d'attributs) [9]
Pour chaque tâche T du domaine, si e est un événement (event), une proposition de production de ressources (produce)[10] **tel que** $e \in relevant(\mathcal{T}, \mathcal{P}_0)$ *et si p est une proposition temporelle quelconque de T, alors $att_p \dot{\prec}_{SI} att_e$. Nous noterons $\prec_{SI}$ la fermeture transitive de $\dot{\prec}_{SI}$.*

Dans le cas particulier ou aucun prédicat de type *assert* ou *produce* n'intervient dans la description du domaine, nous montrons en annexe D la propriété suivante qui stipule que les ordres partiels $\prec$ et $\prec_{SI}$ sont identiques et donc que la prise en compte du scénario initial n'apporte aucun gain sur la hiérarchie générée.

Propriété 14 *En l'absence de prédicats de type assert ou produce,*
$\forall \mathcal{P} \in COMPL(\mathcal{P}_0)$, $\forall \Phi, \Phi'$ *défauts de* $\mathcal{P}$,

$$att_\Phi \prec att_{\Phi'} \Rightarrow att_\Phi \prec_{SI} att_{\Phi'}$$

Rappelons que $COMPL(\mathcal{P}_0)$ désigne l'ensemble des plans partiels $\mathcal{P}$ susceptibles d'être développés au cours de la recherche.

Il est intéressant de voir que la preuve de cette propriété repose sur le fait que les seuls effets possibles d'une tâche consistent en des événements ; lesquels événements expriment à la fois un effet et une condition sur la tâche (le fait que l'ancienne valeur nécessite d'être expliquée).

Dans le cas où des prédicats de type *assert* ou *produce* sont présents (ces prédicats expriment, typiquement, des effets qui ne sont associés à aucune condition particulière), on peut

[9] Nous notons SI pour Scénario Initial.
[10] Si l'on gère le prédicat *assert*, il faut aussi considérer e comme pouvant être une proposition temporelle de type *assert*.

montrer, comme l'illustre l'exemple ci-dessous que la prise en compte du scénario initial peut permettre de relâcher la hiérarchie.

Supposons le cas d'un robot dans une pièce capable d'effectuer des tâches comme le percement d'une saignée dans le mur ($PERCE\_SAIGNEE$), l'installation d'une tuyauterie dans la saignée afin d'installer l'eau courante ($INSTALLE\_EAU$) et de nettoyer les murs ($LAVE\_MUR$).

La formalisation de ces trois tâches est donnée sur la figure 5.3 ainsi qu'un petit scénario initial où, au départ, le mur est propre, l'eau non-installée et dont l'objectif est l'installation de l'eau.

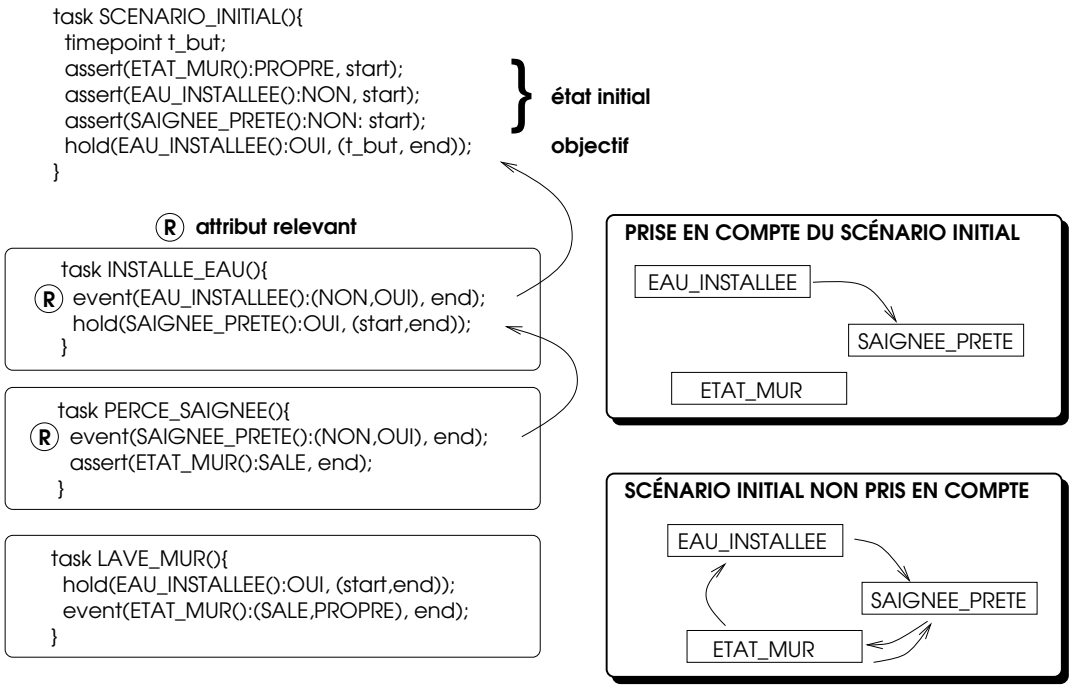


Figure 5.3: Une hiérarchie dépendant du scénario initial

5.4.3 Vers une hiérarchie plus fine

Il peut aussi y avoir des avantages à rendre la notion de niveau d'abstraction d'un défaut dépendante, non seulement du nom de l'attribut associé au défaut, mais aussi des instances possibles de ce défaut étant donné les variables atemporelles non-instanciées qu'il peut contenir. Il peut être par exemple justifié d'associer aux attributs instanciés $POSITION(ROBOT1)$ et $POSITION(OBJET1)$ des niveaux d'abstraction différents dans la mesure où les tâches du domaine ne modifient pas leurs valeurs de manière identique.

La solution implémentée sur ALPINE[Knoblock 94] consiste à associer un niveau d'abstraction à un type particulier de littéraux, typologie qui repose sur une classification *prédéfinie* des constantes du domaine. Par exemple deux niveaux différents seront associés aux littéraux partiellement instanciés $POSITION(?robot)$ et $POSITION(?objet)$ si les deux variables

?*robot* et ?*objet* appartiennent à des classes différentes. Les nœuds du graphe représentent alors des sous-ensembles de littéraux partiellement instanciés directement issus de la description syntaxique des modèles d'opérateurs.

Une amélioration possible de cette technique consiste à ne pas imposer à l'utilisateur la définition a priori d'une classification des constantes du domaine mais de générer celle-ci automatiquement en fonction de la description syntaxique des tâches. Nous montrons dans ce qui suit comment ceci peut être mis en œuvre dans le cadre du planificateur IxTeT .

Dans IxTeT , chaque défaut est constitué d'un ensemble de propositions temporelles contenant des variables atemporelles partiellement instanciées, ceci dans le contexte d'un ensemble C_V de contraintes sur ces variables. Notre objectif est, étant donné deux défauts Φ et Φ', de généraliser, en fonction des instances possibles de ces défauts, la notion de contrainte hiérarchique $att_\Phi \prec att_{\Phi'}$ que nous avons déjà vue.

Soit Φ un défaut sur un attribut *att*, Φ est un ensemble de propositions temporelles sur des attributs $(att(?x_1), ..., att(?x_n))$ (pour des raisons de clarté, nous ne supposons ici que des attributs unaires).

Soit d'autre part la relation d'ordre $\ll$ entre attributs totalement instanciés obtenue en parcourant la description syntaxique de tous les modèles de tâches *totalement instanciés* envisageables et selon les mêmes règles que celles données dans la définition 14.

On peut alors définir les contraintes hiérarchiques suivantes entre défauts :

Définition 20 (Contraintes $\ll_{C_V}$ entre défauts) *:*
Soient Φ et Φ' deux défauts respectivement associés aux ensembles d'attributs partiellement instanciés $(att(?x_1), ..., att(?x_n))$ et $(att(?x'_1), ..., att(?x'_p))$. On dira que Φ est plus abstrait que Φ' et on notera $\Phi \ll_{C_V} \Phi'$ si et seulement si, il existe une instanciation consistante avec C_V, $(X_1, ..., X_n, X'_1, ..., X'_p)$ des variables intervenant dans Φ et Φ' telle que : $\exists (i,j) \in [1,n] \times [1,p] \mid att(X_i) \ll att(X'_j)$.

Une contrainte $\Phi \ll_{C_V} \Phi'$ est ainsi détectée dès qu'il existe une instanciation du plan partiel où l'un quelconque des attributs de Φ est plus abstrait, au sens de l'analyse des tâches totalement instanciées, que l'un quelconque des attributs de Φ'.

Toutefois, la génération en ligne de la hiérarchie, telle qu'elle est décrite dans la définition 17 est complexe. La raison de cette complexité est qu'il y a énormément de relations $\ll_{C_V}$ devant être calculées en cours de planification et ce, d'autant plus que chaque plan partiel possède son propre réseau de contraintes C_V.

Deux simplifications peuvent être envisagées. Tout d'abord, il est possible d'utiliser un réseau de contrainte C'_V en éliminant de C_V toutes les contraintes d'inégalité et de dépendance, contraintes qui complexifient le processus de vérification sur chaque instance possible. Ainsi, nous considèrerons comme instances possibles pour $(?x_1, ..., ?x_n, ?x'_1, ..., ?x'_p)$ tout le produit

cartésien des domaines, ce qui simplifie le calcul de $\ll_{C_V}$ et conduit, en fait, à une relation $\ll_{C'_V}$ surcontrainte par rapport à $\ll_{C_V}$ mais qui contient néammoins des informations intéressantes quant à la structure du domaine de planification. Une seconde possibilité consisterait à modifier le processus de synthèse de l'ordre partiel $\ll$ sur les attributs totalement instanciés. Au lieu de définir une relation entre attributs totalement instanciés, on pourrait définir une relation $\ll'$ entre *ensembles* d'attributs totalement instanciés, ces ensembles correspondant au produit cartésien des domaines des variables tels qu'ils sont définis dans l'attribut (voir l'exemple sur la figure 5.4).

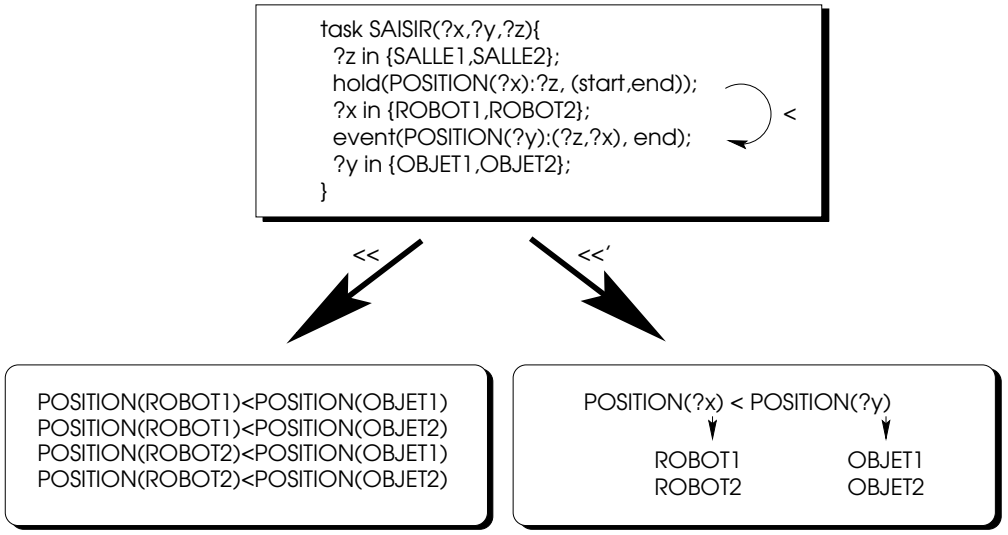


Figure 5.4: Ordre $\ll'$ et classification des objets

A noter que la définition de $\ll'$ est une manière de générer automatiquement la classification des constantes utilisée dans ALPINE. Ces deux méthodes peuvent être efficacement combinées pour définir une relation $\ll'_{C'_V}$ qui devrait pouvoir être calculable en pratique.

5.5 Implémentation, résultats et limitations

5.5.1 Génération automatique de la hiérarchie

L'approche que nous venons de décrire, à l'exception de la prise en compte du scénario initial, §5.4.2 et des attributs partiellement instanciés §5.4.3, a été implémentée sur le système de planification IxTeT . Un *module de génération automatique des niveaux d'abstraction* permet de construire le graphe $\mathcal{G}$ en analysant la description syntaxique des tâches et en ne posant que l'ensemble des contraintes suffisantes à assurer l'OMP. L'utilisateur a ensuite, s'il le souhaite, la possibilité de surcontraindre cette structure hiérarchique.

Les contraintes $\prec_{P/S}$ sont représentées comme un treillis IxTeT [Ghallab 89b] où chaque nœud représente un ensemble de noms d'attributs devant nécessairement être géré dans le même niveau d'abstraction. Ce treillis est initialisé avec autant de nœuds non contraints

qu'il y a de noms d'attributs différents sur le domaine. Les tâches qui sont relevantes pour le domaine sont alors parcourues récursivement et les conditions suffisantes pour satisfaire l'OMP sont posées sur le treillis. A noter que nous avons implémenté la notion d'effets principaux/secondaires d'une tâche : une tâche ne pouvant être insérée dans le plan que pour utiliser ses effets principaux, seuls ses effets principaux doivent être pris en compte lorsque l'on génère la hiérarchie d'abstraction (cf. définition 18). Lorsque l'insertion d'une contrainte fait apparaître un cycle sur le graphe orienté, les nœuds de ce cycle sont fusionnés en un unique nœud. L'efficacité de la propagation des contraintes et des requêtes sur un treillis IXTET (ces deux procédures ont une complexité linéaire en moyenne) permettent une complexité globale du processus de génération de la hiérarchie en $O(m.n.s.e)$ si m est le nombre de modèles de tâches, n le nombre de noms d'attributs du domaine, s la taille moyenne des tâches en terme de propositions temporelles et e le nombre moyen d'effets principaux d'une tâche (en général, $e = 1$ ou 2). Un exemple de hiérarchie automatiquement généré est celui de la figure 6.2.

5.5.2 Gestion de la hiérarchie d'abstraction

Comme nous l'avons décrit plus haut, nous associons à chaque nœud de l'arbre de recherche un *état d'abstraction* local (R, C). Lorsqu'un nom attribut att de C se retrouve vierge de tout défaut, une vérification est effectuée pour voir si l'ensemble $\widehat{att}$ des noms d'attributs $\prec_{P/S}$-équivalents à att sont aussi résolus ; Dans ce cas, le niveau d'abstraction courant est modifié selon la définition 17.

La figure 5.2 présente deux états d'abstraction consécutifs lors de la planification d'expériences de biologies (domaine Columbus).

A un nœud donné de la recherche, la complexité supplémentaire du fait de la prise en compte de la hiérarchie d'abstraction est de l'ordre de $O(n.p_{\mathcal{G}})$ dans les cas où le niveau d'abstraction est modifié. Cette complexité reste toujours négligeable devant la complexité de l'analyse du plan partiel. A noter que pour calculer un nouvel état d'abstraction, nous n'utilisons pas, pour des raisons de complexité, l'algorithme qui pourrait naturellement être suggéré par la définition 17 (et qui demanderait d'exploiter toute la fermeture transitive de $\dot{\prec}$) mais une définition strictement équivalente reposant uniquement sur l'ensemble des contraintes posées $\dot{\prec}$.

Notre approche a été validée sur de nombreux domaines comme le contrôle au niveau tâche d'une équipe de robots mobiles (Hilares, Iares[11]), la planification des expériences de biologie que nous avons évoqué plus haut (Columbus[12]), ou l'ordonnancement de l'intégration de cases à équipements sur un lanceur de satellites (Ariane[13]). Les principaux résultats sont résumés sur les tableaux 5.1 et 5.2. Les plans solution pour ces problèmes comportent entre

[11] Ce domaine a été codé avec IXTET pour le projet IARES en collaboration avec le CNES.

[12] Ce domaine a été modélisé sous IXTET dans le cadre du projet PADRE du MESR, en collaboration avec les sociétés IXI and Matra Marconi Space France (MMS).

[13] Ce problème concernant le lanceur européen Ariane 4 nous a été soumis par MMS.

Domain		facteur de gain	profondeur $d_{\mathcal{G}}$	$n_{\mathcal{G}}/p_{\mathcal{G}}$
Hilares		1.4	2	1.5
Iares		2.7	3	1.5
Columbus	problème 1	1.7	4	1.7
	problème 2	2.3	4	1.7
Ariane		3.9	6	6
Room finishing		8.8	6	4

Tableau 5.1: Gains globaux liés à la hiérarchisation

	hiérarchie ordonnée hors-ligne :	hiérarchie dynamique :
Nœuds	305	59
Retour-arrières	14	0
Temps CPU	14.1 s	2.8 s

Tableau 5.2: Gains liés à l'approche de moindre engagement

20 et 30 tâches environ, excepté le problème Ariane qui est un pur problème d'ordonnancement mettant une cinquantaine de tâches en jeu. Le facteur de gain est le rapport entre le temps CPU sans utiliser d'abstraction et celui en utilisant notre approche de hiérarchie dynamique.

Deux conséquences importantes peuvent se dégager de l'étude que nous avons menée sur la hiérarchisation. Ces résultats ont été montrés dans le cadre du système IxTeT mais, dans la mesure où ils reposent essentiellement sur l'approche de moindre engagement, ils restent aussi valables pour la majorité des systèmes de planification utilisant une telle approche.

1. Comme seulement un sous-ensemble des défauts sont examinés, l'analyse du plan partiel à chaque nœud de l'arbre de recherche est plus rapide que sans l'utilisation de hiérarchie. Le temps moyen passé à un nœud est globalement proportionnel au nombre de noms d'attributs présents dans C et donc, il dépend de la topologie du graphe $\mathcal{G}$. La profondeur d'abstraction $d_{\mathcal{G}}$ et le rapport $n_{\mathcal{G}}/p_{\mathcal{G}}$ entre la taille du graphe d'abstraction et son degré maximal de parallélisme sont deux facteurs qui peuvent permettre d'estimer le gain possible. Dans nos exemples, le facteur de gain varie de 1.4 (Hilares) à 5 (Ariane).

2. La hiérarchisation permet une structuration de l'espace de recherche qui conduit, en général, à une réduction des risques de retour-arrière dans la mesure où les caractéristiques les plus structurantes du plan sont examinées et résolues en premier. Notre approche originale de moindre engagement permet une exploitation encore plus poussée de cette propriété. Le tableau 5.2 illustre ceci à partir d'un problème très contraint du domaine IARES où notre approche dynamique trouve une solution sans retour-arrière et en environ 5 fois moins de temps qu'en utilisant une hiérarchie classique.

D'un autre côté, la hiérarchisation de la recherche signifie que le contrôle va se focaliser sur un niveau d'abstraction donné et "faire abstraction" des niveaux plus concrets. Une conséquence est que l'heuristique h mesurant la distance entre le nœud courant et un nœud solution sera moins informée et moins fiable pour indiquer au module de contrôle sur quel nœud revenir en arrière le cas échéant (nous pourrions bien entendu prendre en compte tous les défauts du plan partiel uniquement dans le calcul de h, mais cela reviendrait à perdre le gain important que nous avons déjà mentionné lié au nombre de défauts examinés). Pour résumer, la hiérarchisation réduit les risques de retour-arrière, mais lorsqu'un retour arrière survient, elle complexifie le problème du choix du nœud où revenir.

5.6 Conclusion

Nous avons vu comment un critère de hiérarchie (la propriété de monotonicité ordonnée) initialement défini sur un formalisme de type STRIPS pouvait être étendu à une représentation beaucoup plus riche prenant en compte le temps et les ressources et où le besoin d'une structuration de la recherche est, de ce fait, encore plus fort.

En respectant la stratégie de moindre engagement utilisée dans IxTeT (et dans la majorité des algorithmes de planification récents), nous avons développé une hiérarchie dynamique dépendant non seulement du problème à résoudre mais aussi de l'état courant du plan partiel en cours de planification. Seules les contraintes nécessaires à la vérification de l'OMP sont prises en compte ; aucune contrainte a priori n'est insérée afin d'aboutir à un ordre total.

L'intégration de notre approche hiérarchique au système de planification est schématisée sur la figure 5.5.

L'intérêt de cette hiérarchisation de la recherche, en termes de réduction des risques de retour-arrière et d'efficacité globale a été clairement établi dans différents domaines.

Une question importante qui reste posée concerne la démonstration théorique de l'intérêt d'une propriété telle que l'OMP et du contrôle associé. Certains résultats intéressants ont déjà été obtenus par exemple dans [Bacchus 94], ou dans le domaine des CSP pour les problèmes de sélection de variables, sélection qui est le pendant CSP de la sélection d'un défaut [Purdom 83], mais aucune analyse complète n'a été proposée à ce jour.

Les recherches futures dans ce domaine devraient consister à raffiner la granularité de notre hiérarchie pour la gérer non plus au niveau des noms d'attributs mais au niveau des attributs partiellement instanciés. Il est aussi envisageable d'exploiter la dimension temporelle du formalisme IxTeT . Cela reviendrait à associer un niveau d'abstraction à un défaut en fonction des attributs partiellement instanciés qui le composent et de sa position temporelle dans le plan partiel. Ces développements devraient renforcer l'idée que les techniques courantes d'abstraction utilisées en planification sont plus liées à une stratégie de sélection du prochain défaut à résoudre qu'à une stratégie globale de résolution hiérarchique du problème.

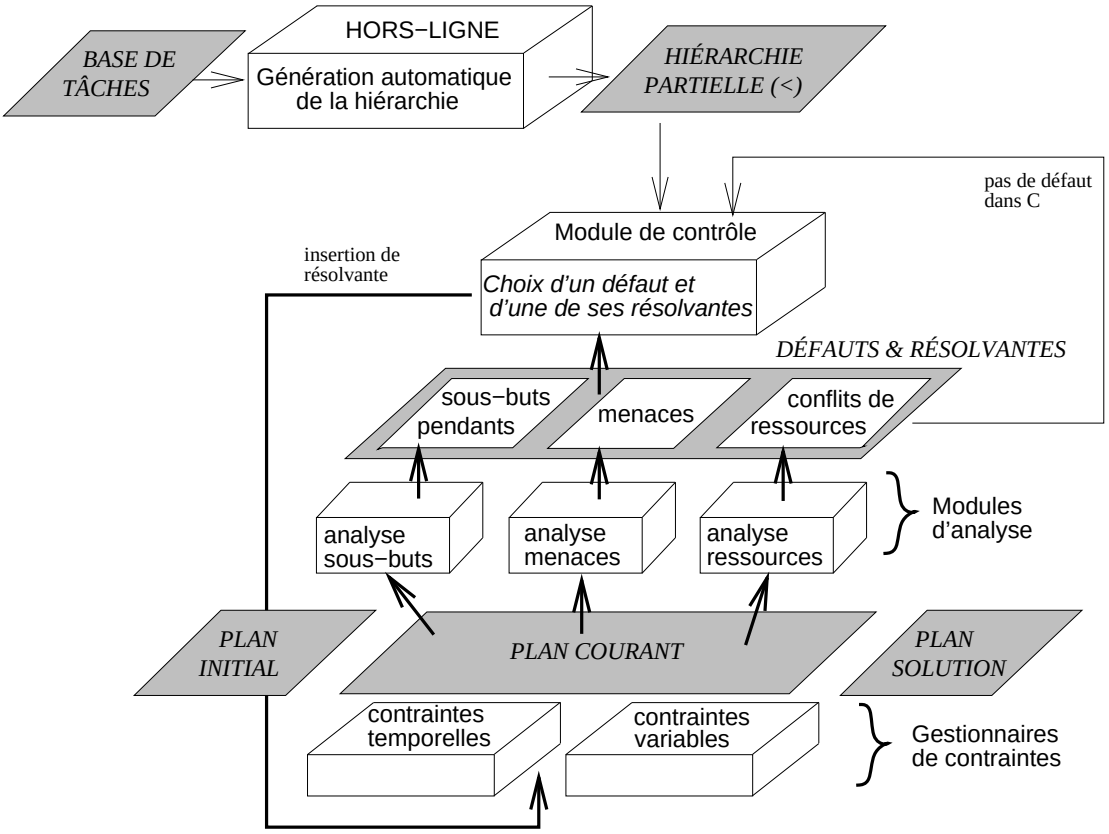


Figure 5.5: Architecture du système hiérarchisé

Chapitre 6

Exemples, résultats et analyses

6.1 Introduction

Dans ce chapitre illustratif, nous présentons une série de domaines que nous avons codés dans le formalisme IXTET ainsi que des exemples de plans solution. On notera la grande variabilité des problèmes pouvant être représentés et résolus qui vont des problèmes d'ordonnancement pur à ceux de pure synthèse de plans sans prise en compte de ressources.

Les principales stratégies pour effectuer les choix non-déterministes que nous avons décrites dans ce mémoire sont analysées dans le cadre de ces domaines de planification.

6.2 Un exemple complet : COLUMBUS

Ce domaine a été modélisé dans le cadre du projet PADRE du MESR en collaboration avec les sociétés IXI et Matra Marconi Space. Il s'agit de planifier des expériences de biologie à bord d'une station spatiale de type Columbus. La figure 6.1 schématise l'environnement du laboratoire. Les expériences à planifier consistent à manipuler des cultures en utilisant certains équipements scientifiques (incubateur, centrifugeuse, microscope, ...). Deux robots peuvent se déplacer dans le laboratoire et déplacer les objets. La plupart des expériences proprement dites se font au moyen d'un bras manipulateur.

Cet environnement fait intervenir un certain nombre de ressources non-partageables (les instruments scientifiques, les robots, le bras manipulateur, les différents éléments manipulés), mais aussi des ressources partageables (la puissance électrique instantanée maximale, le débit des canaux de communication).

D'autre part, il s'agit d'un problème de synthèse de plans dans la mesure où nous spécifions uniquement l'état final des cultures sur lesquelles doivent porter les expériences.

La description du domaine dans le formalisme IXTET (constantes du domaine, attributs d'état et de ressources, modèles de tâches) est donnée ci-dessous. La hiérarchie générée automatiquement à partir de cette modélisation est celle que nous avons déjà vue sur la figure 6.2.

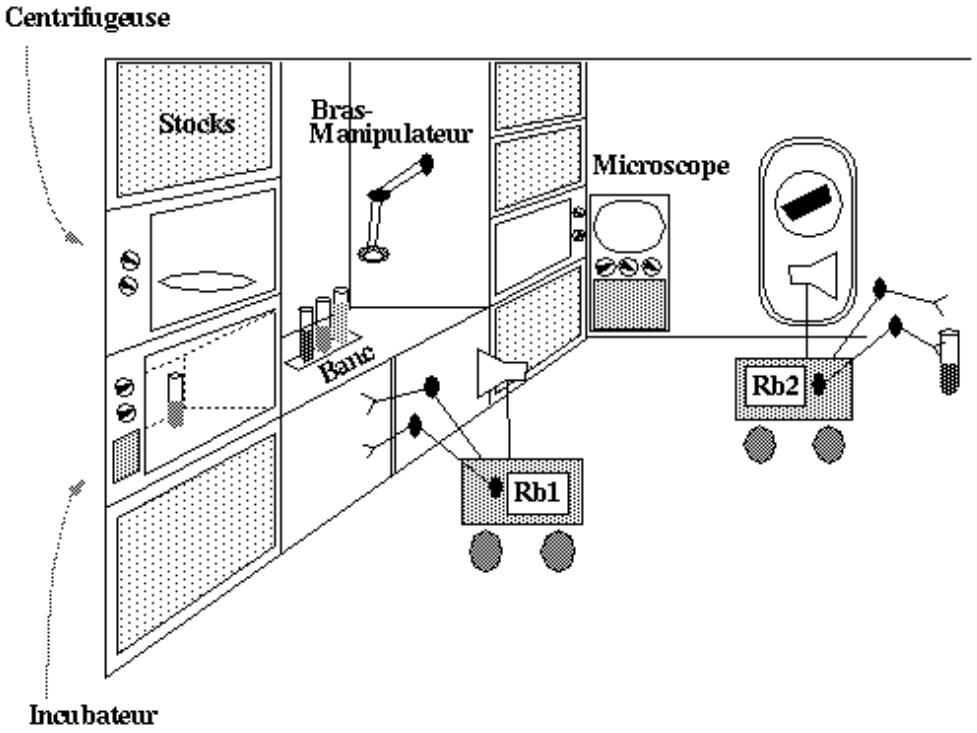


Figure 6.1: Le domaine COLUMBUS

Constantes du domaine:

```
constant ROBOTS = { Rb1, Rb2 };
constant ZONES = { Banc, Stock, Centrif, Poubelle, Incub, Micro, Mouvmt } | ROBOTS;
constant CULTURES = { Culture1, Culture2 };
constant TUBES = { Tube1, Tube2 };
constant FILTRES = { Filtre1, Filtre2, Filtre3 };
constant FLASQUES = { Flasque1, Flasque2 };
constant SERINGUES = { S1, S2, S3, S4, S5 };
constant ELEMENTS = CULTURES | FILTRES | FLASQUES | SERINGUES | TUBES;
constant VITESSES = {V0, V1, V2};
constant TEMPERATURES = { Amb, T20, T30 };
constant BOOLEEN = { vrai, faux };
```

Attributs du domaine:

```
attribute POSITION(?x) {
  ?x in ELEMENTS;
  ?value in ZONES;}
attribute LOCALISATION(?x) {
  ?x in ROBOTS;
  ?value in ZONES;}
attribute TEMPERATURE() {
  ?value in TEMPERATURES;}
attribute VITESSE() {
  ?value in VITESSES;}
attribute INCUBE(?Elt,?Temp) {
  ?Temp in TEMPERATURES;
  ?Elt in CULTURES;
  ?value in BOOLEEN;}
attribute CENTRIFUGE(?Elt,?Vits) {
  ?Vits in VITESSE;
  ?Elt in CULTURES;
  ?value in BOOLEEN;}
attribute DISSOCIE(?Elt) {
  ?Elt in CULTURES;
  ?value in BOOLEEN;}
attribute FIXE(?Elt) {
  ?Elt in CULTURES;
  ?value in BOOLEEN;}
```

Ressources du domaine:

```
resource ROBOT(?r) { ?r in ROBOTS;
  capacity = 1;}
resource BRAS_MANIPULATEUR { capacity = 1;}
resource MICROSCOPE() { capacity = 1;}
resource CENTRIFUGEUSE() { capacity = 1;}
resource PUISSANCE() { capacity = 100;}
resource COMMUNICATION() { capacity = 800;}
resource BANC_TRAVAIL() { capacity = 2;}
resource INCUBATEUR() { capacity = 5;}
resource ZONE_DEPLACEMENT() { capacity = 3;}
```

Description des tâches:

```
task POMPER(?Elt, ?S)(start, end) {
  ?S in SERINGUES;
  hold (POSITION(?Elt):Banc, (start,end));
  use (PUISSANCE:10, (start, end));
  use (COMMUNICATION:30, (start, end));
  use (BRAS_MANIPULATEUR:1, (start, end));
  (end - start) in [00:01:00, 00:03:00]; }

task CHAUFFER(?Temp)(start, end) {
  event (TEMPERATURE():(Amb, ?Temp), start);
  event (TEMPERATURE():(?Temp, Amb), end);
  hold (TEMPERATURE():?Temp, (start,end));
  use (PUISSANCE:40, (start, end));}

task INCUBER(?Elt, ?Temp)(start, end){
  hold (TEMPERATURE():?Temp, (start, end));
  hold (POSITION(?Elt):Incub, (start, end));
  event (INCUBE(?Elt,?Temp):(faux, vrai), end);
  use (PUISSANCE:8, (start, end));
  use (COMMUNICATION:50, (start, end));
  use (INCUBATEUR:1, (start, end));
  (end - start) in [00:09:00, 00:10:00];}

task FIXATION(?Elt, ?S, ?T)(start, end){
  ?S in SERINGUES;
  ?T in TUBES;
  hold (POSITION(?Elt):Banc, (start, end));
  hold (POSITION(?T):Banc, (start, end));
  event (FIXE(?Elt):(faux, vrai), end);
  use (BRAS_MANIPULATEUR:1, (start, end));
  use (PUISSANCE:10, (start, end));
  use (COMMUNICATION:30, (start, end));
  (end - start) in [00:06:00, 00:08:00]; }

task INJECTER(?Elt, ?S)(start, end) {
  ?S in SERINGUES;
  hold (POSITION(?Elt):Banc, (start,end));
  use (PUISSANCE:10, (start, end));
  use (COMMUNICATION:30, (start, end));
  use (BRAS_MANIPULATEUR:1, (start, end));
  (end - start) in [00:01:00, 00:03:00]; }

task OBSERVER(?Elt)(start, end) {
  hold (POSITION(?Elt):Micro, (start,end));
  use (PUISSANCE:10, (start, end));
  use (COMMUNICATION:400, (start, end));
  use (MICROSCOPE:1, (start, end));
  (end - start) in [00:15:00, 00:20:00]; }

task CENTRIFUGER(?Elt, ?Vits)(start, end){
  hold (POSITION(?Elt):Centrif, (start, end));
  event (VITESSE():(V0, ?Vits), start);
  event (VITESSE():(?Vits, V0), end);
  hold (VITESSE():?Vits, (start, end));
  event (CENTRIFUGE(?Elt,?Vits):(faux, vrai), end);
  use (PUISSANCE:20, (start, end));
  use (COMMUNICATION:40, (start, end));
  use (CENTRIFUGEUSE:1, (start, end));
  (end - start) in [00:09:00, 00:10:00];}

task DISSOCIATION(?Elt, ?S, ?F)(start, end){
  ?S in SERINGUES;
  ?F in FILTRES;
  hold (POSITION(?Elt):Banc, (start, end));
  hold (POSITION(?F):Banc, (start, end));
  event (DISSOCIE(?Elt):(faux, vrai), end);
  use (BRAS_MANIPULATEUR:1, (start, end));
  use (PUISSANCE:8, (start, end));
  use (COMMUNICATION:20, (start, end));
  (end - start) in [00:05:00, 00:07:00];}
```

```
task ALLER_A(?r, ?Zone1, ?Zone2)(start, end){
  event (LOCALISATION(?r):(?Zone1, Mouvmt), start);
  event (LOCALISATION(?r):(Mouvmt, ?Zone2) , end);
  hold (LOCALISATION(?r):Mouvmt, (start, end));
  use (ROBOT(?r):1, (start, end));
  use (PUISSANCE:30, (start, end));
  use (COMMUNICATION:300, (start, end));
  use (ZONE_DEPLACEMENT:1, (start, end));
  (end - start) in [00:02:00, 00:04:00];}

task ELIMINER(?r,?Elt)(start, end){
  variable ?Zone0, ?ZoneR;
  event (POSITION(?Elt):(?Zone0, ?r), start);
  event (POSITION(?Elt):(?r, Poubelle) , end);
  hold (POSITION(?Elt):?r, (start, end));
  event (LOCALISATION(?r):(?ZoneR, Mouvmt), start);
  event (LOCALISATION(?r):(Mouvmt, ?Zone) , end);
  hold (LOCALISATION(?r):Mouvmt, (start, end));
  use (ROBOT(?r):1, (start, end));
  use (PUISSANCE:40, (start, end));
  use (COMMUNICATION:400, (start, end));
  use (ZONE_DEPLACEMENT:1, (start, end));
  (end - start) in [00:03:00, 00:10:00];}

task INSTALLER(?r, ?Elt, ?Zone)(start, end){
  variable ?Zone0, ?ZoneR;
  event (POSITION(?Elt):(?Zone0, ?r), start);
  event (POSITION(?Elt):(?r, ?Zone) , end);
  hold (POSITION(?Elt):?r, (start, end));
  event (LOCALISATION(?r):(?ZoneR, Mouvmt), start);
  event (LOCALISATION(?r):(Mouvmt, ?Zone) , end);
  hold (LOCALISATION(?r):Mouvmt, (start, end));
  use (ROBOT(?r):1, (start, end));
  use (PUISSANCE:40, (start, end));
  use (COMMUNICATION:400, (start, end));
  use (ZONE_DEPLACEMENT:1, (start, end));
  (end - start) in [00:03:00, 00:10:00];}
```

Voici un exemple de problème de planification où l'on impose :

- que la culture $Culture1$ qui était initialement au stock passe successivement dans les états $FIXE$, $INCUBE$ puis $DISSOCIE$, après quoi, elle devra être jetée à la poubelle ; et
- que la culture $Culture2$ qui était initialement au stock passe successivement dans les états $DISSOCIE$, $INCUBE$ puis $CENTRIFUGE$, après quoi, elle-aussi, devra être jetée.

Le graphe de contraintes sur la hiérarchie généré à partir des tâches est alors celui de la figure 6.2.

D'autre part, le problème suppose qu'entre la date 00:00:00 de début du plan et la date 00:20:00, une chute de moitié de la puissance électrique instantanée est prévue. On notera au passage comment il est possible de représenter un profil de disponibilité de ressource au moyen des prédicats *use* (la quantité représente une quantité utilisé qui est exogène au plan et qui n'est donc pas disponible pour celui-ci). Le plan recherché doit avoir une durée maximale inférieure à 42 minutes.

La solution trouvée est représentée sur la figure 6.3. Le système met environ 30 s pour générer une telle solution. On remarquera que le plan solution est très contraint étant donné la durée maximale imposée ; le plan proposé exploite donc au maximum le parallélisme possible des actions. Jusqu'à 4 tâches peuvent être exécutées simultanément : $CHAUFFER(T20)$, $INCUBER(Culture1, T20)$, $DISSOCIATION(Culture2, s1, Filtre1)$ et $INSTALLER(Rb2, Tube1, Banc)$. L'aspect linéaire du début du plan est principalement lié à la chute de puissance électrique qui limite le parallélisme.

Scénario initial:

```
task SCENARIO_INITIAL()(start, end){

  timepoint t1, t2, t3, t4, t5, t6, t7, t8, t9;

  //- Etat initial:

  explained event (POSITION(Culture1):(?,Stock) , start);
  explained event (FIXE(Culture1):(?, false), start);
  explained event (DISSOCIE(Culture1):(?, false), start);
  explained event (INCUBE(Culture1,Amb):(?, false), start);
  explained event (INCUBE(Culture1,T20):(?, false), start);
  explained event (INCUBE(Culture1,T30):(?, false), start);
  explained event (CENTRIFUGE(Culture1,V0):(?, false), start);
  explained event (CENTRIFUGE(Culture1,V1):(?, false), start);
  explained event (CENTRIFUGE(Culture1,V2):(?, false), start);

  explained event (POSITION(Culture2):(?,Stock) , start);
  explained event (DISSOCIE(Culture2):(?, false), start);
  explained event (INCUBE(Culture2,Amb):(?, false), start);
  explained event (INCUBE(Culture2,T20):(?, false), start);
  explained event (INCUBE(Culture2,T30):(?, false), start);
  explained event (CENTRIFUGE(Culture2,V0):(?, false), start);
  explained event (CENTRIFUGE(Culture2,V1):(?, false), start);
  explained event (CENTRIFUGE(Culture2,V2):(?, false), start);

  explained event (POSITION(Tube1):(?,Stock) , start);
  explained event (POSITION(Tube2):(?,Stock) , start);

  explained event (POSITION(Filter1):(?,Stock) , start);
  explained event (POSITION(Filter2):(?,Stock) , start);
  explained event (POSITION(Filter3):(?,Stock) , start);

  explained event (POSITION(S1):(?,Stock) , start);

  explained event (POSITION(Flask1):(?,Cooler) , start);
  explained event (POSITION(Flask2):(?,Cooler) , start);

  explained event (LOCALISATION(Rb1):(?, Banc), start);
  explained event (LOCALISATION(Rb2):(?, Stock), start);

  explained event (TEMPERATURE:(?, Amb), start);
  explained event (VITESSE:(?, V0), start);

  //- Objectifs:

  hold (FIXE (Culture1):true, (t2, end));
  hold (INCUBE (Culture1,T20):true, (t3, end));
  hold (DISSOCIE (Culture1):true, (t4, end));
  hold (POSITION (Culture1):Poubelle, (t5, end));

  hold (DISSOCIE (Culture2):true, (t6, end));
  hold (INCUBE (Culture2,T20):true, (t7, end));
  hold (CENTRIFUGE (Culture2,V1):true, (t8, end));
  hold (POSITION (Culture2):Poubelle, (t9, end));

  //- Profil de disponibilité des ressources

  use (PUISSANCE():50, (start,t1));

  //- Contraintes temporelles

  start < t2 < t3 < t4 < t5;
  start < t6 < t7 < t8 < t9;

  (t1 - start) in [00:20:00, 00:20:00];
  (end - start) in [00:00:00, 00:42:00]; }
```

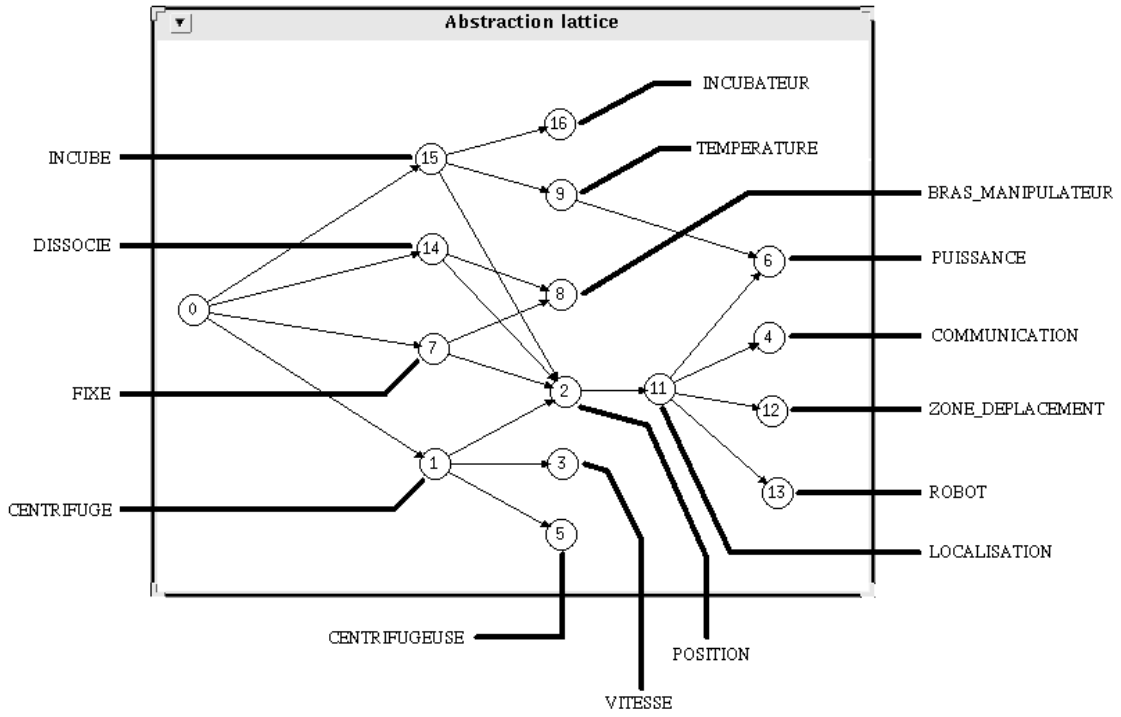


Figure 6.2: Graphe $\mathcal{G}$ pour le domaine COLUMBUS

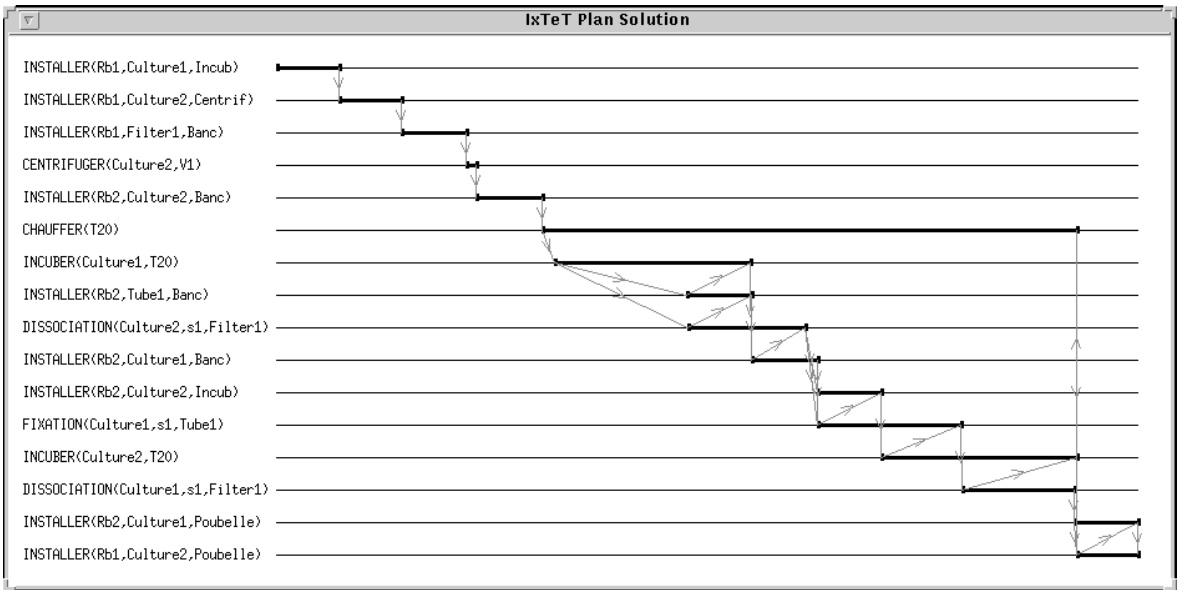


Figure 6.3: Plan solution dans le domaine COLUMBUS

Pour les besoins de l'affichage graphique des événements concernés, une linéarisation possible du plan solution est présentée sur la figure 6.4 (en partie seulement). On y voit les changements de valeurs prévus pour quelques attributs d'état suite à l'exécution des tâches,

ainsi que l'utilisation cumulée des ressources. On vérifiera en particulier que ce plan solution instancié vérifie bien les critères de cohérences d'état et de ressources que nous avons défini aux paragraphes §3.2.1 et §3.3.1.

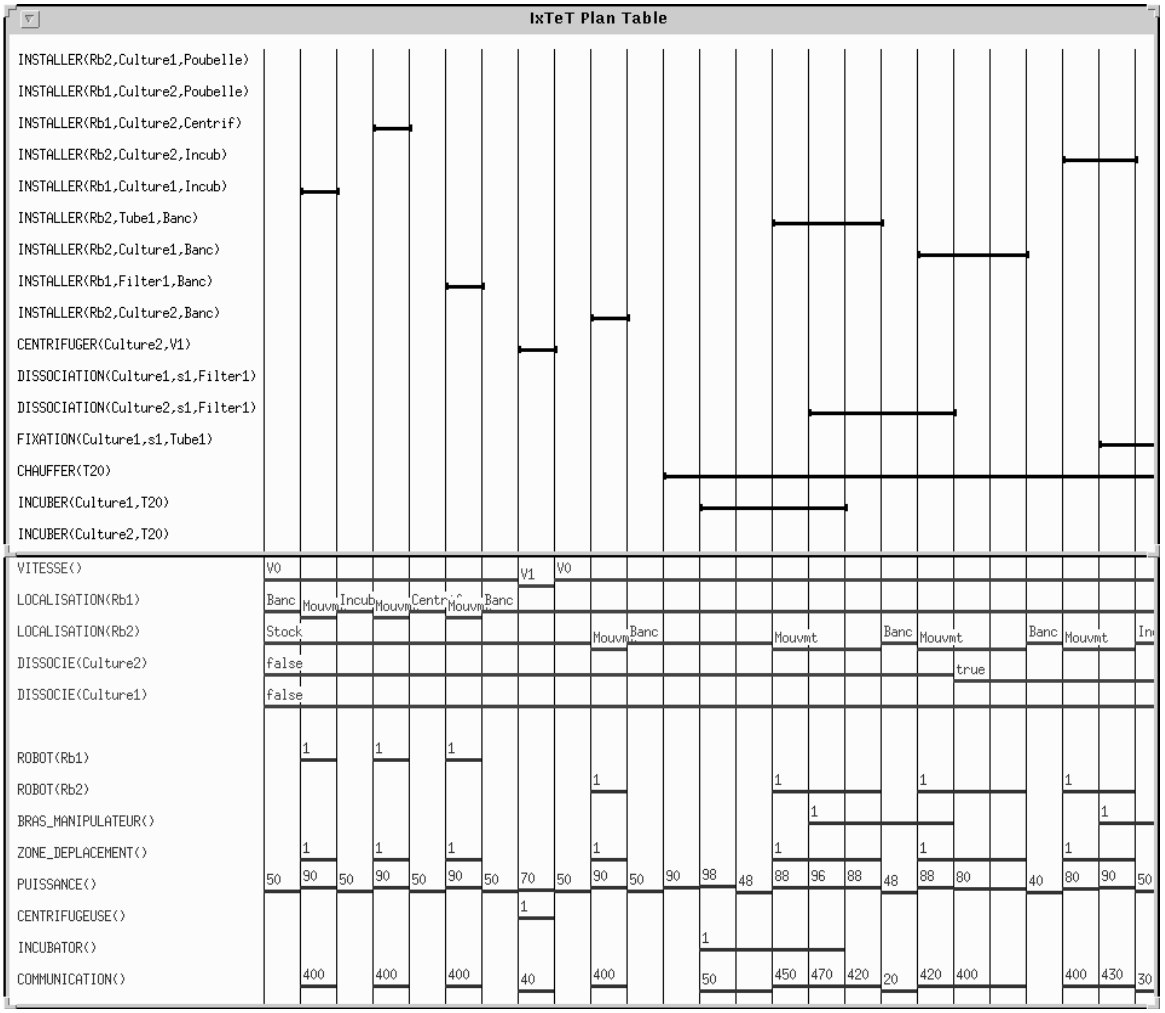


Figure 6.4: Effets du plan et cumul des ressources utilisées dans COLUMBUS

6.3 Autres exemples

Dans le but de donner au lecteur une idée de l'éventail de problèmes pouvant être modélisés et résolus avec IxTeT , nous décrivons succinctement dans ce paragraphe quelques autres domaines parmi ceux qui ont été implémentés. Les tests de performance que nous donnerons par la suite se réfèrent à ces domaines.

6.3.1 ATELIER

Il s'agit d'un exemple extrêmement classique en planification sur lequel ont été testés des planificateurs tels que (AB)TWEAK, PRODIGY, SNLP, STRIPS, TLPLAN, ou UCPOP.

C'est un domaine de planification et d'ordonnancement de machines dans un atelier de production. Un certain nombre de machines correspondent chacune aux différentes tâches envisageables dans le domaine. Ces tâches sont les suivantes : $POLIR$, $LAMINER$, $TOURNER$, $MEULER$, $PEINDRE\_VAPORISATEUR$, $PEINDRE\_IMMERSION$, $SOUDER$, $CHEVILLER$ et $POINCONNER$.

Chaque machine ne peut travailler que sur un objet à la fois à condition que la machine soit libre et que l'objet soit disponible. On veut alors planifier l'état final de certains objets. Diverses contraintes doivent être prises en compte ; par exemple le fait qu'une opération de laminage va rendre l'objet chaud et, comme certaines machines ne peuvent pas travailler dans ces conditions, il sera nécessaire d'attendre que l'objet refroidisse. Le système doit trouver un ensemble d'opérations satisfaisant les objectifs exprimés sur l'état final des objets (aspect, couleur, rugosité) tout en respectant les contraintes de ressources (machines, objets) et d'état (température, forme, etc ...).

Nous avons enrichi le domaine de base en considérant l'ensemble des techniciens qui commandent les machines comme une ressource partageable en quantité limitée.

A titre indicatif, notre système génère des plans pour l'usinage en parallèle de 10 objets comportant plus de 30 tâches en environ 1mn.

Sur les courbes de la figure 6.5, nous donnons, pour ce problème, l'évolution des temps passés à chaque nœud de l'arbre de recherche sur le chemin conduisant du scénario initial à la solution trouvée. Le temps passé à un nœud se décompose en celui passé dans chacun des modules d'analyse des défauts (sous-buts, menaces, ressources) et le temps passé à l'insertion de la résolvante choisie (et à la propagation des contraintes qui en découle).

On remarquera tout d'abord l'aspect en forme de pics des courbes (ceci est particulièrement vrai pour la courbe associée à l'analyse des sous-buts, avec un pic vers le nœud 70 et un autre autour du nœud 120). Ces pics sont dus aux transitions entre les différents états d'abstraction : lorsque l'on change d'état d'abstraction, cela signifie que tous les défauts liés à certains attributs $\widehat{att}_{résolu}$ ont été gérés et que l'on va maintenant considérer de nouveaux attributs $\widehat{att}_{nouveau}$. La remontée en temps lors du changement d'état est due à tous les défauts sur $\widehat{att}_{nouveau}$ qu'il faut maintenant considérer dans l'analyse.

On notera aussi comment entre les nœuds 100 et 150, synthèse de plans (gestion de sous-buts et menaces) et gestion de ressources sont menées de front. Cet exemple fait intervenir beaucoup d'attributs de ressources par rapport au nombre d'attributs d'état ; ceci explique pourquoi en fin de planification (après le nœud 140, il n'y a plus de défaut sur les attributs d'état : le plan est complètement généré), l'analyse des ressources semble un peu coûteuse (jusqu'à plus de 300ms/nœud) si on la compare par exemple à l'analyse des menaces (inférieur à 50 ms/nœud). Enfin, les temps liés à l'insertion et à la propagation des résolvantes présentent quelques pics (autour de 150 ms/nœud) qui sont liés à l'insertion des tâches.

6.3.2 ARIANE

Ce problème nous a été soumis par Matra Marconi Space. Il concerne l'ordonnancement de l'intégration des cases à équipements sur le lanceur Ariane. Celui-ci est composé de 8 cases à équipements destinées à recevoir les satellites. L'intégration d'une case donnée nécessite l'exécution d'une séquence de tâches ; ces tâches étant contraintes par des dates au plus tôt (arrivée des structures, de l'équipement) et des dates au plus tard (signatures de contrats, départ des bateaux pour Kourou). Chacune de ces tâches s'étale sur plusieurs jours de

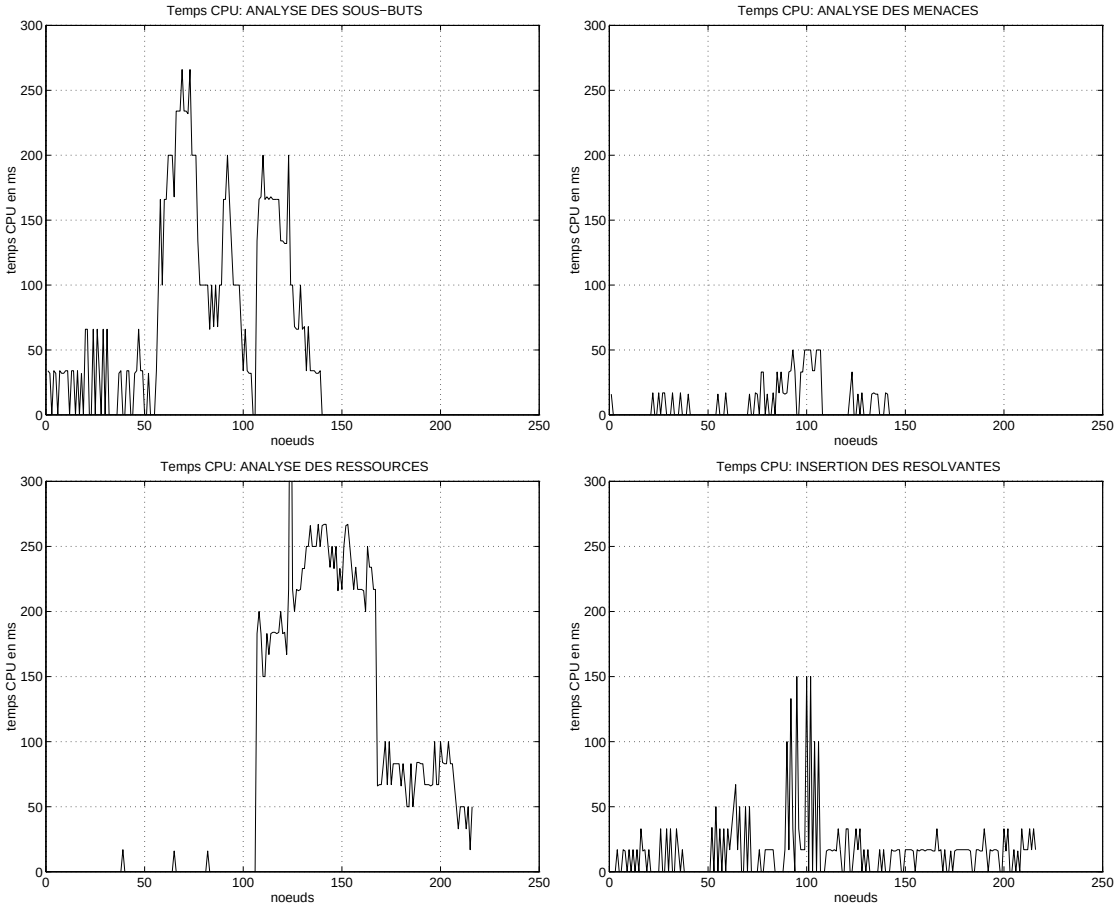
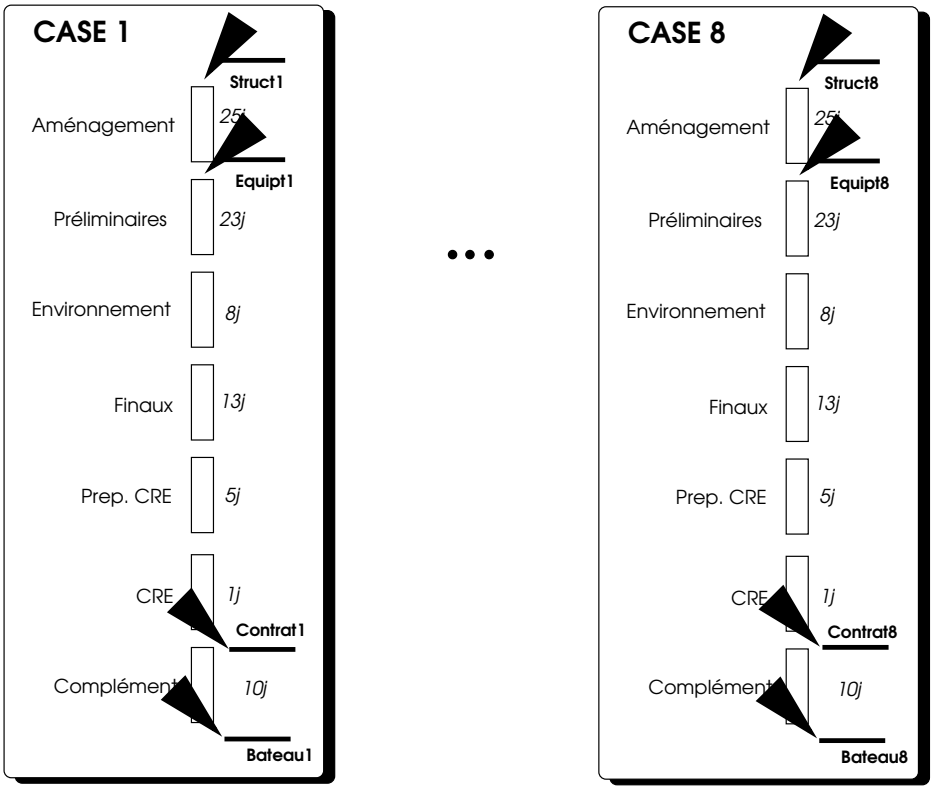


Figure 6.5: Evolution et répartition du temps passé dans les modules d'analyse

travail et nécessite diverses ressources humaines et moyens techniques. La figure 6.6 résume les tâches liées à l'intégration des cases à équipements ainsi que leurs durées. Les différentes ressources sont représentées dans le tableau avec un exemple d'utilisation de ces ressources par la tâche *Environnement*.

Toutes les données du problème, soient les 56 tâches correspondant aux 8 cases à intégrer, sont alors représentées avec leurs emprunts de ressources et leurs contraintes temporelles dans la tâche décrivant le scénario initial. La résolution consiste simplement à résoudre l'ensemble des conflits de ressources (ECM). Cela prend environ 4s à notre système en utilisant une hiérarchie (ordre sur les ressources à traiter) définie a priori par l'utilisateur (ici, l'on ne dispose pas de modèle de tâche pour générer automatiquement une hiérarchie). Cet exemple illustre comment notre approche peut résoudre des problèmes de pur ordonnancement.

La figure 6.7 présentée ci-dessous correspond à une recherche non-hiérarchisée. La courbe de gauche présente l'évolution du facteur de branchement le long de l'arbre de recherche sur le chemin allant du scénario initial (nœud 0) au plan solution trouvé (nœud 27). La croissance de la courbe est due à la stratégie d'opportunisme de la recherche : sont toujours résolus en priorité les défauts qui minimisent le facteur de branchement dans l'arbre de recherche. L'évolution de la fonction coût $f(N)$ associée à un plan partiel au nœud N est représentée sur la courbe de droite. Rappelons que cette fonction est la somme de $g(N)$, la combinaison de l'engagement associé à chacune des résolvantes qui ont été insérées depuis le nœud initial, et $h(N)$ l'estimation de l'engagement associé aux défauts qu'il reste à résoudre. On notera



RESSOURCES	CAPACITÉ MAX	tâche Environnement
Ajusteur	4	2
Électriciens	6	3
Qualité	3	1
Méca	1	0
Pot Vibrant	1	1
Bancs	2	0

Figure 6.6: Le domaine ARIANE

que l'heuristique h peut aussi bien être minorante que majorante. Ceci est dû au fait que la résolution d'un défaut par l'insertion d'une résolvante (qu'il s'agisse d'une contrainte ou d'une nouvelle tâche) peut aussi bien rendre plus difficile ou plus aisée (au sens de la mesure de l'engagement $engagt$ définie en §3.6.1) la résolution future d'un autre défaut donné.

6.3.3 FINITION PIECE

Le domaine $FINITION\ PIECE$ correspond à celui décrit dans [Missiaen 91] : un ensemble de robots (un robot-plombier, un robot-électricien et un robot-tapissier) doit effectuer les travaux de finition d'une pièce (cf. figure 6.8).

Le robot-plombier peut connecter les tuyauteries d'eau ($CONNECT\_EAU$) après que celles-ci aient été installées dans le mur ($INSERER\_TUYAUX$) par l'un quelconque des trois robots. L'installation de ces dernières demande de creuser des saignées dans les murs ($PREPARE\_SAIGNEE$) ce qui a pour effet de salir le mur. Le robot-plombier peut aussi ouvrir ($OUVRIR\_EAU$) ou couper ($COUPER\_EAU$) l'alimentation générale en eau. Le robot-tapissier encolle les laies ($ENCOLLER\_PAPIER$) en utilisant de la colle;

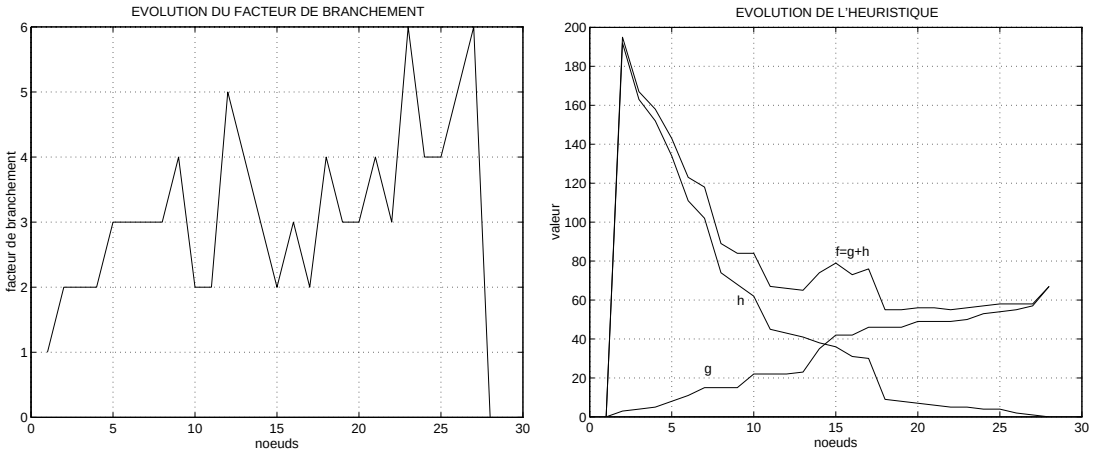


Figure 6.7: Facteur de branchement et heuristique le long de l'arbre de recherche

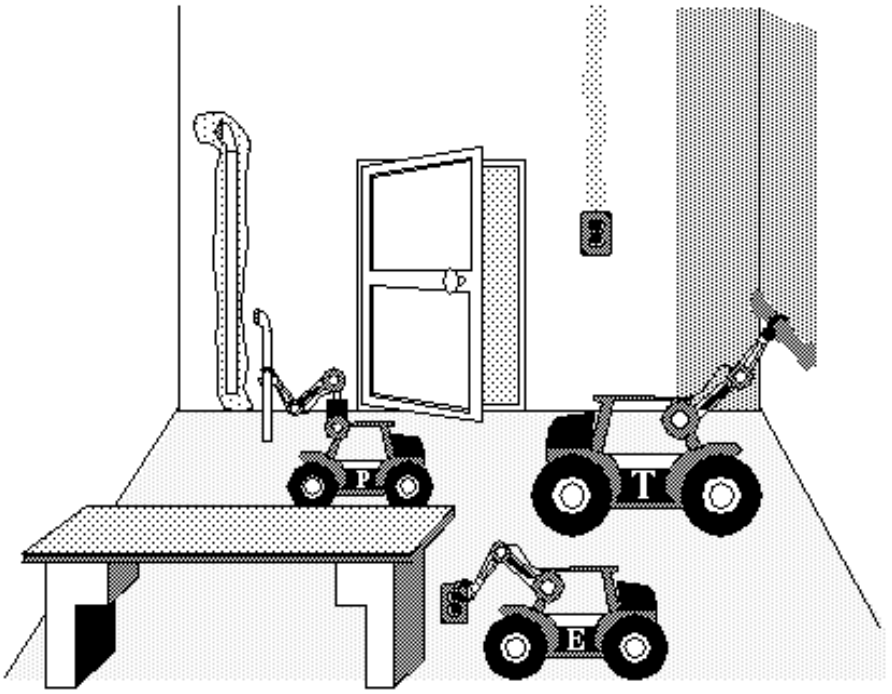


Figure 6.8: Le domaine FINITION PIECE

laquelle colle est une ressource consommable et productible ($FAIRE\_COLLE$). Il pose ensuite le papier sur les murs ($POSE\_PAPIER$) à condition que ceux-ci soient propres ($LAVER\_MUR$). Le robot-électricien connecte le circuit électrique ($CONNECT\_ELEC$) lorsque celui-ci est installé dans un mur ($INSERER\_CIRCUITS$). Le problème à résoudre consiste à tapisser les quatre murs de la pièce, à installer l'eau courante sur un mur et l'électricité.

Sur les courbes de la figure 6.9, nous présentons l'évolution de l'heuristique le long de l'arbre de recherche lié à la résolution de ce problème en utilisant la hiérarchie d'abstraction automatiquement générée à partir des tâches.

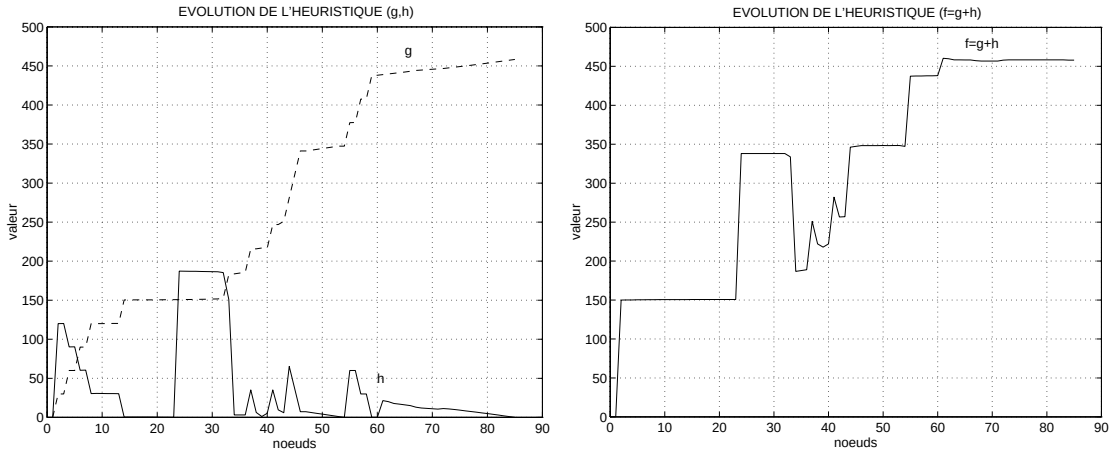


Figure 6.9: Heuristique avec hiérarchie d'abstraction

Ces courbes sont à comparer avec celle de la figure 6.7 où aucune hiérarchie n'est utilisée. On remarquera essentiellement le comportement en "dent de scie" de l'heuristique h qui est dû au fait qu'à un niveau d'abstraction donné, on fait précisément abstraction, dans le calcul de h, des défauts de plus bas niveau. Ce comportement en dent de scie de h induit sur la fonction d'estimation f un comportement en "plateaux", chaque plateau correspondant à un état d'abstraction. Le risque, avec cette heuristique moins informée, est de revenir en arrière sur des nœuds situés trop haut dans l'arbre de recherche dans la mesure où l'estimation du coût de ces nœuds-là, qui correspondent à des états d'abstraction élevés, ne prend pas en compte le travail qui devra être effectué dans les niveaux inférieurs.

6.3.4 GAMMES ALTERNATIVES

Soit le problème suivant[1] où il s'agit de produire cinq éléments A, B, C, D et E à l'aide de trois machines M1, M2 et M3 non-partageables. Chaque élément peut être produit de deux manières différentes selon l'ordre dans lequel il utilise les machines. Suivant la nature de l'opération à effectuer, la durée pendant laquelle une machine est utilisée est variable.

élément A :	M2(durée=3), M1(durée=3), M3(durée=6) ou bien
	M2(durée=3), M3(durée=6), M1(durée=3)
élément B :	M2(durée=2), M1(durée=5), M2(durée=2), M3(durée=7) ou bien
	M2(durée=2), M3(durée=7), M2(durée=2), M1(durée=5)
élément C :	M1(durée=7), M3(durée=5), M2(durée=3) ou bien
	M3(durée=5), M1(durée=7), M2(durée=3)
élément D :	M2(durée=4), M3(durée=6), M1(durée=7), M2(durée=4) ou bien
	M2(durée=4), M3(durée=6), M2(durée=4), M1(durée=7)
élément E :	M2(durée=6), M3(durée=2) ou bien
	M3(durée=2), M2(durée=6)

Ce problème est facilement modélisable avec IxTeT en définissant une tâche par gamme alternative dont l'effet est la production de l'élément et en spécifiant comme buts que tous

[1] Ce problème nous a été proposé par Pierre Lopez.

les éléments doivent être produits. On montre alors, en contraignant peu à peu la durée maximale du plan, que le plan donné sur la figure 6.10 (généré par IxTeT en 3s environ) est celui qui minimise la durée totale avec une durée minimale de 26 ; pour une durée strictement inférieure à 26, le planificateur explore tout l'arbre de recherche en 30s environ sans trouver de solution. Dans cet exemple, la *complétude* du système est exploitée pour montrer l'*optimalité* d'un plan généré.

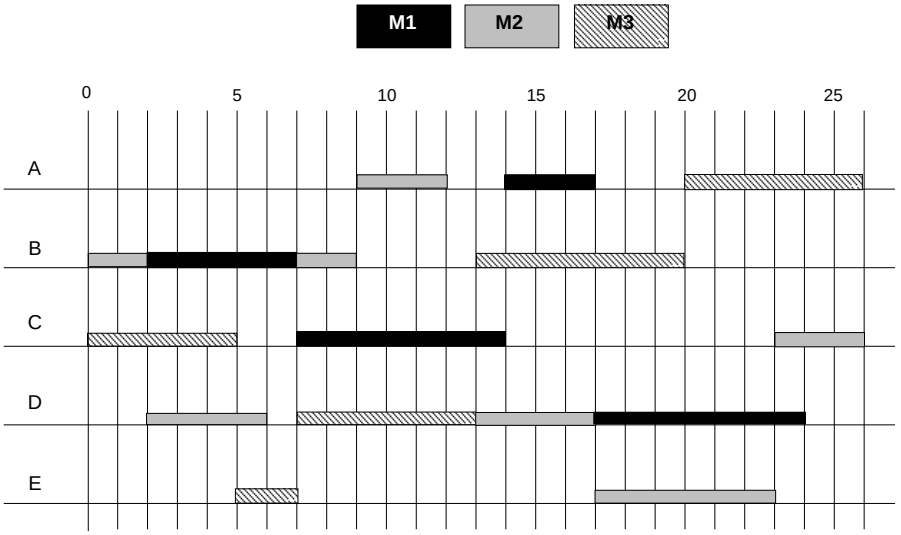


Figure 6.10: Exemple de plan solution

6.3.5 INTREPIDE

Dans cet exemple, un robot est chargé de la maintenance d'un laboratoire (livraison du courrier : $SAISIR\_COURRIER, DELIVRER\_COURRIER$), rechargement en papier des imprimantes, photocopieuses, télécopieurs : $RECHARGER$). On suppose que les rames de papier neuves se trouvent dans une salle spéciale et que le robot ne peut transporter que 3 rames de papier au maximum ($SAISIR\_PAPIER$).

Nous exprimons cette dernière contrainte par l'intermédiaire de deux ressources :

- $PAPIER\_SUR\_ROBOT$ qui est une ressource consommable (au moyen de la tâche $RECHARGER$) et productible (par la tâche $SAISIR\_PAPIER$) ; Cette ressource exprime la contrainte que lorsque le robot recharge une machine, il doit posséder assez de rames de papier ; et
- $TAILLE\_PORTE\_PAPIER$ qui représente le volume sur le robot limitant la quantité de papier que celui-ci peut porter. Cette ressource est bien entendu consommée par la tâche $SAISIR\_PAPIER$ et produite par la tâche $RECHARGER$. Elle permet d'exprimer la contrainte que, lorsque le robot saisit du papier au magasin, la quantité totale de rames sur le robot ne doit jamais excéder 3.

Le temps de 0.8 s que nous donnons dans le paragraphe suivant correspond à la résolution d'un problème comportant 3 courriers à délivrer et 4 machines à recharger. Le plan solution est alors constitué de 13 tâches.

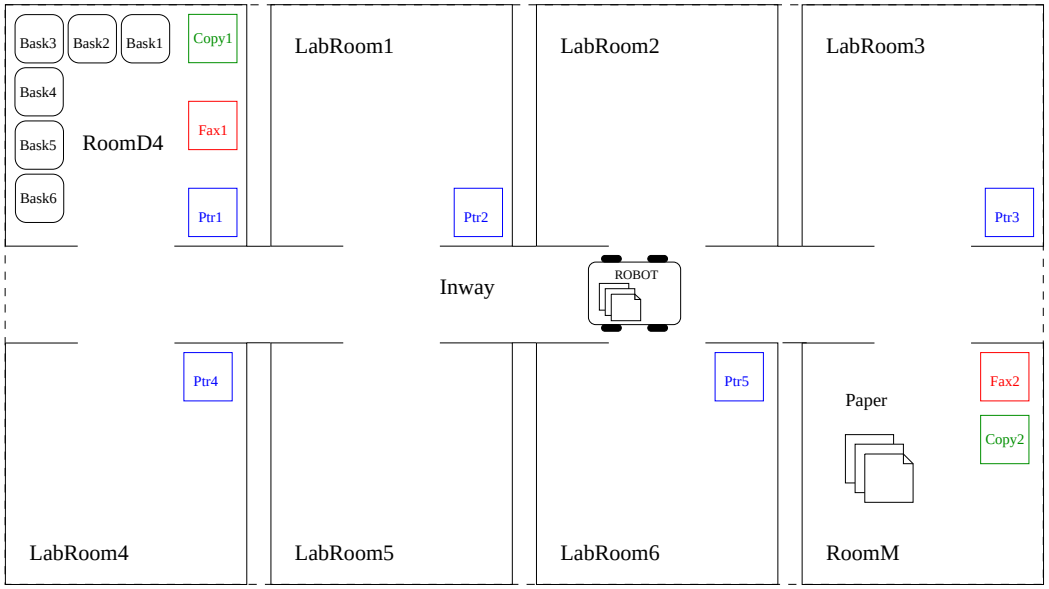


Figure 6.11: Le domaine INTREPIDE

Nous avons étudié comment se comportaient les performances du système lorsque l'on resserre les contraintes sur la durée maximale autorisée du plan solution. Cette étude est résumée par les deux courbes de la figure 6.12.

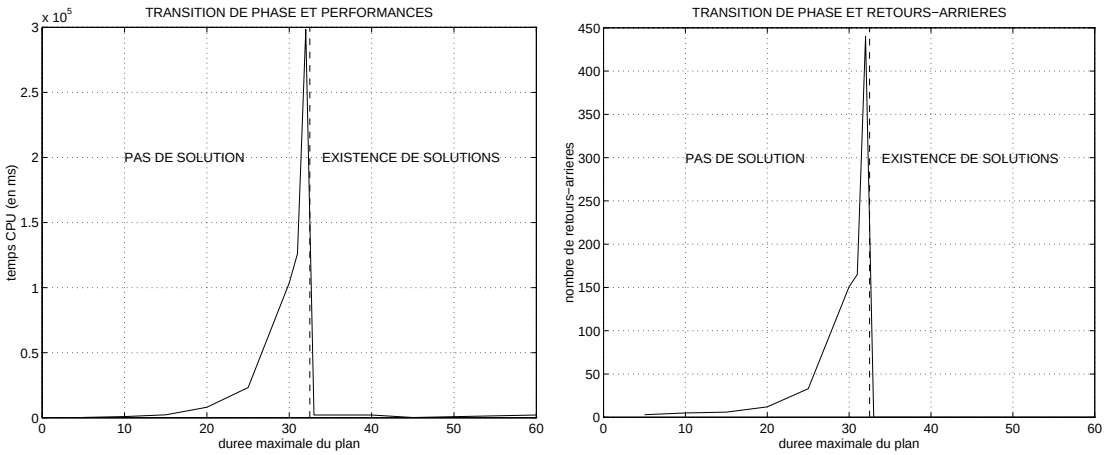


Figure 6.12: Transition de phase sur le domaine INTREPIDE

Lorsque le plan est très contraint (durée maximale inférieure à 20 mn), il est relativement facile de se rendre compte (après éventuellement l'insertion de quelques tâches), qu'aucune solution ne pourra être trouvée étant donné les contraintes temporelles. Lorsque la durée maximale est très grande, les contraintes sur la durée du plan importent peu, on rentre dans le cadre de la planification classique où le temps n'est pas explicitement géré. IxTeT , du fait de la gestion explicite d'une *métrique temporelle* permet d'explorer la zone de transition de phase où il est nécessaire de développer une grande partie de l'arbre de recherche (voire l'arbre de recherche en totalité) avant de répondre au problème de l'existence d'une solution. Nous avons pu réaliser ces courbes sur le problème décrit plus haut dans la mesure où la taille de l'arbre de recherche associé n'explose pas trop dans la zone de transition de phase.

Explorer cette zone pour des problèmes de grande taille reste à l'heure actuelle hors de portée de notre système. On notera toutefois que les problèmes réels que l'humain désire sous-traiter à la machine tombent en général dans deux catégories : soit des problèmes de grande taille mais situés hors de la zone de transition de phase (en général, il existe des solutions) ; soit des problèmes de taille plus restreinte mais très combinatoires dans la zone de transition.

6.4 Analyse des différentes stratégies utilisées

Nous avons mesuré, dans le contexte des domaines que nous venons de décrire, l'apport des trois principales stratégies décrites dans ce mémoire:

1. la stratégie de moindre engagement associée à une recherche opportuniste (cf. Chapitre 3) que nous opposons à une recherche où défauts et résolvantes sont choisis aléatoirement ;
2. le fait de gérer en concurrence les défauts liés aux états (sous-buts, menaces) et ceux liés aux ressources (ECM) (cf. Chapitre 3 et 4). Cette stratégie étant opposée à une gestion des ressources après synthèse complète du plan (approche classique) ; et
3. le fait d'utiliser ou non une hiérarchie d'abstraction telle que nous l'avons décrite au chapitre 5.

Ces trois stratégies sont bien entendu complémentaires : dans l'approche que nous avons décrite dans ce mémoire, la hiérarchie d'abstraction permet de structurer le problème et d'entrelacer synthèse du plan et gestion de ressources. La stratégie de recherche opportuniste est alors utilisée pour planifier au sein de chaque niveau d'abstraction donné. Dans ce sens-là, la hiérarchisation est une manière de structurer et d'encadrer la recherche opportuniste. Mais la réciproque est aussi vraie : du fait de l'aspect dynamique de la hiérarchie, que nous avons décrit au chapitre 5, c'est la stratégie de recherche opportuniste qui, au sein d'un niveau d'abstraction donné va déterminer quels défauts résoudre et donc, influencer la manière d'avancer parmi les différents états d'abstraction. Si bien qu'en final, les différents niveaux d'abstraction qui auront été parcourus sont fortement déterminés par l'opportunisme de la recherche.

Pour résoudre un problème donné, si b est le nombre de retours-arrière rencontrés au cours de la recherche, l la longueur du chemin allant de la racine à la solution et d la durée moyenne de développement d'un nœud, on peut estimer très grossièrement la durée du processus de planification par $D = b.l.d$.

La courbe 6.13 illustre le fait qu'une recherche opportuniste (au sens où nous l'entendons, c'est-à-dire vis-à-vis d'une stratégie de moindre engagement) permet d'améliorer la recherche, essentiellement en diminuant le nombre de retours-arrière b suite à un choix efficace des défauts et des résolvantes qui sont peu contraignants. La profondeur l de la solution ainsi que la durée de développement d'un nœud ne sont en général pas affectées par le moindre engagement[2] (lorsque le nombre de retours-arrière n'est pas réduit, les performances restent du même ordre).

[2]On note tout de même, quelquefois, un léger accroissement de la profondeur l suite à l'utilisation de la stratégie de moindre engagement ; ceci provenant du fait que l'insertion d'une résolvante peu contraignante a moins de chance de résoudre du même coup d'autres défauts du plan partiel.

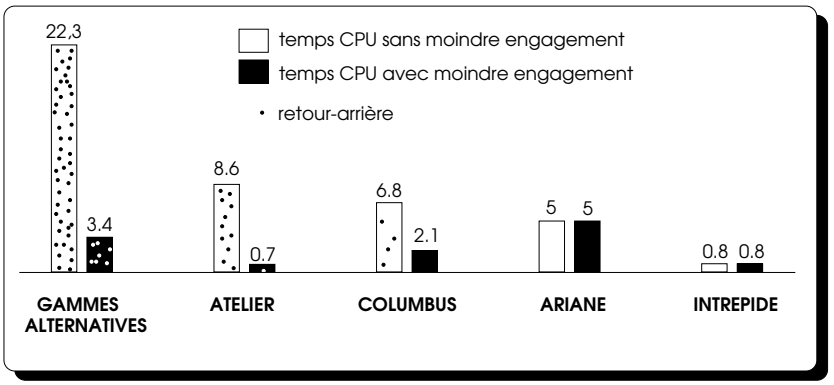


Figure 6.13: Gain apporté par la stratégie de moindre engagement

L'intégration des ressources au processus de synthèse de plans a un effet qui va dans le même sens que l'opportunisme de la recherche. Ceci n'est pas surprenant puisqu'une des raisons algorithmique principales ayant motivé l'intégration des deux processus est justement de pouvoir augmenter l'opportunisme de la recherche. La figure 6.14 présente quelques résultats. On remarquera aussi que dès qu'il existe des ressources productibles et que les tâches susceptibles de produire ces ressources peuvent intéragir avec les autres tâches du plan (comme c'est le cas dans le domaine $INTREPIDE$ suite à la gestion des ressources $PAPIER\_SUR\_ROBOT$ et $TAILLE\_PORTE\_PAPIER$), il devient *nécessaire* de gérer de front synthèse du plan et gestion des ressources.

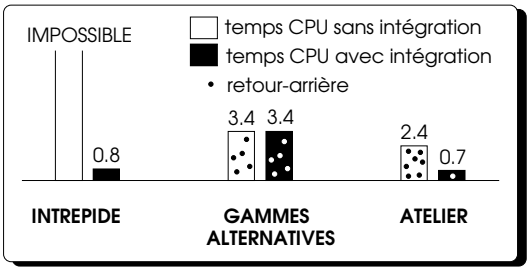


Figure 6.14: Gain apporté par une prise en compte hâtive des ressources

Les gains apportés par la hiérarchisation de la recherche, comme nous l'avons déjà vu, consistent principalement en une réduction des risques de retours-arrière (c'est-à-dire une diminution de b, cf. $FINITION\ PIECE$) associée à une accélération de la recherche à chaque nœud (diminution de d, cf. $ARIANE$ ou $COLUMBUS$). Nous présentons ces gains sur la figure 6.15.

6.5 Conclusion

Nous avons modélisé et testé de nombreux domaines sous IxTeT . Hormis ceux que nous avons décrit dans ce chapitre, citons aussi la planification dans le cadre d'un chantier de construction [Alami 92] ($CHANTIER$), la planification niveau tâche d'un robot d'exploration planétaire[3] ($IARES$) ainsi que d'un groupe de robots naviguant en intérieur ($HILARES$), la planification d'une activité de changement d'un pneu crevé sur une automobile ($PNEU$)

[3]Ce domaine a été modélisé dans le cadre du projet IARES avec le Centre National d'Etudes Spatiales.

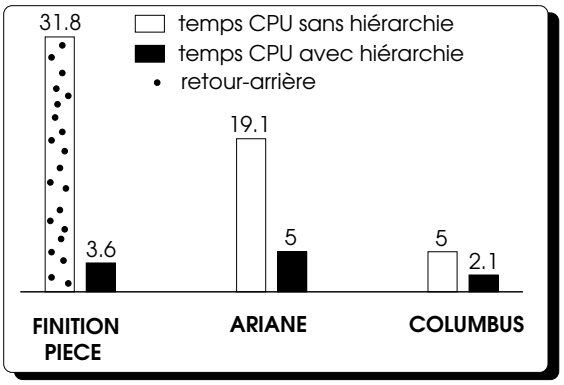


Figure 6.15: Gain apporté par la hiérarchisation de la recherche

et la manœuvre d'un voilier au cours d'une régate ($JACBY$). Ces exemples ont permis de valider, d'améliorer et de toucher les limites de notre représentation et de nos algorithmes.

La gestion explicite du temps et des ressources partageables ou non-partageables ainsi que l'utilisation d'attributs valués permet d'exprimer une large gamme de problèmes. Les principales limites de la représentation concernent :

- le besoin de *concision* et de *simplicité* pour décrire le domaine de planification. Ceci pourrait être amélioré par la conception d'un module d'aide à l'utilisateur pour formaliser son problème mais aussi par un enrichissement direct de la représentation, par exemple, en pouvant exprimer des axiomes du domaine ;
- la nécessité de pouvoir représenter des *contraintes* entre les variables atemporelles et les variables temporelles. Par exemple, la durée d'une tâche peut être fonction de ses arguments ; ou bien, certaines ressources sont telles que la quantité de ressource utilisée est liée à la durée d'utilisation de manière inversement proportionnelle.

Les stratégies permettant d'effectuer les choix non-déterministes se sont révélées performantes et complémentaires : la prise en compte des *ressources* au même titre que les états, la *gestion opportuniste* des défauts permettent une réduction des risques de retour-arrière tandis que la *hiérarchisation* de la recherche permet en plus de cela, une accélération de l'analyse du plan courant. Les améliorations envisageables consistent principalement en une sophistication des stratégies de retours-arrière en utilisant des techniques de retours-arrière guidés par les échecs ou en affinant davantage l'heuristique utilisée. A mesure que les performances de l'algorithme de planification s'améliorent, il serait aussi intéressant de se poser la question de la *qualité de la solution* produite. Nous avons insisté sur le fait que les stratégies utilisées conduisent en général à un plan de "bonne qualité" dans le sens de la compacité de celui-ci. Il conviendrait de formaliser cette notion de qualité en définissant un ou plusieurs critères de qualité du plan et en assurant l'optimalité (ou, de manière plus réaliste, une ϵ-optimalité) de la solution vis-à-vis de ce critère et ceci par l'utilisation de techniques classiques dérivées du "Branch-and-Bound" ou de techniques plus récentes de "Branch-and-Cut".

Conclusion générale et perspectives

Dans ce mémoire, nous avons montré comment il est possible de gérer en concomitance *synthèse de plans* et *gestion de ressources*.

Nos principales contributions dans ce domaine concernent les points suivants :

- un *enrichissement de la représentation* du système de planification afin de pouvoir représenter une large classe de ressources du domaine et leur utilisation par les tâches;
- une formalisation de la preuve de la *correction* et de la *complétude* du système global ainsi qu'une clarification des heuristiques et stratégies utilisées ;
- une procédure efficace de détection et de gestion des conflits de ressources potentiels basée sur la *théorie des graphes* ;
- une approche originale de hiérarchisation de la recherche basée sur la notion de *hiérarchie dynamique* et globalisant la stratégie de moindre engagement ; enfin,
- une *validation expérimentale* de notre approche générale implémentée en C++ sur le planificateur IxTeT .

Dans notre approche, le plan partiel courant consiste en un ensemble de propositions temporelles référençant des variables (temporelles ou atemporelles) et en un réseau de contraintes permettant de gérer des contraintes explicitement posées sur ces variables. Pour l'instant et pour des raisons de complexité, deux réseaux séparés gèrent, l'un les contraintes temporelles sur les instants, l'autre les contraintes sur les variables atemporelles. L'avantage est que ces deux réseaux spécifiques mettent en œuvre des mécanismes performants mais ceci s'effectue au détriment de l'expressivité du système. Il serait en effet très utile de pouvoir exprimer des contraintes hybrides *temps/variables* pour exprimer par exemple que la durée d'une tâche dépend de ses arguments. Un bel enrichissement de notre approche consisterait à gérer les quantités de ressources intervenant dans les propositions temporelles d'utilisations non plus comme des constantes mais comme des variables numériques. Cela permettrait entre autre de gérer des contraintes hybrides *temps/quantité de ressource utilisée* ou bien *variables (ressource utilisée)/quantité utilisée*.

D'autres enrichissements de la représentation seraient les bienvenus comme par exemple la possibilité de gérer des *effets dépendant du contexte* ou des *quantificateurs universels* comme cela est fait dans UCPOP. Le fait que UCPOP puisse être considéré comme une extension de l'algorithme SNLP milite en faveur du fait que nos algorithmes puissent eux aussi être étendus sans trop de mal dans la même direction. D'autres axiomes du domaine pourraient

aussi être pris en compte. D'une manière générale et d'après l'expérience de UCPOP, les notions de défauts et de résolvantes devraient pouvoir s'étendre suffisamment pour englober un formalisme plus expressif.

Nous avons évoqué le gain qui pourrait être apporté par des techniques de *filtrage du CSP défauts/résolvantes*. Ces techniques mériteraient d'être étudiées avec plus d'attention. Elles touchent en effet au cœur de la complexité du mécanisme de planification en se posant la question fondamentale de la nature des intéractions entre les résolvantes utilisées pour résoudre les défauts. Citons, dans le même ordre d'idée, l'approche *multi-contributeurs* consistant à choisir un ensemble disjonctif d'établisseurs pour une proposition non-expliquée donnée. Cette approche revient à ne pas s'engager au choix d'une résolvante mais à gérer la disjonction d'établisseurs en la filtrant au fur et à mesure que certains établissements sont incompatibles avec d'autres.

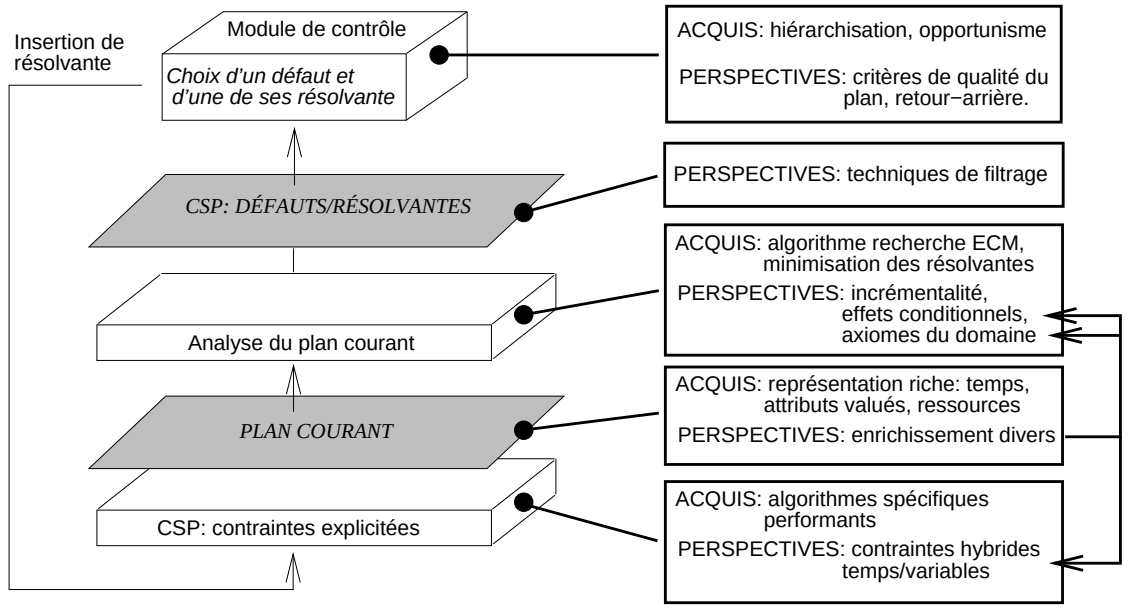


Figure 6.16: Acquis et perspectives

Enfin, ne perdons pas de vue le travail qui resterait à effectuer au niveau du contrôle de la recherche. Nous avons souligné, avec raison, que les plans solution étaient en général de "bonne qualité" au sens d'une parallélisation maximale du plan qui respecte les contraintes de ressources. Il faudrait maintenant formaliser un ou plusieurs critères de qualité d'un plan solution (de la même manière que des critères classiques sont définis pour les problèmes d'ordonnancement) et s'assurer de l'*optimalité* ou de la *quasi-optimalité* de la solution proposée par rapport à ce critère. Convenons toutefois que ce problème n'est pas aisé dans la mesure où jusque ici, pour des raisons de taille de l'arbre de recherche, les systèmes de synthèse de plans suivent tous une stratégie de type *profondeur d'abord* tandis que les problèmes d'optimisation sont essentiellement résolus par des recherches en *largeur d'abord*. Que l'on cherche ou non à optimiser un critère donné, se pose le problème du retour-arrière. L'implémentation d'un *retour-arrière intelligent* guidé par les échecs, stratégie séduisante s'il en est, est très difficile à mettre en œuvre en planification à cause de l'interdépendance quasi-totale entre tous les éléments du problème. Nous ne doutons pas toutefois qu'il y ait là une voie de recherche très intéressante.

Les acquis principaux et perspectives que nous venons de décrire sont résumés sur la figure

6.16.

Dans une perspective plus directement liée à la robotique, nous ne ferons qu'évoquer les problématiques de la *planification incrémentale* où de nouveaux objectifs doivent être intégrés à un plan déjà en cours d'exécution et la *fusion de plans* où il s'agit de coordonner plusieurs plans générés par des agents différents mais non-indépendants entre eux (à cause de l'utilisation de ressources communes à l'ensemble des agents par exemple). L'utilisation d' IxTeT dans ce cadre-là a déjà fait l'objet de travaux intéressants où est analysé le problème de l'interaction entre plusieurs plans d'actions ainsi que celui du retrait de certains objectifs non-prioritaires lorsque l'ensemble des objectifs ne peut être atteint [Chater 95].

Coder un domaine dans le formalisme IxTeT peut être un problème ardu pour qui n'est pas familier avec le système. L'implémentation d'une interface graphique serait un premier pas pour assister l'utilisateur dans ce travail. Il n'empêche que beaucoup de connaissances sont nécessaires pour coder un domaine et un problème de planification général ; ces connaissances doivent être transférées au système soit en dur dans le codage des algorithmes - on a alors un système dédié à une application donnée -, soit de manière plus déclarative sous forme de paramétrage d'un sytème indépendant du domaine comme c'est le cas dans IxTeT .

Enfin, nous concluerons avec le large débat que nous laissons ouvert sur la robustesse de notre approche vis a vis des problèmes d'incomplétude de la représentation.

Annexe A

Principales notations utilisées

$I_i = [a_i, b_i]$	intervalle de la droite réelle
T	ensemble d'instants
t, t_i	instants
u,u_i,v	intervalles temporels $u = [u^-, u^+]$
C_T	réseau de contraintes temporelles
V	ensemble de variables (atemporelles)
$?x$,$?x_i$,$?y$	variables
D,D_i	domaines de valeurs d'une variable
X,X_i,Y,V	valeurs d'une variable
C_V	réseau de contraintes sur des variables
$attS$	nom d'attribut d'état
$attR$	nom d'attribut de ressource
att	nom d'attribut quelconque
EVT,(EVT_{attS})	ensemble d'événements (sur un attribut d'état)
HLD,(HLD_{attS})	ensemble d'assertions
USE,(USE_{attR})	ensemble d'emprunts de ressources (sur un attribut de ressource)
$CONS$,$(CONS_{attR})$	ensemble de consommations de ressources
$PROD$,$(PROD_{attR})$	ensemble de productions de ressources
e, e_i	événement
$val^-(e), val^+(e)$	ancienne et nouvelle valeurs d'un événement
$t(e)$	instant d'un événement
n_e	nombre maximal d'événements liés à un attribut
h, h_i	assertion
$val(h)$	valeur d'une assertion
$t^-(h), t^+(h)$	instants de début et de fin d'une assertion
n_h	nombre maximal d'assertions liées à un attribut
r, r_i	emprunt de ressources
$t^-(r), t^+(r)$	instants de début et de fin d'un emprunt de ressources

$q(r)$	quantité de ressource utilisée (id. productions et consommations)
$\mathcal{B}, \mathcal{T}, \mathcal{P}_0, \mathcal{P}, \mathcal{P}_i, \mathcal{P}_S$	bases de connaissances IXTET (générale, tâche, scénario initial, plan partiel, plan solution)
$\mathcal{B} \leftarrow \mathcal{B}'$	$\mathcal{B}'$ supporte $\mathcal{B}$
$\underline{\mathcal{B}} \in INST(\mathcal{B})$	instance d'une base de connaissances
$NEX(\mathcal{P})$	propositions non-expliquées d'un plan partiel
$MNC(\mathcal{P})$	menaces d'un plan partiel
$ECM(\mathcal{P})$	ensembles critiques minimaux d'un plan partiel
$\Phi \in DEFAUTS(\mathcal{P}) = NEX(\mathcal{P}) \cup MNC(\mathcal{P}) \cup ECM(\mathcal{P})$	défaut d'un plan partiel
$\rho \in resolvantes(\Phi)$	résolvante d'un défaut
$\mathcal{P} \oplus \rho$	opérateur d'insertion d'une résolvante sur un plan
$COMPL(\mathcal{P})$	ensemble des complétions possibles d'un plan partiel
$INSTCOMPL(\mathcal{P})$	ensemble des complétions instanciées d'un plan partiel
$engagt(\mathcal{P}, \rho) \in [0, 1]$	engagement lié à l'insertion d'une résolvante
$opp(\mathcal{P}, \Phi) \in]0, 1]$	opportunité de la résolution d'un défaut
$\alpha = (t, t')$	contrainte de précédence
$\beta = [\alpha_1, ...]$	contrainte temporelle conjonctive
$\gamma = \{\beta_1, ...]$	contrainte temporelle disjonctive
$\Diamond \cap (U)$	possible intersection globale d'intervalles
$\neg\Diamond \cap (U)$	impossible intersection globale d'intervalles
$\Box \cap (U)$	nécessaire intersection globale d'intervalles
$\dot{\prec}$ ($\dot{\prec}_{P/S}$, $\dot{\prec}_{SI}$)	contraintes d'abstraction entre noms d'attributs
$\prec$ ($\prec_{P/S}$, $\prec_{SI}$)	fermetures transitives de $\dot{\prec}$
$<$	ordre partiel sur des classes d'attributs $\prec$-équivalents
(R, C)	état d'abstraction
L	niveau d'abstraction

Annexe B

Eléments de Théorie des Graphes

Dans cette annexe, nous passons en revue certaines familles de graphes évoquées dans le mémoire. Pour chacune de ces familles, nous donnons quelques propriétés fondamentales ainsi que des résultats de complexité algorithmique pour certains problèmes dont en particulier ceux liés à la recherche de cliques.

B.1 Quelques notions générales sur les graphes

Soit $G = \{V, E\}$ un graphe non-orienté.

Une **clique** de G est un sous-ensemble $U \subset V$ de sommets de G qui définit sur G un sous-graphe complet, c'est-à-dire tel que $\forall u_i, u_j \in U, \{u_i, u_j\} \in E$.

Une **clique maximale** de G est une clique de G maximale au sens de l'inclusion ensembliste.

Une **clique maximum** de G est une des cliques de G les plus grandes au sens de la cardinalité ensembliste.

Un **partitionnement en cliques** de G est une partition de V en p sous-ensembles $V_1, ..., V_p$ chaque ensemble V_i étant une clique de G. Un **partitionnement en cliques minimum** est un partitionnement en clique qui minimise le nombre p de sous-ensembles de la partition.

Un **ensemble de sommets indépendants** est un sous-ensemble $U \subset V$ de sommets de G dont aucune paire n'est reliée par une arête : $\forall u_i, u_j \in U, \{u_i, u_j\} \notin E$. Un **ensemble maximum de sommets indépendants** est un ensemble de sommets indépendants parmi les plus grands au sens de la cardinalité ensembliste.

Pour k entier, une **k-coloration** de G est une coloration des sommets de G c'est-à-dire une fonction $c : V \rightarrow [1, k] / \forall \{u_i, u_j\} \in E, c(u_i) \neq c(u_j)$.

Une **coloration minimale** de G est une k-coloration qui minimise k.

Une **coloration avec liste de couleurs** de G est un cas plus général que la $k - coloration$ où chaque sommet du graphe possède sa propre liste de couleurs possibles.

Sur un graphe quelconque, les problèmes de recherche de clique maximum (CLM), de recherche d'ensemble indépendant maximum (IM), de coloration minimale du graphe (COM), de recherche d'une coloration avec liste de couleurs (CLC) et de partitionnement minimum par un ensemble de cliques (PM) sont tous NP-complets. En outre, le nombre de cliques

maximales d'un graphe quelconque est, en pire cas, exponentiel en fonction du nombre de sommets du graphe.

Sur un graphe $G = (V, E)$, un **pseudo-cycle** est une séquence de sommets $(w_1, ..., w_m)$ telle que $\{w_m, w_1\} \in E$ et $\forall i \in [1, m-1], \{w_i, w_{i+1}\} \in E$. Un **cycle** est un pseudo-cycle dont les sommets sont tous disjoints : $\forall i, j \in [1, m],\ i \neq j \Rightarrow w_i \neq w_j$.

Une **corde** est définie comme une arête joignant deux sommets non-consécutifs d'un cycle.

B.2 Graphes faiblement triangulés

Un **graphe faiblement triangulé (G.F.T.)** est un graphe G tel que ni G, ni son complémentaire G^c ne possèdent de cycle de longueur strictement supérieure à 4 sans corde.

Les graphes faiblement triangulés ont jusqu'alors été principalement étudiés pour les problèmes de décomposition minimale de certains polygones complexes en une couverture par des polygones plus simples de type polygones convexes ou étoilés [Motwani 88].

Il a été montré dans [Hayward 85] que les G.F.T. sont des *graphes parfaits* c'est-à-dire des graphes pour lesquels le nombre chromatique (nombre minimal de couleurs suffisant à colorier le graphe) est égal à la taille des cliques maximales[1]. Sur les graphes parfaits, les problèmes CLM, IM, COM et PM peuvent être résolus en temps polynômial [Grötschel 81]. Malheureusement, la méthode de l'ellipsoïde décrite dans [Grötschel 81] est connue en programmation linéaire pour avoir de sérieux problèmes d'arrondi sur les réels et est difficilement utilisable en pratique. A l'heure actuelle, aucune méthode polynômiale purement combinatoire ne permet de résoudre ces problèmes sur les graphes parfaits.

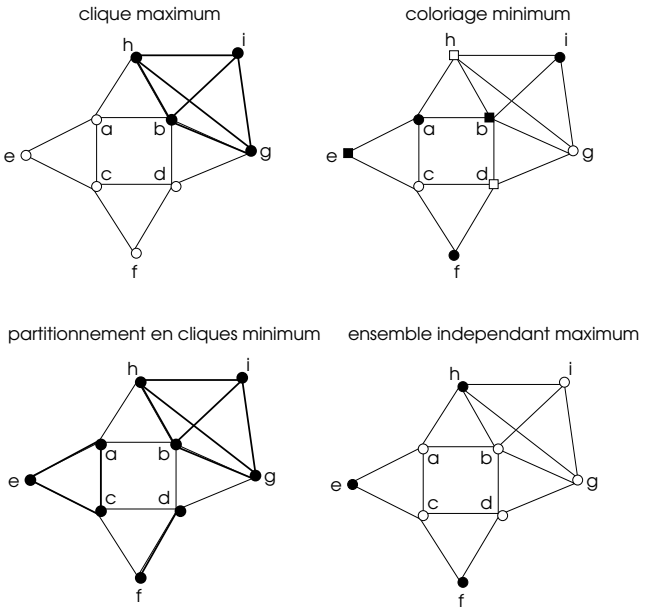


Figure B.1: Quelques problèmes sur un Graphe Faiblement Triangulé

De telles méthodes combinatoires existent toutefois dans le cas particulier des G.F.T. Dans [Raghunathan 89], l'auteur décrit des algorithmes combinatoires permettant de résoudre les

[1] Pour plus d'informations sur les graphes parfaits, on se reportera à [Berge 70, Golumbic 80].

problèmes CLM et COM en $O(e.v^2)$ et les problèmes IM et PM en $O(v^4)$ si v est le nombre de sommets et e le nombre d'arêtes. Des algorithmes pour les versions pondérées de ces problèmes sont donnés d'une complexité en $O(v^5)$.

B.3 Graphes triangulés

Un **graphe triangulé (G.T.)** (ou **graphe cordal**) est un graphe qui ne possède aucun cycle de longueur strictement supérieure à 3 sans corde.

On peut facilement montrer que les graphes triangulés sont un cas particulier de graphes faiblement triangulés (il suffit de montrer que le complémentaire d'un graphe triangulé ne possède pas de cycle de longueur strictement supérieure à 4 sans corde). Ceci est aisé en remarquant que le complémentaire de tout cycle sans corde de longueur strictement supérieure à 4 contient au moins un cycle sans corde de longueur 4 ou 5.

Historiquement, les graphes triangulés ont été les premiers graphes parfaits étudiés. Beaucoup de problèmes difficiles sur les graphes quelconques deviennent faciles sur les graphes triangulés. Les quatre problèmes évoqués plus haut (CLM, COM, IM et PM) sont solubles en $O(e+v)$ sur les graphes triangulés [Gavril 72].

Une propriété importante des graphes triangulés est que le nombre de cliques maximales est toujours borné par le nombre de sommets du graphe. Un algorithme de recherche de l'ensemble des cliques maximales d'une complexité en $O(v+e+deg)$, où deg désigne le nombre maximal de voisins d'un sommet v donné, est décrit dans [Tarjan 84]. Nous le résumons ci-dessous.

Soit $G=(V,E)$ un graphe. Si $U \subset V$ est un sous-ensemble de sommets de G, on note $G(U)$ le sous-graphe de G induit par U. Si v est un sommet de G, on notera $Adj(v)$ l'ensemble de ses voisins sur G. Un sommet $v \in V$ est dit **simpliciel** si l'ensemble constitué par $v \cup Adj(v)$ est une clique de G (c'est-à-dire si il existe une *unique* clique maximale contenant v). Un ordre $\sigma = (v_1, v_2, ..., v_n)$ est appelé une **combinaison parfaite** si et seulement si, pour $i = 1, ..., n$, v_i est simpliciel dans le graphe $G(v_1, v_2, ..., v_i)$.

Un résultat connu [Golumbic 80] est qu'un graphe G est triangulé si et seulement si il admet une combinaison parfaite σ.

Pour un graphe G triangulé, une combinaison parfaite $\sigma = (v_1, v_2, ..., v_n)$ peut être facilement calculée par un algorithme d'une complexité en $O(v+deg)$. Cet algorithme, décrit ci-dessous, consiste à indexer par $i+1$ le sommet qui a le nombre maximal de voisins déjà indexés dans $[1, i]$:

Procédure COMBINAISON_PARFAITE(V,E)
$SOMMETS\_INDEXES = \emptyset$
$indice\_courant = 0$
tantque $SOMMETS\_INDEXES \neq V$ **faire**
  $indice\_courant \leftarrow indice\_courant + 1$
  choisir dans $V \backslash SOMMETS\_INDEXES$ un sommet v qui maximise
    $card(Adj(v) \cap SOMMETS\_INDEXES)$
  $v_{indice\_courant} = v$
fintantque

Supposons construite une telle combinaison parfaite σ. Alors, on peut montrer que les ensembles $\{C_i\}_{i=1,...,n}$ définis par $C_i = \{v_i\} \cup (\ \{x_1, x_2, ..., x_i\} \cap Adj(x_i)\ )$ sont tous des cliques de G et contiennent en particulier l'ensemble des cliques maximales de G.

Calculer les ensembles $\{C_i\}_{i=1,...,n}$ et n'en retenir que les éléments maximaux au sens de l'inclusion ensembliste (c'est-à-dire précisément les cliques maximales de G) peut être effectué en $O(e)$ en parcourant les ensembles C_i dans le sens des i décroissants et en éliminant les C_i qui sont tels que $C_i \subset C_{i+1}$. La figure B.2 résume l'algorithme de recherche des cliques maximales sur un graphe triangulé particulier.

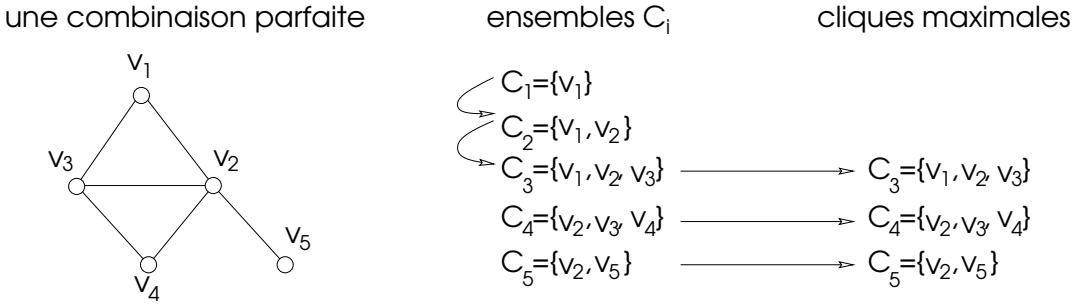


Figure B.2: Recherche des cliques maximales sur un graphe triangulé

B.4 Graphes de comparabilité

Un graphe $G = (V, E)$ est appelé **graphe de comparabilité** s'il est possible, en orientant les arêtes, d'en faire le graphe $(V, \prec)$ d'une relation d'ordre, c'est-à-dire avec :

$$v_i \prec v_j,\ v_j \prec v_k \ \Rightarrow\ v_i \prec v_k \quad (transitivité)$$

$$v_i \prec v_j \ \Rightarrow\ v_j \not\prec v_i \quad (antisymétrie)$$

Le graphe donné sur la figure B.3, n'est pas un graphe de comparabilité puisque, si on choisit initialement d'orienter $a \rightarrow b$ par exemple, il devient impossible d'orienter l'arête $\{e, f\}$ sans faire apparaître de nouveaux arcs par transitivité. Orienter initialement $b \rightarrow a$ conduit à une situation similaire.

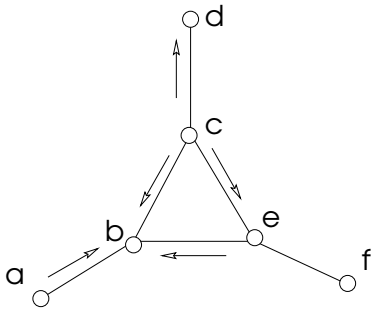


Figure B.3: Un graphe qui n'est pas de comparabilité

Les graphes de comparabilité sont des graphes parfaits. D'autre part, le complémentaire de tout graphe de comparabilité est un graphe faiblement triangulé.

Il existe une caractérisation structurelle des graphes de comparabilité : pour qu'un graphe $G = (V, E)$ soit un graphe de comparabilité, il faut et il suffit que, pour tout pseudo-cycle $(w_1, w_2, ..., w_{2q}, w_{2q+1})$ de longueur impaire, il existe une arête de la forme $\{w_i, w_{i+2}\}$ (en convenant que l'addition entre indices se fait modulo $2q + 1$).

Le graphe de la figure B.3 admet le pseudo-cycle $(a, b, c, d, c, e, f, e, b)$ de longueur impaire, et il n'existe pas d'arêtes de la forme sus-indiquée.

B.5 Graphes d'intervalles

Considérons, sur une droite, un ensemble $\mathcal{I} = \{I_1, I_2, ..., I_n\}$ d'intervalles. On peut former un graphe G, dont les sommets $i_1, i_2, ..., i_n$ représentent respectivement les intervalles $I_1, I_2, ..., I_n$, deux sommets étant joints si et seulement si les intervalles correspondant s'intersectent (cf. figure B.5). Un tel graphe est appelé **graphe d'intervalles** associé à l'ensemble d'intervalles $\mathcal{I}$.

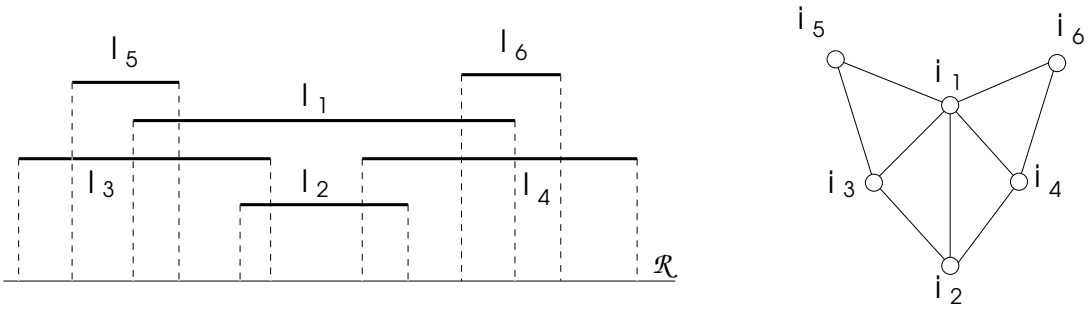


Figure B.4: Un ensemble d'intervalles et le graphe associé

Les graphes d'intervalles forment une sous-classe stricte des graphes triangulés. Le théorème suivant permet de caractériser les graphes d'intervalles :

Un graphe G est un graphe d'intervalles si et seulement si :

1. G est un graphe triangulé ; et
2. son complémentaire G^c est un graphe de comparabilité.

Dans [Korte 89], les auteurs décrivent un algorithme linéaire en $O(v + e)$ permettant de reconnaître si un graphe G donné est un graphe d'intervalles.

Le problème CLC est NP-complet même sur les graphes d'intervalles [Arkin 87].

B.6 Bilan

La figure ci-dessous résume les relations d'inclusion mutuelle entre les différentes classes de graphes évoquées dans cette annexe. Pour la plupart des classes, la complexité du problème de la recherche d'une clique maximum est donnée, ainsi que le nombre de cliques maximales.

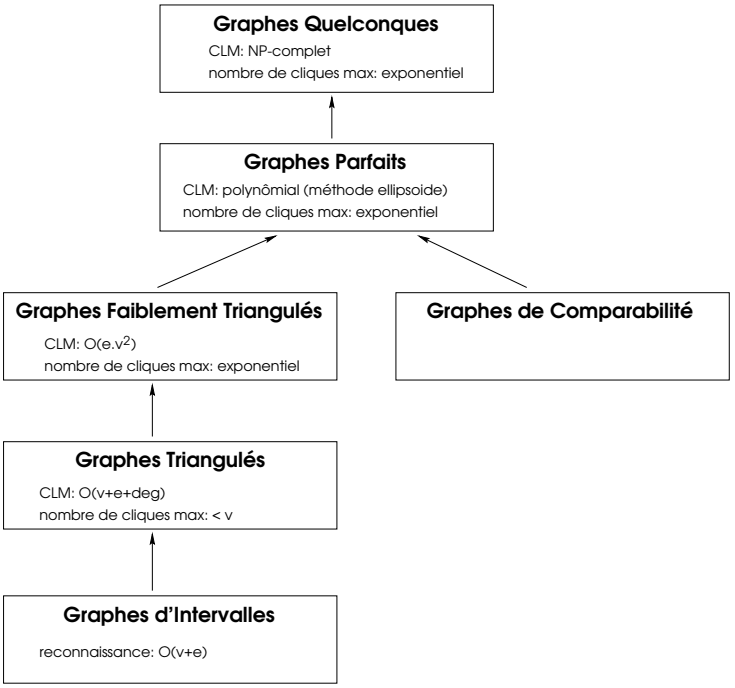


Figure B.5: Bilan

Annexe C

Eléments sur les C.S.P.

Nous donnons dans cette annexe quelques éléments de base liés aux problèmes de satisfaction de contraintes. La littérature dans ce domaine est très riche ; notre objectif est essentiellement d'en dégager quelques définitions et techniques très classiques auxquelles nous faisons référence dans ce mémoire. Pour une revue beaucoup plus complète sur les problèmes de satisfaction de contraintes, le lecteur pourra consulter [Jegou 91] ou bien [Bessière 92].

C.1 Définitions

Un problème de satisfaction de contrainte (CSP) se définit classiquement ([Montanari 74]) sous la forme d'un quadruplet $P = \{V, D, C, R\}$ où:

- $V = \{?x_1, ..., ?x_n\}$ est un ensemble de variables ;
- $D = D_1 \times ... \times D_n$ représente le produit cartésien du domaine de valeurs D_i de chacune des variables $?x_i$;
- $C = \{C_1, ..., C_m\}$ représente un ensemble de contraintes ; une contrainte C_k est définie par un ensemble de variables $\{?x_{k1}, ..., ?x_{kn_k}\}$; et
- un ensemble de relations $R = \{R_1, ..., R_m\}$ où R_k est l'ensemble des combinaisons de valeurs qui satisfont la contrainte C_k : il s'agit de la relation associée ; R_k est définie par un sous-ensemble du produit cartésien $D_{k1} \times ... \times D_{kn_k}$.

Dans la définition donnée ci-dessus, les relations R_k associées aux contraintes C_k sont données en extension. Dans les problèmes pratiques, ces relations sont plus souvent données en intension grâce à une fonction du type : $r_k(?x_{k1}, ..., ?x_{kn_k}) \in \{0, 1\}$ prenant la valeur 1 lorsque le n-uplet $(?x_{k1}, ..., ?x_{kn_k})$ vérifie la contrainte C_k et 0 s'il ne la vérifie pas.

Dans cette annexe, nous nous intéressons principalement aux CSP **binaires**, c'est-à-dire ceux pour lesquels les contraintes font uniquement intervenir deux variables. Les contraintes sont alors notées $C_{ij} = \{?x_i, ?x_j\}$. Un exemple de CSP binaire est donné sur la figure C.1 qui représente le problème des 4 reines [1].

[1] Rappelons que, de manière générale, le problème des n reines consiste à disposer n reines sur un échiquier $n \times n$ de manière à ce qu'aucune reine ne soit en position d'être prise par une autre.

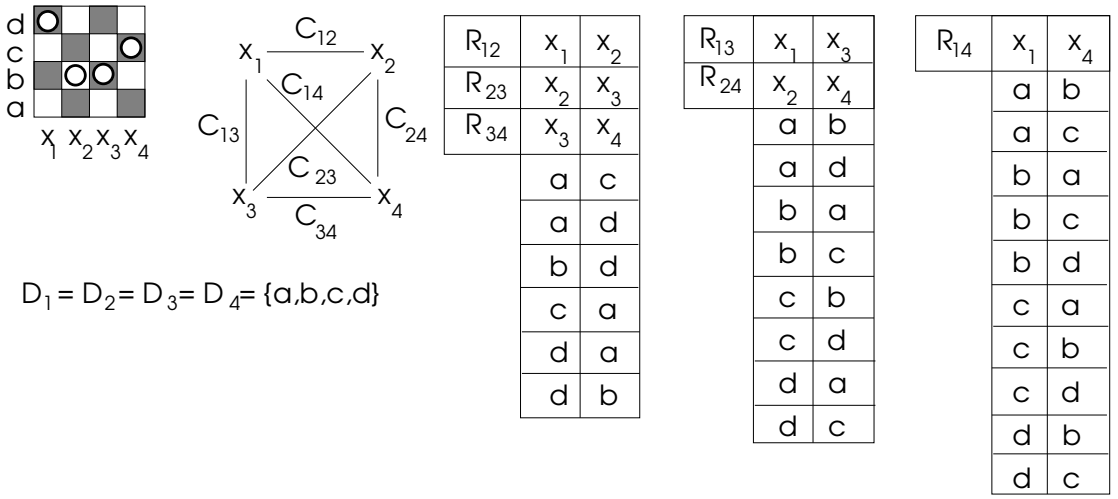


Figure C.1: Une formulation CSP du problème des 4 reines

Etant donné un CSP $P = \{V, D, C, R\}$, une **solution** de P est une instanciation consistance $(X_1, ..., X_n) \in D_1 \times ... \times D_n$ des variables de V qui vérifie toutes les contraintes c'est-à-dire :

$$\forall k \in [1, m], (X_{k1}, ..., X_{kn_k}) \in R_i$$

L'ensemble des solutions de P est noté sol_P. Un exemple de solution au problème des 4 reines est donné sur la figure ci-dessous.

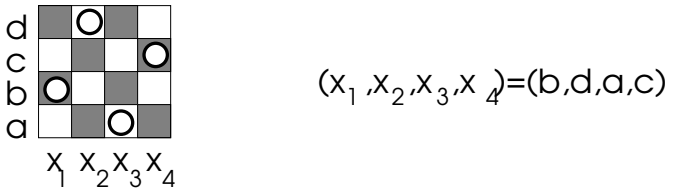


Figure C.2: Une solution au problème des 4 reines

Etant donné un CSP P, les principaux problèmes qui peuvent se poser sur P sont les suivants :

1. *existence d'une solution* : $sol_P \neq \emptyset$?
2. *recherche d'une solution* : trouver $d \in sol_P$.
3. *étiquetage minimal* : ôter du domaine D_i de chacune des variables $?x_i$ les valeurs qui ne participent à aucune solution.

De manière générale, toutes ces questions, même lorsque l'on se restreint à des CSP binaires sont difficiles (le problème d'existence d'une solution est NP-complet). La résolution d'un CSP se décompose généralement en deux phases : (i) une étape de pré-traitement (filtrage, propagation de contraintes) permettant de simplifier le problème à partir de considérations locales et (ii) une étape de recherche d'une solution par exploration d'un arbre de recherche grâce à une procédure basée sur le retour-arrière. En général, la première étape qui a pour objet de réduire la taille de l'arbre de recherche exploré par la seconde est de complexité polynômiale.

C.2 Filtrage et propagation de contraintes

C.2.1 Consistance d'arc

Etant donné un CSP $P = \{V, D, C, R\}$, un domaine D_i vérifie la **consistance d'arc** si et seulement si l'on a : $D_i \neq \emptyset$ et $\forall d_i \in D_i, \forall C_k \mid ?x_i \in C_k, d_i \in C_k[?x_i]$ où $C_k[?x_i]$ est la projection des tuples de C_k sur la variable $?x_i$. On dit que P vérifie la consistance d'arc si et seulement si chacun de ses domaines la vérifie.

En d'autres termes, un CSP vérifie la consistance d'arc si chacune des valeurs apparaissant dans les domaines D_i est susceptible de vérifier chacune des contraintes C_k portant sur $?x_i$. Le CSP de la figure C.1 vérifie cette propriété.

Un filtrage des domaines est associé à la notion de consistance d'arc : il s'agit d'éliminer du domaine des variables toutes les valeurs qui ne sont pas compatibles avec au moins une contrainte et de propager cette réduction du domaine. Ceci peut être effectué par l'algorithme AC-3 donné ci-dessous [Mackworth 77].

Procédure REVISE(k,i,j)
inconsistance $\leftarrow$ faux
pour $a \in D_i$ **faire**
  si $\nexists b \in D_j \mid (a,b) \in R_k$ **alors**
    $D_i \leftarrow D_i \backslash \{a\}$
    inconsistance $\leftarrow$ vrai
  finsi
finpour
retourne inconsistance

Procédure AC-3
$L \leftarrow \{(k,i,j) \mid C_k = \{?x_i, ?x_j\} \in C\}$
tantque $L \neq \emptyset$ **faire**
  choix puis suppression d'un (k,i,j) dans L
  si REVISE(k,i,j) **alors**
    $L \leftarrow L \cup \{(k',i,j') \mid C_{k'} = \{?x_i, ?x_{j'}\} \in C\}$
      $\cup \{(k'',i'',j) \mid C_{k''} = \{?x_{i''}, ?x_j\} \in C\}$
  finsi
fintantque

Un algorithme plus performant que AC-3, de complexité linéaire en fonction du nombre de contraintes est décrit dans [Mohr 86] : l'algorithme AC-4.

C.2.2 Consistances d'ordre supérieur

La consistance d'arc permet d'éliminer certaines valeurs du domaine des variables mais il ne s'agit que d'une consistance très locale et dans le cas général, un certain nombre de valeurs qui n'appartiennent à aucune solution du CSP ne sont pas éliminées.

La consistance d'arc a été généralisée sous la forme de la notion de **k-consistance** qui se définit, de manière informelle de la façon suivante: un CSP sera **k-consistant** si toute instanciation de $k-1$ variables qui satisfait les contraintes qui les relient peut être étendue

à toute $k^{ième}$ variable pour constituer une instanciation de k variables qui satisfasse les contraintes. Si $k = 2$, on retrouve la notion de consistance d'arc.

Pour les CSP binaires, la 3-consistance a particulièrement été étudiée sous l'appellation de **consistance de chemin**. Dans le filtrage qui lui est associé, il s'agit d'éliminer certains couples de valeurs (X_i, X_j) d'une relation R_{ij} dès qu'il existe une variable $?x_k$ qui, à cause des contraintes C_{ik} et C_{kj}, est incompatible avec l'instanciation $(?x_i, ?x_j) = (X_i, X_j)$.

Procédure PC-1
$R^n \leftarrow R$
répéter
  $R^0 \leftarrow R^n$
  pour $k \leftarrow 1$ **à** n **faire**
    pour $i \leftarrow 1$ **à** n **faire**
      pour $j \leftarrow 1$ **à** n **faire**
        $R_{ij}^k \leftarrow R_{ij}^{k-1} \cap (R_{ik}^{k-1} \circ R_{kj}^{k-1})$ [2]
      finpour
    finpour
  finpour
jusquà $R^n = R^0$
$R \leftarrow R^n$

PC-3, un algorithme plus évolué que PC-1 et décrit dans [Mohr 86] permet d'effectuer le filtrage par consistance de chemin en $O(n^3)$ si n désigne le nombre de variables du CSP. La figure C.3 représente le CSP correspondant au problème des 4 reines filtré par un algorithme de consistance de chemin.

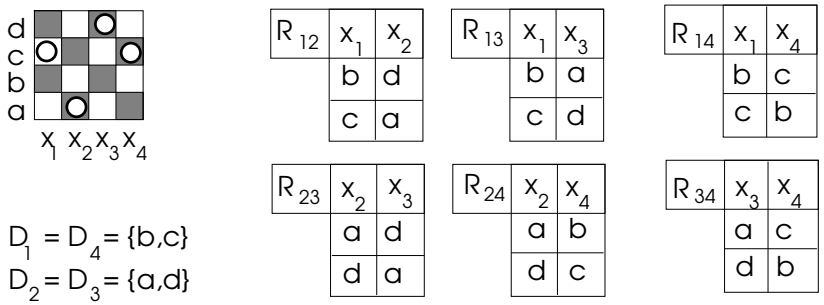


Figure C.3: Le CSP des 4 reines filtré par consistance de chemin

De manière générale, il existe des algorithmes de filtrage par k-consistance d'une complexité polynômiale en temps et en place en $O(n^k)$ [Cooper 90].

C.3 Stratégies d'exploration de l'arbre de recherche

En transformant le problème initial en un problème équivalent mais plus simple, les méthodes de filtrage évoquées ci-dessus permettent de réduire la taille de l'arbre de recherche d'une solution.

[2] La relation $\circ$ représente ici la composition de deux relations au sens des bases de données relationnelles.

La procédure de recherche proprement dite est décrite dans l'algorithme non-déterministe ci-dessous.

Algorithme RESOLUTION(P) $P = (V, D, C, R)$

a. si il existe un domaine vide : retourner(ECHEC)
b. sinon, si toutes les variables sont instanciées: retourner(P)
c. sinon
   1. sélectionner une variable $?x_i \in V$
   2. choisir une valeur $X \in D_i$ pour $?x_i$
   3. instancier $?x_i$ à la valeur X : $D_i \leftarrow \{X\}$
   4. filtrage dynamique : $P' \leftarrow$ FILTRAGE_DYNAMIQUE(P)
   5. appel récursif : RESOLUTION(P')

Dans l'algorithme nous employons le terme *choix* pour désigner un choix non-déterministe et par conséquent un point possible de retour-arrière par opposition à *sélection* qui correspond à un choix heuristique sur lequel un retour-arrière ne sera pas nécessaire.

La procédure de **filtrage dynamique** consiste à simplifier le CSP chaque fois qu'une décision d'instanciation d'une variable est prise. Une possibilité est par exemple d'appliquer un algorithme de filtrage par consistance d'arc de type AC-4. L'application d'un algorithme de filtrage dynamique sophistiqué permet d'élaguer l'arbre de recherche mais le fait que cet algorithme soit constamment appelé à chaque étape de la recherche peut aussi lourdement pénaliser les performances globales. Le compromis entre importance du filtrage dynamique et rapidité du développement de l'arbre de recherche est un point délicat et largement dépendant du type de problème.

Plusieurs stratégies de retours-arrière sont envisageables pour revenir sur le choix de la valeur allouée à une variable. Ces stratégies vont du simple retour-arrière chronologique aux retours-arrière guidés par les échecs. Une définition de ces différents types de retour-arrière ainsi qu'une comparaison théorique entre ceux-ci est effectuée dans [Kondrak 95].

On peut distinguer deux types de stratégies pour sélectionner la prochaine variable qui sera instanciée : les *ordres statiques* et les *ordres dynamiques*.

Un ordre statique est un ordre calculé hors-ligne avant l'exploration de l'arbre de recherche. Si l'on cherche à minimiser la taille de l'arbre de recherche, on pourra par exemple faire une sélection arbitraire de la première variable et, à chaque étape, sélectionner celle qui est liée par des contraintes au plus grand nombre de variables déjà sélectionnées (stratégie de *cardinalité maximale*). Une autre stratégie est d'ordonner les variables par ordre décroissant du nombre de contraintes qui pèsent sur elles (stratégie du *degré maximum*).

Les ordres dynamiques, calculés en-ligne au cours de la recherche, sont en général plus judicieux car plus opportunistes. Ils ne sont par contre pas toujours compatibles avec l'approche choisie pour effectuer les retours-arrière. Un ordre dynamique particulièrement intéressant consiste à toujours sélectionner la variable dont le domaine (après propagation et filtrage dynamique) possède le plus petit nombre de valeurs (stratégie de *réarrangement dynamique*).

En ce qui concerne le choix de la valeur à allouer à la variable sélectionnée, on peut par exemple considérer le *degré de contrainte d'une valeur*. Une valeur sera dite très contraignante si le nombre de valeurs qui sont compatibles avec elle dans les domaines des variables voisines est faible. Une stratégie classique consiste à choisir en priorité les valeurs les moins contraignantes.

Annexe D

Preuves

Théorème 1 *Soit $\mathcal{B}$ une base de connaissances temporelles $I_X T_E T$. Si toutes les propositions temporelles de $\mathcal{B}$ sont expliquées et si elle ne contient aucune menace, alors, $\mathcal{B}$ est nécessairement cohérente au niveau des états.*

Preuve :

Soit $\mathcal{B}$ une base de connaissances temporelles vérifiant les conditions du théorème.

Soit $\underline{\mathcal{B}} = \{EVT, HOLD, d_V, d_T\}$ une instance de $\mathcal{B}$ et $(t_0, t_1, t_2, ..., t_m, t_{m+1})$ l'ordre sur les instants associé. Soit $attS(X_1, ..., X_n)$ un attribut d'état instancié.

Nous allons montrer chacun des trois points intervenant dans la cohérence d'une instance. Pour des raisons de clarté de la preuve, nous y ignorons les "effets de bords" liés, pour un attribut instancié donné, aux instants extrémités.

(Point 1) (par l'absurde)

Supposons qu'il existe $i \in [1, m]$ tel que $|V^-(attS(X_1, ..., X_n), t_i)| \neq 1$.

(1.1) Si $|V^-(attS(X_1, ..., X_n), t_i)| > 1$, il existe au moins un couple (h, h') tel que : $(h, h') \in HOLD^-(attS(X_1, ..., X_n), t_i))$ et $V \neq V'$ si V et V' désignent respectivement les valeurs de h et h'.

h est donc une assertion de la forme $hold(attS(?x_1, ..., ?x_n) : ?v, (t^-, t^+))$ avec $d_V(?x_i) = X_i$, $d_V(?v) = V$ et $t^- < t_i \leq t^+$. De même, h' est de la forme $hold(attS(?x'_1, ..., ?x'_n) : ?v', (t'^-, t'^+))$ avec $d_V(?x'_i) = X_i$ $d_V(?v') = V'$ et $t'^- < t_i \leq t'^+$.

On a donc en particulier : $d_V(?x_i) = d_V(?x'_i)$, $d_V(?v) \neq d_V(?v')$, $t^- < t'^+$ et $t'^- < t^+$.

Or, par hypothèse, $\mathcal{B}$ ne contient pas de menace donc $< h, h' >$ n'en est pas une ; et la base contient donc au moins une contrainte parmi les suivantes : $(?x_1 = ?x'_1$ *et* ... *et* $?x_n = ?x'_n$ *et* $?v = ?v')$, $(t^+ < t'^-)$, $(t'^+ < t^-)$ ce qui est contradictoire avec l'existence de l'instance d_V.

(1.2) Si $|V^-(attS(X_1,...,X_n),t_i)| = 0$

Soit $t_{need}(i)$ le premier instant après t_i tel que $EVT(attS(X_1,...,X_n),t_{need}(i)) \neq \emptyset$, $HOLD^-(attS(X_1,...,X_n),t_{need}(i)) \neq \emptyset$ ou $HOLD^+(attS(X_1,...,X_n),t_{need}(i)) \neq \emptyset$. Soit $V = V^-(attS(X_1,...,X_n),t_{need}(i))$.

Par hypothèse, il existe un événement établisseur e_{est} à l'instant $t_{est}(i)$ qui établit la valeur V pour la proposition temporelle située à l'instant $t_{need}(i)$. On a nécessairement $t_{est}(i) < t_{need}(i)$ et $e_{est} \in EVT(attS(X_1,...,X_n),t_{est}(i))$. Donc, par construction de $t_{need}(i)$, on en déduit que $t_{est} < t_i$. D'après la définition d'une proposition expliquée, il existe un lien causal $h = hold(attS(X_1,...,X_n) : V,(t_{est}(i),t_{need}(i)))$. Or comme $t_{est}(i) < t_i \leq t_{need}(i)$, il s'ensuit nécessairement que $h \in V^-(attS(X_1,...,X_n),t_i)$. Ceci contredit $V^-(attS(X_1,...,X_n),t_i) = \emptyset$.

Ceci termine la preuve du premier point.

(Point 2)

Soit $attS(X_1,...,X_n)$ un attribut d'état instancié et t_i un instant quelconque. Il faut montrer que $V^+(attS(X_1,...,X_n),t_{i-1}) = V^-(attS(X_1,...,X_n),t_i)$.

De la même manière que ci-dessus, on peut construire pour cet attribut instancié $t_{need}(i)$ et $t_{est}(i)$. On aura alors : $t_{est}(i) \leq t_{i-1} < t_i \leq t_{need}(i)$ et il existe un lien causal
$h = hold(attS(X_1,...,X_n) : V,(t_{est}(i),t_{need}(i)))$.

Donc $V \in V^+(attS(X_1,...,X_n),t_{i-1})$ et $V \in V^-(attS(X_1,...,X_n),t_i)$. Comme on a montré, dans le premier point, que ces deux ensembles sont des singletons, le second point est démontré.

(Point 3)

Soit t_i un instant quelconque.

(3.1) Preuve que $|EVT(attS(X_1,...,X_n),t_i)| \leq 1$ (par l'absurde).

Supposons $|EVT(attS(X_1,...,X_n),t_i)| > 1$, et $e,e' \in EVT(attS(X_1,...,X_n),t_i)$ avec $e \neq e'$. Supposons $val^-(e) \neq val^-(e')$. Nécessairement, e et e' sont établis avec des liens causaux $h = hold(attS(X_1,...,X_n) : val^-(e),(t_j,t_i))$ et $h' = hold(attS(X_1,...,X_n) : val^-(e'),(t_{j'},t_i))$ avec $t_j < t_i$ et $t_{j'} < t_i$. Il est alors clair que $< h,h' >$ est une menace dans la base $\mathcal{B}$ ce qui est exclu.

Le cas $val^+(e) \neq val^+(e')$ est similaire en considérant ces deux événements non plus comme des propositions établies mais comme des établisseurs.

(3.2) Si $EVT(attS(X_1,...,X_n),t_i) \neq \emptyset$, soit e un événement de $EVT(attS(X_1,...,X_n),t_i)$.

Si e n'est pas le dernier événement sur $attS(X_1,...,X_n)$ alors, nécessairement, e explique le prochain événement e_{need} à l'instant t_{need} (en effet, si ce n'était pas le cas, e_{need} serait expliqué par un événement e' qui aurait lieu avant e et e romprait le lien causal entre e' et e_{need}). Il existe donc un lien causal $h^+ = hold(attS(X_1,...,X_n) : val^+(e),(t_i,t_{need}))$. Comme $h^+ \in HOLD^+(attS(X_1,...,X_n),t_i)$, on a : $val^+(e) \in V^+(attS(X_1,...,X_n),t_i)$ et même $V^+(attS(X_1,...,X_n),t_i) = \{val^+(e)\}$ à cause de l'unicité de la valeur montrée au point 1.

De manière symétrique, il existe un événement e_{est} à un instant $t_{est} < t_i$ établissant e, ainsi qu'un lien causal $h^+ = hold(attS(X_1, ..., X_n) : val^-(e), (t_{est}, t_i))$. On en déduit facilement que $V^-(attS(X_1, ..., X_n), t_i) = \{val^-(e)\}$.

Finalement, comme $val^-(e) \neq val^+(e)$, cela montre le résultat attendu :

$$V^-(attS(X_1, ..., X_n), t_i) \neq V^+(attS(X_1, ..., X_n), t_i)$$

□□□

Théorème 2 *Soit $\mathcal{B}$ une base de connaissances temporelles $I_X T_E T$.$\mathcal{B}$ est nécessairement cohérente au niveau des ressources si et seulement si $\mathcal{B}$ ne contient aucun ensemble critique minimal.*

Preuve :
Soit une base $\mathcal{B} = (USE, C_V, C_T)$ ne contenant aucun ECM (nous supposons que $\mathcal{B}$ ne contient pas de production et consommation moyennant la transformation Θ).

Soit d'autre part une instance $\underline{\mathcal{B}} = (USE, d_V, d_T)$ de $\mathcal{B}$ qui impose sur les instants de la base un ordre total $(t_0, t_1, ..., t_m)$.

Supposons que $\underline{\mathcal{B}}$ ne soit pas cohérente pour les ressources alors, il existe une ressource $attR(X_1, ..., X_n)$ et un instant t_i tels que :

$$Q_0(attR(X_1, ..., X_n)) + \sum_{k=1}^{k=i} \Delta(attR(X_1, ..., X_n), t_k) < 0$$

avec

$$\Delta(attR(X_1, ..., X_n), t_k) = \sum_{r \in USE^-(t_k)} q(r) \; - \sum_{r \in USE^+(t_k)} q(r)$$

Soit $U(t_i)$ l'ensemble des propositions d'emprunt de la ressource $attR(X_1, ..., X_n)$ dont l'intervalle temporel associé contient l'instant t_i :

$$U(t_i) =$$

$$\{use(attR(?x_1, ..., ?x_n) : q, (t_{i^-}, t_{i^+})) \in USE \mid \forall i \in [1, n], d_V(?x_i) = X_i \; et \; i^- \leq i < i^+\}$$

Nous allons montrer que $U(t_i)$ est un ensemble critique de $\mathcal{B}$; c'est-à-dire les points (1) et (2) de la définition d'un ECM.

(Point 1)

Observons tout d'abord que, étant donné la ressource $attR(X_1, .., X_n)$ et l'existence d'une instanciation d_V des variables de $\mathcal{B}$ qui instancie tous les attributs de ressources des propositions de $U(t_i)$ à $attR(X_1, .., X_n)$, il est clair que la compatibilité de la contrainte $\bigwedge_{j \in [1,n]}(?x_{1j} = ... = ?x_{kj} = ... = ?x_{mj} = X_j)$ avec C_V est assurée.

Pour $k \leq i$ donné, soit $B(t_k, t_i)$ l'ensemble des *use* de $USE^+(t_k)$ qui se terminent à ou avant t_i et $W(t_k, t_i)$ l'ensemble des *use* de $USE^+(t_k)$ qui se terminent strictement après t_i. Il est clair que $USE^+(t_k) = B(t_k, t_i) \cup W(t_k, t_i)$.

Sur $\underline{\mathcal{B}}$, nous avons

$$\sum_{k=1}^{k=i} \Delta(attR(X_1,...,X_n),t_k)$$

$$= \sum_{k=1}^{k=i} \sum_{r\in USE^-(t_k)} q(r) - \sum_{k=1}^{k=i} \sum_{r\in USE^+(t_k)} q(r)$$

$$= [\sum_{k=1}^{k=i} \sum_{r\in USE^-(t_k)} q(r) - \sum_{k'=1}^{k'=i} \sum_{r'\in B(t_{k'},t_i)} q(r')] - \sum_{k=1}^{k=i} \sum_{r\in W(t_k,t_i)} q(r)$$

$$= \sum_{k=1}^{k=i} \sum_{r\in W(t_k,t_i)} q(r)$$

$$= \sum_{r\in U(t_i)} q(r)$$

En effet, la quantité entre crochets s'annule car chaque emprunt de ressource r se terminant à t_k (i.e. appartenant à $USE^-(t_k)$) débute nécessairement avant t_i; il existe donc un unique $t_{k'}$, qui n'est autre que le début de r, tel que $r \in B(t_{k'},t_i)$. Réciproquement, pour chaque emprunt r appartenant à $B(t_{k'},t_i)$, comme par définition de cet ensemble r se termine avant t_i, il existe un unique instant t_k (la fin de r) tel que $r \in USE^-(t_k)$.

Le passage à la dernière égalité s'explique par le fait que $U(t_i) = \bigcup_{k\leq i} W(t_k,t_i)$.

Du fait du choix de t_i comme un instant de $\underline{\mathcal{B}}$ où la ressource $attR(X_1,..,X_n)$ est sur-utilisée, nous avons donc :

$$Q_0(attR(X_1,...,X_n)) + \sum_{r\in U(t_i)} q(r) < 0$$

ce qui termine la preuve du premier point.

(Point 2)

Sur l'instance $\underline{\mathcal{B}}$ nous avons, par construction de $U(t_i)$, $\forall r \in U(t_i), t^-(r) \leq t_i < t^+(r)$. On peut donc facilement en déduire que $\forall r,r' \in U(t_i), t^-(r) \leq t_i < t^+(r')$ et donc, que la contrainte $\bigwedge_{r,r'\in U(t_i)}(t^-(r) < t^+(r')$ est compatible avec les contraintes temporelles C_T de $\mathcal{B}$.

(Fin de la preuve)

$U(t_i)$ est donc un ensemble critique.

Pour montrer l'existence d'un ensemble critique minimal, il suffit d'extraire de $U(t_i)$ un sous-ensemble V parmi les plus petits au sens de l'inclusion ensembliste tels que :

$$Q_0(attR(X_1,...,X_n)) + \sum_{r\in V} q(r) < 0$$

Pour ce qui est de la réciproque, il est clair que si B contient un ensemble critique minimal, U, alors, on peut construire une instance $\underline{\mathcal{B}}$ où les intervalles de U s'intersectent et utilisent la même ressource $attR(X_1,...,X_n)$. $\underline{\mathcal{B}}$ est alors incohérente pour cette ressource.

□□□

Théorème 3 *L'algorithme de recherche d'IXTET est sain : tout plan retourné est un plan solution.*

Preuve :

Soit $\mathcal{P}_0, \{\mathcal{T}_1, ..., \mathcal{T}_m\}$ un problème initial. Supposons que l'algorithme $PLANIFIER$ s'arrête en retournant un plan $\mathcal{P}_s$. Nous allons montrer les trois points de la caractérisation d'un plan solution (cf. §3.4).

(Point 1)

Nécessairement, $\mathcal{P}_s$ est consistant (d'après le point 1 de l'algorithme : un plan n'est retourné que s'il est consistant).

(Point 2)

De plus, il existe une liste de résolvantes (ρ_1,...,ρ_n) telle que $\mathcal{P}_s = \mathcal{P}_0 \oplus \rho_1 \oplus ... \oplus \rho_n$. Or, vu la définition de l'opérateur d'insertion $\oplus$, il est clair que pour toute résolvante ρ et toute base de connaissances $\mathcal{P}$ on a : $\mathcal{P} \leftarrow \mathcal{P} \oplus \rho$. On en déduit donc par transitivité immédiate que : $\mathcal{P}_0 \leftarrow \mathcal{P}_s$.

(Point 3)

Vu la nature des résolvantes ρ, les seuls changements du monde insérés à $\mathcal{P}_I$ en cours de planification (grâce aux propositions temporelles *event*, *use*, *consume* ou *produce*) sont ceux liés à l'insertion d'une nouvelle tâche au plan partiel pour expliquer une proposition d'état ou produire une ressource. Le troisième point de la caractérisation d'un plan solution est donc bien vérifié.

Finalement, $\mathcal{P}_s$ est bien solution du problème $\mathcal{P}_0, \{\mathcal{T}_1, ..., \mathcal{T}_m\}$.

□□□

Théorème 4 *L'algorithme de recherche d'IXTET est complet au sens restreint : s'il existe une solution au problème de planification, l'algorithme s'arrêtera au bout d'un temps fini et retournera un plan solution.*

Preuve :

Pour montrer la complétude nous avons besoin dans un premier temps d'utiliser le lemme suivant dont la preuve est décrite plus bas:

Lemme 1 *Soit $\mathcal{P}$ un plan partiel contenant au moins un défaut et admettant une solution instanciée $\underline{\mathcal{P}_s}$. Soit $\Phi \in DEFAUTS(\mathcal{P})$. Alors, il existe au moins une*[1] *résolvante $\rho \in resolvantes(\Phi)$ telle que $\mathcal{P}_s$ est aussi une solution de $\mathcal{P} \oplus \rho$.*

[1] Pour un algorithme de planification respectant la propriété de *systématicité* tel SNLP, cette résolvante est unique. Dans la version que nous donnons du calcul des résolvantes, qui a été implémentée sur IXTET, cette propriété n'est pas respectée.

Supposons ce lemme montré. Soit $\mathcal{P}_0, \{\mathcal{T}_1, ..., \mathcal{T}_m\}$ un problème de planification possédant au moins une solution $\mathcal{P}_s$. Nous allons montrer que l'algorithme $PLANIFIER$ d'IxTeT va alors nécessairement trouver une solution $\mathcal{P}'_s$ (qui ne sera d'ailleurs pas nécessairement $\mathcal{P}_s$, puisque nous ne montrons qu'une complétude au sens restreint).

Soit $\underline{\mathcal{P}_s} \in INST(\mathcal{P}_s)$ une instanciation de $\mathcal{P}_s$. Il est clair que $\underline{\mathcal{P}_s}$ est aussi une solution du problème initial. L'idée de la preuve consiste à guider, grâce à l'application du lemme avec la solution $\underline{\mathcal{P}_s}$, l'algorithme non-déterministe vers une solution.

Si $\mathcal{P}_0$ est consistant alors, c'est une solution au problème retournée par l'algorithme ; sinon, soit $\Phi_1 \in DEFAUTS(\mathcal{P}_0)$ un défaut quelconque sélectionné. D'après le lemme, il existe une résolvante ρ_1 de Φ_1 telle que $\underline{\mathcal{P}_s}$ est solution de $\mathcal{P}_0 \oplus \rho_1$. Choisissons cette résolvante pour Φ_1.

Si nous réappliquons le lemme de manière récursive au nouveau problème $\mathcal{P}_0 \oplus \rho_1$ et ainsi de suite, nous obtenons une suite de résolvantes $(\rho_1, ..., \rho_i)$ telle que $\underline{\mathcal{P}_s}$ est solution de $\mathcal{P}_i = \mathcal{P}_0 \oplus \rho_1 \oplus ... \oplus \rho_i$. Il est clair que d'après le lemme, l'algorithme ne peut pas retourner d'échec sur le chemin $(\rho_1, ..., \rho_i)$; seuls deux cas peuvent se présenter : soit la suite de résolvante est infinie, soit elle est finie et alors l'algorithme finira par retourner un plan $\mathcal{P}_n$ sans défaut c'est-à-dire un plan solution. La fin de la preuve consiste à montrer que cette suite est finie.

On a $\mathcal{P}_0 \leftarrow \mathcal{P}_1 \leftarrow ... \leftarrow \mathcal{P}_i \leftarrow ... \leftarrow \underline{\mathcal{P}_s}$.

Pour un plan partiel $\mathcal{P} = \{EVT, HLD, USE, C_T, C_V\}$ donné, estimons sa taille par $sz(\mathcal{P}) = |EVT| + |HLD| + |USE| + nb\_ct\_précédence(\mathcal{P}) + nb\_ct\_variables(\mathcal{P})$ où $nb\_ct\_précédences$ est le nombre total de contraintes temporelles de précédences pouvant être déduites de C_T et $nb\_ct\_variables$ le nombre de contraintes d'égalité $(=)$ et de différence $(\neq)$ pouvant être déduites des contraintes exprimées sur les variables atemporelles.

Il est clair que pour un plan partiel $\mathcal{P}$ donné, $sz(\mathcal{P})$ est une quantité bornée. D'autre part, si $\mathcal{P} \leftarrow \mathcal{P}'$ alors $sz(\mathcal{P}) \leq sz(\mathcal{P}')$.

De plus il est facile de se convaincre en examinant chaque type de résolvante que si ρ est une résolvante d'un défaut de $\mathcal{P}$ alors $sz(\mathcal{P}) < sz(\mathcal{P} \oplus \rho)$.

Donc, la liste d'entiers $sz(\mathcal{P}_i)$ est strictement croissante et bornée par $sz(\underline{\mathcal{P}_s})$. D'où le fait qu'elle soit finie, ce qui termine la preuve du théorème.

Preuve du lemme 1

Nous allons montrer le lemme pour chacun des 3 types de défauts Φ d'un plan partiel.

(Cas 1) $\Phi \in NEX(\mathcal{P})$

Nous ne traitons ici que le cas d'une explication d'un événement ; la démonstration dans le cas de l'explication d'une assertion est strictement identique.

Soit $e = event(attS(?x_1, ..., ?x_n) : (?v, ?), t)$ l'événement de $\mathcal{P}$ à expliquer.

Dans $\underline{\mathcal{P}_s}$, e est instancié en $\underline{e} = event(attS(X_1, ..., X_n) : (V, ?), t)$ et est expliqué par un événement $\underline{e}_{est} = event(attS(X_1, ..., X_n) : (?, V), t_{est})$ avec $t_{est} < t$; de plus, il existe une assertion $hold(attS(X_1, ..., X_n) : V, (t_{est}, t))$.

Deux cas sont à distinguer :

- si $e_{est} \in \mathcal{P}$ alors, soit ρ la résolvante consistant à établir e par e_{est} grâce à un lien causal. Si dans le plan partiel $\mathcal{P}$ on a $e_{est} = event(attS(?x'_1, ..., ?x'_n) : (?v_{est}, ?), t_{est})$, elle consiste à rajouter au plan : $?x'_1 = ?x_1, ..., ?x'_n = ?x_n, ?v_{est} = ?v, t_{est} < t$ et $hold(attS(?x_1, ..., ?x_n) : ?v, (t_{est}, t))$. Il est alors clair que si $\underline{\mathcal{P}_s}$ est une solution de $\mathcal{P}$ ($\mathcal{P} \leftarrow \underline{\mathcal{P}_s}$) alors on a $\mathcal{P} \oplus \rho \leftarrow \underline{\mathcal{P}_s}$ et $\underline{\mathcal{P}_s}$ est une solution de $\mathcal{P} \oplus \rho$;
- si $e_{est} \notin \mathcal{P}$, dans $\underline{\mathcal{P}_s}$, par définition d'un plan solution, il existe une tâche $\mathcal{T}$ insérée au plan qui contient e_{est}. Il est clair que l'insertion de cette tâche $\mathcal{T}$ fait partie des résolvantes de Φ sur le plan partiel $\mathcal{P}$. Soit ρ cette résolvante, alors de manière similaire au point précédent, il est aisé de voir que $\mathcal{P} \oplus \rho \leftarrow \underline{\mathcal{P}_s}$.

D'où finalement le choix de $\rho \in résolvantes(\Phi)$.

(Cas 2) $\Phi \in MNC(\mathcal{P})$

Seul le cas d'une menace entre un événement et une assertion est examiné ici, le cas d'une menace entre deux assertions étant très similaire.

Soit $\Phi =< e, h_p >$ une menace de $\mathcal{P}$ avec $e = event(attS(?x'_1, ..., ?x'_n) : (?v^-, ?v^+), t)$ et $h_P = hold(attS(?x_1, ..., ?x_n) : ?v_{protégée}, (t_{début}, t_{fin}))$.

Dans $\underline{\mathcal{P}_s}$, $< e, h_p >$ n'est pas une menace donc au moins une des contraintes intervenant dans la définition de la menace : $\{(?x_i \neq ?x'_i), (t < t_{début}), (t = t_{début} \text{ et } ?v^+ = ?v_{protégée}), (t = t_{fin} \text{ et } ?v^- = ?v_{protégée}), (t_{fin} < t)\}$ peut se déduire de $\underline{\mathcal{P}_s}$. D'après la définition des résolvantes associées à une menace, il existe donc une résolvante ρ permettant de poser cette contrainte et qui sera donc telle que $\mathcal{P} \oplus \rho \leftarrow \underline{\mathcal{P}_s}$.

(Cas 3) $\Phi \in ECM(\mathcal{P})$

Soit $\Phi = \{r_1, ..., r_k\}$ un ECM de $\mathcal{P}$ avec $r_i = use(attR(?x_{i1}, ..., ?x_{in}) : q_i, (t_i^-, t_i^+))$.

Supposons que dans $\underline{\mathcal{P}_s}$ toutes les variables $?x_{ik}$ soient instanciées à des valeurs X_{ik} et, de manière générale, les propositions r_i en $\underline{r_i}$. Dans $\underline{\mathcal{P}_s}$ il n'existe aucun ECM, donc en particulier, il n'existe aucun ECM contenant Φ.

Plusieurs cas sont à distinguer :

- si il existe deux propositions $\underline{r_i}$ et $\underline{r_j}$ auxquelles sont allouées des ressources différentes ($\exists l / X_{il} \neq X_{jl}$), alors la résolvante de séparation ρ constituée de la contrainte ($?x_{il} \neq ?x_{jl}$) convient et vérifie le lemme ;
- sinon, si une même ressource $attR(X_1, ..., X_n)$ est, dans $\underline{\mathcal{P}_s}$, allouée à toutes les propositions de Φ et si il existe deux propositions $\underline{r_i}$ et $\underline{r_j}$ ordonnancées dans $\underline{\mathcal{P}_s}$ ($t_i^+ < t_j^-$), alors la résolvante d'ordonnancement ρ constituée de la contrainte ($t_i^+ < t_j^-$) convient et vérifie le lemme ;
- sinon, si une même ressource $attR(X_1, ..., X_n)$ est, dans $\underline{\mathcal{P}_s}$, allouée à toutes les propositions de Φ et si toutes les propositions de Φ s'y intersectent temporellement, notons t^-_{max} le dernier instant parmi les débuts des propositions $\underline{r_i}$. A l'instant t^-_{max}, comme le plan $\underline{\mathcal{P}_s}$ est un plan consistant, la ressource $attR(X_1, ..., X_n)$ existe au moins dans une quantité $\sum_{i \in [1,k]} q(r_i)$. Puisqu'une telle quantité n'existait pas initialement dans le

plan partiel $\mathcal{P}$, c'est donc que la ressource $attR(X_1, ..., X_n)$ a été produite par un ensemble de tâches en une quantité suffisante pour assurer sa disponibilité avant l'instant t^-_{max}. Comme l'ensemble des résolvantes d'un ECM recouvre l'ensemble des combinaisons permettant de produire la ressource conflictuelle en quantité suffisante et avant chaque début de propositions, il existe une résolvante ρ de Φ compatible avec $\underline{\mathcal{P}_s}$.

Si on utilise la définition de la cohérence nécessaire au sens large, la démonstration est la même en prenant, pour $\underline{\mathcal{P}_s}$ une instance de $\mathcal{P}_s$ à laquelle on a rajouté les liens causaux adéquats pour protéger à tout instant les valeurs des attributs instanciés ; il est en effet aisé de voir dans ce cas-là que $\underline{\mathcal{P}_s}$ est aussi un plan solution.

□□□

Propriété 3 (Décomposabilité de la relation d'intersection possible)
soit U un sous-ensemble d'intervalles :
$\diamond \cap (U) \Leftrightarrow \forall V \subset U, \diamond \cap (V) \Leftrightarrow \forall \{I, J\} \subset U, \diamond \cap (\{I, J\})$

Preuve :

Le sens ($\Rightarrow$) est évident. Montrons ($\Leftarrow$) c'est-à-dire: $\forall \{I, J\} \subset U, \diamond \cap (\{I, J\}) \Rightarrow \diamond \cap (U)$.

Soient $T^- = \{I^-\}_{I \in U}$ l'ensemble des débuts des intervalles de U, $T^+ = \{I^+\}_{I \in U}$ l'ensemble des fins des intervalles de U, et T l'ensemble de tous les autres instants intervenant dans le réseau de contraintes.

Par hypothèse, $\forall I, J \in U$, les contraintes $I^+ < J^-$ et $J^+ < I^-$ ne peuvent pas être déduites des contraintes temporelles. Cela signifie que, sur le graphe représentatif de la relation d'ordre $<$ des contraintes entre instants, il n'existe aucun chemin entre les ensembles T^+ et T^- et d'autre part, $T^- \cap T^+ = \emptyset$ (cf. figure D.1).

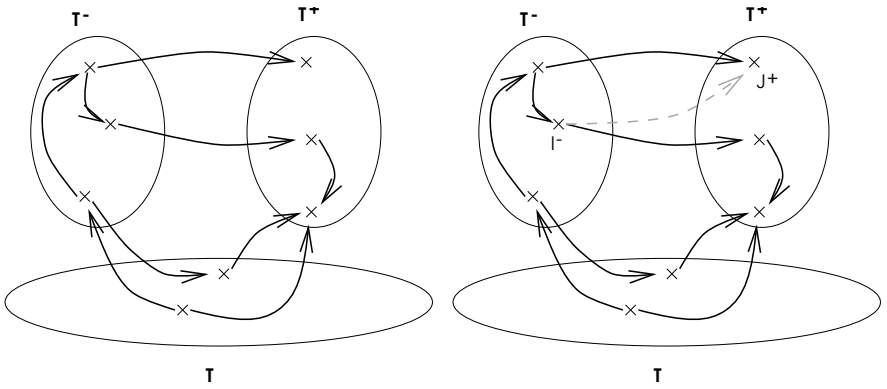


Figure D.1: Graphe des contraintes temporelles symboliques entre instants

Il suffit alors de montrer que l'on peut forcer une intersection nécessaire entre les intervalles de U en insérant les contraintes de précédences $\{I^- < J^+\}_{I,J \in U}$; c'est-à-dire de montrer que cet ensemble de contraintes est consistant avec les contraintes déjà contenues dans le réseau. Supposons que ce ne soit pas le cas et que l'insertion des contraintes $\{I^- < J^+\}_{I,J \in U}$

fasse apparaître un circuit C sur le graphe. Comme le graphe initial était consistant et donc sans circuit, c'est donc que C contient au moins un des arcs $I^- < J^+$ rajoutés. Soit alors $P = C \backslash (I^- < J^+)$; P est un chemin dans le nouveau graphe joignant J^+ à I^-. Comme il n'existe toujours pas d'arc entre T^+ et T^-, il est facile de construire un sous-chemin de P joignant un instant J_0^+ de T^+ à un instant I_0^- de T^- et n'empruntant, à part ces extrémités, que des instants de T. On en déduit alors par transitivité que $J_0^+ < I_0^-$ ce qui contredit l'hypothèse.

□□□

Théorème 5 (Consistance de la recherche) *Soit T un arbre de recherche d'ECM vérifiant les propriétés [C1,C2,C3]. Pour chaque nœud N de T , les éléments de $\Phi(N)$ sont des ensembles critiques minimaux.*

Preuve :

Il suffit de montrer que les éléments de $\Phi(N)$ sont des cliques du GIP puisqu'ils vérifient déjà par construction (cf. [C3]) la propriété de sur-consommation minimale.

On va en fait montrer que les propriétés [C1,C2,C3] suffisent à conférer à $\Pi(N)$ une structure de clique ; ce qui montrera le théorème puisque tout sous-ensemble de sommets d'une clique forme une clique.

Montrons auparavant les deux propriétés suivantes qui découlent de [C1,C2] :
P2: $M \in Ascendants(M') \Rightarrow P(M') \subset P(M)$
P3: $M \in Ascendants(M') \Rightarrow \Delta(M') \subset P(M)$

Preuve de [P2]:

Il est clair que $P(M') \subset \Sigma(M')$ par construction ([C2]). Or, $\Sigma(M')$ est constitué des ensembles $\{\Delta(K)\}_{K \in Cadets(M')}$; chacun de ces ensembles $\Delta(K)$ est, d'après [C1], un sous-ensemble de $P(Parent(M'))$, ce qui montre que $P(M') \subset P(Parent(M'))$. Une récurrence immédiate en remontant le long de la branche allant de M à M' montre alors le résultat $P(M') \subset P(M)$.

Preuve de [P3] :

On a $\Delta(M') \subset P(Parent(M'))$ par [C1], donc, comme d'après [P2] $P(Parent(M')) \subset P(M)$, on en déduit $\Delta(M') \subset P(M)$.

Suite de la preuve du théorème :

Soient $u, u' \in \Pi(N)$. Deux cas se présentent:

1. soit $\exists M \mid \{u, u'\} \in \Delta(M)$ et alors, comme $\Delta(M)$ est une clique, $\{u, u'\}$ est une arête du GIP ;
2. sinon, on peut supposer $\exists M, M' \mid u \in \Delta(M), u' \in \Delta(M')$ et $M \in Ascendants(M')$ et dans ce cas, $u' \in P(M)$ (d'après le résultat [P3] montré ci-dessus). Donc, par construction de $P(M)$, il existe une arête entre u et u'.

On voit donc que dans tous les cas, il existe une arête entre u et u'.

□□□

Théorème 6 (Systématicité de la recherche) *Soit T un arbre de recherche d'ECM vérifiant les propriétés [C1, C2, C3]. Les ensembles de conflits de ressources minimaux $\Phi(N)$ associés aux nœuds de T sont tous disjoints.*

Preuve :

Soit N et M deux nœuds distincts de T. Soit U et V deux ECM respectivement associés à N et M, c'est-à-dire $U \in \Phi(N)$ et $V \in \Phi(M)$. Il faut montrer que $U \neq V$.

Plusieurs cas peuvent se présenter :

1. $M \in Ascendants(N)$. Dans ce cas, si $V \in \Phi(M)$, on a aussi $V \subset \Pi(M)$ par [C3]. U étant un ECM associé à N, on a $U \cap \Delta(N) \neq \emptyset$ par [C3]. Pour établir $U \neq V$, il suffit de montrer que $\Delta(N) \cap \Pi(M) = \emptyset$. Nous allons en fait montrer $P(M) \cap \Pi(M) = \emptyset$ ce qui emportera la preuve puisque $\Delta(N) \subset P(M)$ (cf. [P3]). On a $\forall M' \in Ascendants(M), P(M) \subset P(M')$ [P3] et $P(M') \cap \Delta(M') = \emptyset$ [P1]. Comme $\Pi(M) = \cup_{M' \in Ascendants(M)} \Delta(M')$, il est clair que $P(M) \cap \Pi(M) = \emptyset$.

2. $N \in Ascendants(M)$. C'est le symétrique du cas ci-dessus.

3. Sinon, considérons O le plus récent antécédent commun aux nœuds M et N. Soient $M' = Enfants(O) \cap Ascendants(M)$ et $N' = Enfants(O) \cap Ascendants(N)$ (cf. figure D.2). Le fait d'avoir déjà examiné les cas (1) et (2) assure que $M' \neq N'$. On peut supposer $N' \in Cadets(M')$ quitte à inverser N et M. On a $V \cap \Delta(M') \neq \emptyset$ [C3]. Nous allons montrer que $U \cap \Delta(M') = \emptyset$ ce qui emportera la preuve que $U \neq V$.

   On a $\Pi(N) = \Pi(O) \cup \Delta(N') \cup ... \cup \Delta(N)$ et donc, d'après [P3] : $\Pi(N) \subset \Pi(O) \cup \Delta(N') \cup P(N')$. Il est clair que $\Delta(N') \cap \Delta(M') = \emptyset$ (cf. [C1]). D'autre part, d'après [P1], on peut facilement voir que $\Pi(O) \cap \Delta(M') = \emptyset$ en descendant le long de la branche allant jusqu'à M'. Enfin, $P(N') \cap \Delta(M') = \emptyset$ car $P(N') \subset \cup_{K \in Cadets(N')} \Delta(K)$ et tous ces ensembles $\Delta(K)$ sont disjoints de $\Delta(M')$ ([C1]) vu que $M' \notin Cadets(N')$. Finalement, on a montré $\Pi(N) \cap \Delta(M') = \emptyset$. Comme $U \in \Pi(N)$, cela montre bien que $U \cap \Delta(M') = \emptyset$

□□□

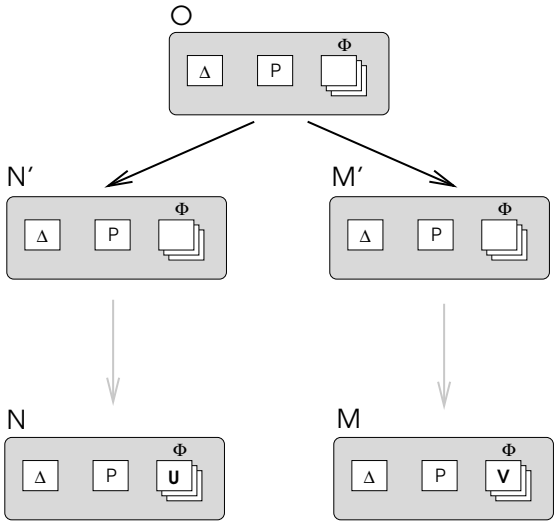


Figure D.2: Systématicité de la recherche des ECM

Théorème 7 (Complétude de la recherche) *Soit T un arbre de recherche d'ECM vérifiant la propriété [C1 à C4]. Supposons que T est entièrement développé (c'est-à-dire que chaque feuille N vérifie $P(N) = \emptyset$) ; alors si U est un ECM, il existe un nœud N de T tel que $U \in \Phi(N)$.*

Preuve :

Soit U un ECM. On recherche un nœud N tel que $U \in \Phi(N)$.

Posons $U_0 = U$.

Soit N_0 la racine de l'arbre. D'après [C4], comme U_0 est une clique de $P(N_0) = USE_{attR}$, il existe $N_1 \in Enfants(N_0)$ tel que $U_0 \in (\Delta(N_1) \cup P(N_1))$ et $U_0 \cap \Delta(N_1) \neq \emptyset$. Posons $U_1 = U_0 \backslash \Delta(N_1)$;

Si $U_1 = \emptyset$ posons $N = N_1$; sinon, U_1 est une clique de $P(N_1)$ et donc, en ré-appliquant [C4], il existe $N_2 \in Enfants(N_1)$ tel que $U_1 \in (\Delta(N_2) \cup P(N_2))$ et $U_1 \cap \Delta(N_2) \neq \emptyset$. Posons $U_2 = U_1 \backslash \Delta(N_2)$;

On peut poursuivre ainsi la récurrence jusqu'à trouver U_n tel que $U_n = \emptyset$: la convergence est assurée puisque, comme $U_i \cap \Delta(N_{i+1}) \neq \emptyset$, la suite U_i est strictement décroissante pour l'inclusion ensembliste.

Finalement, on a donc bien trouvé un nœud N qui vérifie par construction :

- $U \in \cup_{M \in Ascendants(N)} \Delta(M)$;
- $\forall M \in Ascendants(N), U \cap \Delta(M) \neq \emptyset$.

Comme d'autre part U est un ECM donc sur-consommateur minimal, on a bien $U \in \Phi(N)$.

□□□

Propriété 8 *L'agorithme DECLIC conserve les propriétés [C1 à C4] de l'arbre de recherche des ECM. En conséquence, il conduit à une recherche des ECM qui est* **consistante**, **systématique** *et* **complète**.

Preuve :

Le résultat est immédiat pour la propriété [C3].

Preuve de [C1] :

La propriété [C1] vient du fait que dans la boucle de la ligne 2.4, les éléments u_i de $P(N)$ sont mis un à un dans l'ensemble Δ (ligne 2.4.3), lequel ensemble Δ est quelquefois vidé dans les ensembles $\Delta(M)$ des nœuds fils créés (ligne 2.4.4.2). D'où le fait que les $\Delta(M)$ forment une partition de $P(N)$.

Reste à montrer que chaque ensemble $\Delta(M)$ est une clique du GIP. Supposons que $card(\Delta(M)) > 1$ (un sommet isolé est toujours une clique). Par construction $\Delta(M)$ est de la forme $\{u_i, u_{i+1}, ..., u_j\}$ avec $C(u_i) \subset C(u_{i+1}) \subset ... \subset C(u_j)$ où i désigne l'itération où l'ensemble Δ a été vidé pour créer un nouveau nœud (le dernier frère de M créé) pour la dernière fois. Soit $i \leq l < m \leq j$; comme $u_l \in C(u_l) \subset C(u_m)$, par construction de $C(u_m)$, $\{u_l, u_m\}$ est une arête du GIP. Ceci est vrai pour toute paire $\{l, m\}$ donc $\Delta(M)$ est bien une clique du GIP.

Preuve de [C4] :

Soit N un nœud de l'arbre et U une clique du GIP telle que $U \subset P(N)$. Il faut montrer l'existence d'un enfant M de N tel que $U \subset (\Delta(M) \cup P(M))$ et $U \cap \Delta(M) \neq \emptyset$.

$P(N)$ étant indexé selon σ, soit $u \in U$ la proposition qui maximise la fonction d'indexation σ sur U. Supposons $u = u_l$.

Il est clair que $U \subset C(u_l)$.

Soit m le plus grand indice tel que $C(u_l) \subset C(u_{l+1}) \subset ... \subset C(u_m)$. A l'indice m correspond la création d'un enfant M du nœud N tel que $\{u_l, u_{l+1}, ..., u_m\} \subset \Delta(M)$ et $\Delta(M) \cup P(M) = C(u_m)$. Le nœud M convient donc.

Preuve de [C2] :

Si la création d'un nœud M correspond à l'indice m et si l désigne le plus petit indice tel que $C(u_l) \subset C(u_{l+1}) \subset ... \subset C(u_m)$ on a, par construction $P(M) = C(u_l) \backslash \{u_l\}$ et $\Delta(M) = \{u_l, u_{l+1}, ..., u_m\}$.

Soit $u \in P(M)$ et $u_j \in \Delta(M)$. Alors comme $P(M) \subset C(u_l) \subset C(u_j)$, $\{u, u_j\}$ est une arête du GIP.

Réciproquement, soit $u \in \Sigma(M) \mid \forall j \in [1, m], \{u, u_j\}$ est une arête du GIP. Avec les notations de l'algorithme, $\Sigma(M) = \{u_1, ..., u_{l-1}\}$. Supposons donc $u = u_i$.

Comme $\{u_i, u_j\}$ est une arête du GIP, on a $u_i \in C(h_l)$ et comme $i < l$, $u_i \in C(u_l) \backslash \{u_l\} = P(M)$. On a donc bien montré que $u \in P(M)$.

Propriété 9 *La relation $\leftarrow$ (ainsi que $\stackrel{<}{\leftarrow}$) définit un préordre sur l'ensemble des contraintes temporelles disjonctives.*

Preuve :

(1) Reflexivité :

Il est clair que $\forall \beta' \in \gamma, \exists \beta \in \gamma \mid \beta \leftarrow \beta'$ (il suffit de prendre $\beta = \beta'$). On a donc $\forall \gamma, \gamma \leftarrow \gamma$.

(2) Transitivité :

Soient γ, γ' et γ'' telles que $\gamma \leftarrow \gamma'$ et $\gamma' \leftarrow \gamma''$. Soit $\beta'' \in \gamma''$, comme $\gamma' \leftarrow \gamma''$, il existe $\beta' \in \gamma' \mid \beta' \leftarrow \beta''$. Soit un tel β', alors, comme $\gamma \leftarrow \gamma'$, il existe $\beta \in \gamma \mid \beta \leftarrow \beta'$. La transitivité de $\leftarrow$ sur les contraintes temporelles conjonctives étant évidente, on a donc trouvé $\beta \in \gamma \mid \beta \leftarrow \beta''$, d'où $\gamma \leftarrow \gamma''$.

□□□

Théorème 8 (Unicité des contraintes minimales) *Soit γ une contrainte temporelle disjonctive. Il existe une* **unique** *contrainte disjonctive γ_{min} telle que (1) γ et γ_{min} sont* **équivalentes** *sur $<$ et (2) γ_{min} est* **minimale** *sur $<$.*

Preuve :

Nous allons d'abord montrer le théorème (existence et unicité d'une contrainte équivalente minimale) pour une contrainte conjonctive β (étapes 1.1 et 1.2) ; ensuite, dans le cas de contraintes disjonctives, nous commencerons par montrer l'existence d'une contrainte γ_m vérifiant les conditions du théorème (étape 2.1) avant de prouver son unicité (étape 2.2).

Etape 1.1

Soit G le graphe associé à la contrainte temporelle conjonctive $\beta \wedge \beta_<$ et $\tau(G)$ sa fermeture transitive. $\tau(G)$ est sans circuit puisque par hypothèse, β est cohérente sur $<$.

$\tau(G)$ admet donc un graphe G_{min} qui lui soit τ-équivalent et qui soit τ-minimal (cf. [Roy 69], Tome 1, page 259). Ce graphe est unique (c'est la relation de couverture associée à l'ordre induit par G) et de plus G_{min} est un graphe partiel de G.

Soit β_m définie par les arcs (t, t') de G_{min} ne vérifiant pas $t < t'$.

On a $\beta \stackrel{<}{\longleftrightarrow} \beta_m$ car par construction, $\tau(\beta \wedge \beta_<) = \tau(\beta_m \wedge \beta_<)$. De plus β_m est minimale sur $<$ car si on a $\beta' \subset \beta$ avec $\beta' \neq \beta$ alors, par définition de la τ-minimalité, on aura $\tau(\beta' \wedge \beta_<) \subset \tau(\beta \wedge \beta_<)$ avec $\tau(\beta' \wedge \beta_<) \neq \tau(\beta \wedge \beta_<)$ et donc on ne pourra pas avoir $\beta' \stackrel{<}{\longleftrightarrow} \beta$.

Etape 1.2

Unicité de β_m : soit $\beta' \neq \beta$ telle que $\beta' \overset{\leq}{\longleftrightarrow} \beta_m$.

Par définition : $\tau(\beta_m \wedge \beta_<) = \tau(\beta \wedge \beta_<) = \tau(\beta' \wedge \beta_<)$. Soit G' le graphe associé à $\beta' \wedge \beta_<$. G est τ-équivalent à G' et est τ-minimal. Par théorème, G est un graphe partiel strict de G' et donc $\beta \subset \beta'$ avec $\beta \neq \beta'$. Donc β' n'est pas minimale.

Etant donné une contrainte conjonctive β, nous noterons $\mu_<(\beta)$ l'unique contrainte conjonctive β_m minimale et équivalente à β sur $<$.

La construction de $\beta_m = \mu_<(\beta)$ peut être effectuée à l'aide de l'algorithme suivant qui s'inspire de celui proposé dans [Roy 69] (p 291) pour la détection de contraintes de précédence τ-supprimables :

Algorithme $\mu(<, \beta)$
Nous supposons $\beta = [\alpha_1, \alpha_2, ..., \alpha_p]$
1. $T_\beta \leftarrow \{t \mid \exists \alpha_i \in \beta \mid t \in \alpha_i\}$
2. $M_\beta \leftarrow$ matrice associée à $\beta \wedge \beta_<$ sur les instants de T_β i.e.
$M_\beta(t_1, t_2) = 1$ ssi $(t_1, t_2) \in \beta$ ou $t_1 < t_2$
$M_\beta(t_1, t_2) = 0$ sinon
3. Soit $A_\beta \leftarrow M_\beta + \tau(M_\beta)$, τ désignant la fermeture transitive matricielle
4. Pour $i = 1$ jusqu'à p faire : (soit $\alpha_i = (t_1, t_2)$)
4.1. Pour $t'_1 \in T_\beta \backslash \{t_1, t_2\}$
4.1.1. Pour $t'_2 \in T_\beta \backslash \{t_1, t_2\}$
4.1.1.1. Si $t'_1 = t'_2$ et $A_\beta(t'_2, t_2) = 2$ et $A_\beta(t_1, t'1) = 2$
Alors ôter α_i de β
4.1.1.2. Si $t'_1 \neq t'_2$ et $A_\beta(t'_2, t_2) = 2$ et $A_\beta(t_1, t'1) = 2$ et $A_\beta(t'_1, t'_2) = 2$
Alors ôter α_i de β
5. Retourner β

La preuve de ce théorème découle directement de la construction de β_m donnée dans la preuve ci-dessus.

Etape 2.1

Etant donné $\gamma = \{\beta_1, \beta_2, ..., \beta_q\}$, nous allons dans un premier temps (1) *construire* une contrainte disjonctive γ_m équivalente à γ. Puis nous prouverons (2) que, par construction, γ_m est minimale.

(1) construisons à partir de γ une contrainte disjonctive γ_1 en ne conservant de γ que les contraintes conjonctives β_i minimales au sens du préordre $\overset{\leq}{\leftarrow}$ c'est-à-dire :
$\gamma_1 = \gamma \backslash \{\beta \mid \exists \beta' \in \gamma, \beta' \neq \beta \mid (\beta' \overset{\leq}{\leftarrow} \beta)\}$. Montrons tout d'abord $\gamma \overset{\leq}{\longleftrightarrow} \gamma_1$.

Il est clair d'une part que $\gamma \overset{\leq}{\leftarrow} \gamma_1$ vu que par construction $\gamma_1 \in \gamma$.

D'autre part, soit $\beta \in \gamma$; deux cas peuvent se présenter :

- si $\beta \in \gamma_1$ alors, comme $\beta \overset{\leq}{\leftarrow} \beta$, $\exists \beta \in \gamma_1 \mid \beta \overset{\leq}{\leftarrow} \beta$;
- si $\beta \notin \gamma_1$ alors β n'est pas minimal au sens du préordre $\overset{\leq}{\leftarrow}$ dans γ, donc, c'est qu'il existe $\beta' \in \gamma_1 \mid \beta' \overset{\leq}{\leftarrow} \beta$ (dans un préordre, tout élément non-minimal est supérieur à un élément minimal).

On a donc montré $\forall \beta \in \gamma, \exists \beta' \in \gamma_1 \mid \beta' \overset{<}{\leftarrow} \beta$ donc $\gamma_1 \overset{<}{\leftarrow} \gamma$.

Soit maintenant $\gamma_m = \{\mu_<(\beta)\}_{\beta \in \gamma_1}$. Il est évident que $\gamma' \overset{<}{\longleftrightarrow} \gamma_1$ d'où $\gamma' \overset{<}{\longleftrightarrow} \gamma$.

(2) Minimalité de γ_m.

Il est clair que par construction, γ_m vérifie le point (1) de la définition de la minimalité.

Il reste à montrer que $\forall \gamma' \neq \gamma_m \mid (\gamma_m \overset{<}{\longleftrightarrow} \gamma'), \gamma' \not\subset \gamma_m$. Nous allons montrer, ce qui est équivalent, que $\forall \gamma' \neq \gamma_m \mid \gamma' \subset \gamma_m$, on n'a pas $(\gamma_m \overset{<}{\longleftrightarrow} \gamma')$.

Soit γ' telle que $\gamma' \neq \gamma_m$ et $\gamma' \subset \gamma_m$. Soit $\beta \in \gamma_m \backslash \gamma'$, alors, $\forall \beta' \in \gamma'$, on n'a pas $\beta' \overset{<}{\leftarrow} \beta$ car $\beta \in \gamma_m$ et β est minimale pour $\overset{<}{\leftarrow}$ sur γ_m.

On vient donc de montrer:
$\forall \gamma' \neq \gamma_m \mid \gamma' \subset \gamma_m, [\exists \beta \in \gamma_m \mid \forall \beta' \subset \gamma',$ *on n'a pas* $\beta' \overset{<}{\leftarrow} \beta]$ soit
$\forall \gamma' \neq \gamma_m \mid \gamma' \subset \gamma_m, [$ *on n'a pas* $\gamma_m \overset{<}{\leftarrow} \gamma']$

D'où le résultat de minimalité de γ_m.

Etape 2.2

Soit $\gamma_m = \{\beta_{m1}, \beta_{m2}, ..., \beta_{mr}\} \mid (\gamma_m \overset{<}{\leftarrow} \gamma)$ et γ_m minimale sur $<$.

Alors, γ_m vérifie : $\forall k, l \in [1, r] \mid k \neq l$, on n'a pas $\beta_{mk} \overset{<}{\leftarrow} \beta_{ml}$ puisque sinon, on pourrait retrancher de γ_m les éléments non-minimaux pour le préordre $\overset{<}{\leftarrow}$ et on obtiendrait une contrainte disjonctive équivalente et strictement contenue dans γ_m.

Pour terminer la preuve du théorème, il suffit alors de montrer le lemme ci-dessous. En effet, on vient de voir que tout couple de contraintes minimales vérifie ce lemme.

Lemme 2 *Soient* $\gamma = \{\beta_1, \beta_2, ..., \beta_q\}$ *et* $\gamma' = \{\beta'_1, \beta'_2, ..., \beta'_r\}$ *deux contraintes disjonctives ne contenant que des contraintes conjonctives* β *minimales.*

Si γ *et* γ' *vérifient :*

- $\gamma \overset{<}{\longleftrightarrow} \gamma'$
- $\forall i, j \in [1, q] \mid i \neq j$, *on n'a pas* $\beta_i \overset{<}{\leftarrow} \beta_j$
- $\forall k, l \in [1, r] \mid k \neq l$, *on n'a pas* $\beta'_k \overset{<}{\leftarrow} \beta'_l$

Alors $\gamma = \gamma'$.

Preuve du lemme 2

$\gamma' \overset{<}{\leftarrow} \gamma$ s'écrit $\forall \beta_i \in \gamma, \exists \beta'_k \in \gamma' \mid (\beta'_k \overset{<}{\leftarrow} \beta_i)$. [i]

$\gamma \overset{<}{\leftarrow} \gamma'$ s'écrit $\forall \beta'_k \in \gamma', \exists \beta_j \in \gamma \mid (\beta_j \overset{<}{\leftarrow} \beta'_k)$. [ii]

On va montrer $\gamma \subset \gamma'$; on montrerait de manière strictement symétrique $\gamma' \subset \gamma$.

Soit $\beta_i \in \gamma$; soit $\beta'_k \in \gamma' \mid (\beta'_k \overset{<}{\leftarrow} \beta_i)$ (d'après [i]) ; soit $\beta_j \in \gamma \mid (\beta_j \overset{<}{\leftarrow} \beta'_k)$ (d'après [ii]).

Nécessairement, $\beta_i = \beta_j$ car sinon, on aurait par transitivité $\beta_j \overset{<}{\leftarrow} \beta_i$ ce qui contredirait [1].

On a donc $\forall \beta_i \in \gamma, \exists \beta'_k \in \gamma' \mid (\beta'_k \overset{<}{\longleftrightarrow} \beta_i)$.

Or, par hypothèse, β_i et β'_k sont minimales ; d'après le théorème d'unicité montré précédemment pour les contraintes conjonctives (étape 1), on a donc $\beta_i = \beta'_k$.

Ceci montre bien $\gamma \subset \gamma'$.

Algorithme $\mu(<, \gamma)$

Nous supposons $\gamma = [\beta_1, \beta_2, ..., \beta_q]$

1. Pour $i = 1$ jusqu'à q faire :
  1.1. $\beta_i \leftarrow \mu(<, \beta_i)$ (cf. algorithme précédent)

2. Pour $i = 1$ jusqu'à q faire :
  2.1. Pour $j = 1$ jusqu'à q faire :
    2.1.1. $T_{ij} \leftarrow T_{\beta_i} \cup T_{\beta_j}$
    2.1.2. $M_i \leftarrow$ matrice associée à $\beta_i \wedge \beta_<$ sur T_{ij}
    2.1.3. $M_j \leftarrow$ matrice associée à $\beta_j \wedge \beta_<$ sur T_{ij}
    2.1.4. Si $(i \neq j)$ et $M_i \subset \tau(M_j)$
      Alors, ôter β_j de γ (car $\beta_i \stackrel{<}{\leftarrow} \beta_j$)

3. Retourner γ

Ici encore, la preuve de l'algorithme est directement donnée par la méthode de construction de γ_m dans la preuve ci-dessus.

Théorème 9 *: Les hiérarchies d'abstraction utilisées dans I_XT_ET vérifient la propriété de monotonicité ordonnée : pour chaque plan partiel courant possible, aucun raffinement envisageable de ce plan partiel suite à la résolution de défauts ne fera apparaître de nouveau défaut qui soit d'un niveau strictement supérieur au niveau d'abstraction courant.*

Avant de montrer ce théorème, nous allons montrer les deux lemmes suivants :

Lemme 3 *Soient $C_0, \ldots, C_k$ une séquence d'ensembles C consécutifs. $\forall att_k \in C_k$, il existe une séquence $att_0, \ldots, att_k$, telle que $\forall i$, $att_i \in C_i$ et, ou bien $att_{i+1} = att_i$ ou bien $\widehat{att}_{i+1} < \widehat{att}_i$.*

Preuve :

Démonstration immédiate par récursion sur la règle de mise à jour en partant de att_k et en utilisant le fait que $\widehat{att}_{nouveau} < \widehat{att}_{résolu}$.

□□□

Lemme 4 *Pour tout état d'abstraction (R, C), on a $C \cap R = \emptyset$. De plus si att et att' sont deux noms d'attributs dans C, alors ou bien $\widehat{att} = \widehat{att}'$, ou bien $\widehat{att}$ et $\widehat{att}'$ sont non-comparables c'est-à-dire que l'on ne peut avoir $\widehat{att} < \widehat{att}'$ ou $\widehat{att}' < \widehat{att}$.*

Preuve :
(Par récurrence le long d'une chemin de l'arbre de recherche.)

Pour la racine de l'arbre, il est facile de voir à partir de la définition de C_0 que la propriété est vérifiée pour (R_0, C_0).

Supposons maintenant que la propriété est vérifiée pour tout état d'abstraction sur le chemin allant de (R_0, C_0) à (R, C). Soit (R', C') l'état d'abstraction juste après (R, C) (cf. figure D.3).

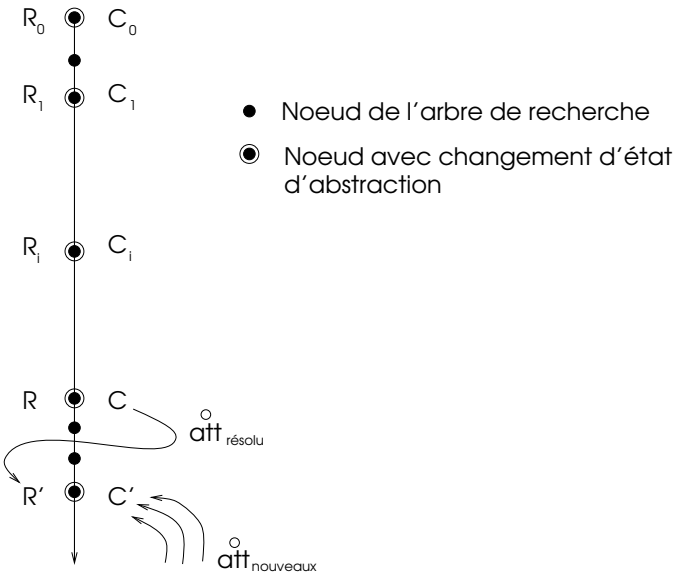


Figure D.3: Etats d'abstraction consécutifs sur un arbre de recherche

$$\begin{aligned} C' \cap R' &= (C - \widehat{att}_{résolu} \cup \{att_{nouveau}\}) \cap (R \cup \widehat{att}_{résolu}) \\ &= \{att_{nouveau}\} \cap (R \cup \widehat{att}_{résolu}) \\ &= (\{att_{nouveau}\} \cap R) \cup (\{att_{nouveau}\} \cap \widehat{att}_{résolu}) \end{aligned}$$

Nous avons $\{att_{nouveau}\} \cap \widehat{att}_{résolu} = \emptyset$ puisque $\widehat{att}_{nouveau} < \widehat{att}_{résolu}$.

Supposons que $\exists att_{nouveau} \in R$. Le long du chemin allant jusqu'au nœud courant, soit (R^-, C^-) le dernier état d'abstraction où $att_{nouveau} \notin R^-$ et $att_{nouveau} \in C^-$. Par le lemme 3 appliqué à l'état (R', C'), $\exists att \in C^- \mid \widehat{att}_{nouveau} < \widehat{att}$. Comme att et $att_{nouveau}$ sont dans C^-, il y a contradiction avec l'hypothèse de récurrence et donc $C' \cap R' = \emptyset$.

Soient maintenant att_1, att_2 dans C' avec $\widehat{att}_1 \neq \widehat{att}_2$.

Si $att_1 \in C$ et $att_2 \in C$ alors, d'après l'hypothèse de récurrence, att_1 et att_2 vérifient la deuxième partie du lemme donc att_1 et att_2 ne sont pas comparables par $<$.

Supposons maintenant que $att_1 \in C' \backslash C$. Alors nécessairement, att_1 fait partie des nouveaux noms d'attributs $\{\widehat{att}_{nouveau}\}$; dans la suite, nous confondrons les notations att_1 et $att_{nouveau1}$.

Si on avait $\widehat{att}_{nouveau1} < \widehat{att}_2$ cela voudrait dire que l'on a un $att_{nouveau}$ qui vérifie $\widehat{att}_{nouveau} < \widehat{att}_2$ avec $\widehat{att}_2 \in C'$ ce qui est interdit par la définition de l'ensemble des $\{att_{nouveau}\}$ (cf. définition 17).

Supposons maintenant que l'on ait $\widehat{att}_2 < \widehat{att}_{nouveau1}$. Si $\widehat{att}_2 \in C$, Soit C_i le premier état d'abstraction tel que l'on ait $\widehat{att}_2 \in C_{i+1}$ et $\widehat{att}_2 \notin C_i$. D'après la définition 17 pour passer

de C_i à C_{i+1}, $\forall att \mid \widehat{att}_2 < \widehat{att}, att \in R_i$; or comme l'on a par hypothèse $\widehat{att}_2 < \widehat{att}_{nouveau1}$, on en déduit $\widehat{att}_{nouveau1} \in R_i$ or $R_i \subset R_{i+1} \subset ... \subset R \subset R'$ d'où $\widehat{att}_{nouveau1} \in R'$; comme d'autre part $\widehat{att}_{nouveau1} \in C'$ et que l'on a montré plus haut $R' \cap C' = \emptyset$, il y a contradiction. Reste le cas où $\widehat{att}_2 \notin C$ c'est-à-dire si att_2 est lui aussi un $att_{nouveau2}$. Il est facile de se convaincre, uniquement en considérant la formule de passsage de C à C' dans la définition 17 que deux attributs nouveaux $att_{nouveau1}$ et $att_{nouveau2}$ insérés en même temps sont nécessairement non-comparables puisque si par exemple $\widehat{att}_{nouveau1} < \widehat{att}_{nouveau2}$ alors il est clair que $\widehat{att}_{nouveau1} \notin C'$.

□□□

Preuve du Théorème 9 :

Soit (R, C) l'état d'abstraction courant, et L le niveau d'abstraction associé. Soit Φ un défaut de L ($\widehat{att}_\Phi \in C$). Supposons que la résolution du défaut Φ conduise à l'apparition d'un nouveau défaut Φ'. Nous devons montrer que nécessairement, $att_{\Phi'} \notin R$.

Supposons que $att_{\Phi'} \in R$. La seule manière d'obtenir un tel nouveau défaut sur $att_{\Phi'}$ est d'insérer une tâche $\mathcal{T}$ dans le but de résoudre Φ. D'après la définition 14, l'analyse de la tâche $\mathcal{T}$ apportera la contrainte $\widehat{att}_{\Phi'} < \widehat{att}_\Phi$ ou $\widehat{att}_{\Phi'} = \widehat{att}_\Phi$.

Le long du chemin arrivant jusqu'au nœud courant, soit (R^-, C^-) le dernier état d'abstraction où $att_{\Phi'} \notin R^-$ et $att_{\Phi'} \in C^-$. A partir de $\widehat{att}_{\Phi'} < \widehat{att}_\Phi$ ou $\widehat{att}_{\Phi'} = \widehat{att}_\Phi$ et du lemme 4, on en déduit $att_\Phi \notin C^-$. Ainsi, en appliquant le lemme 3, $\exists att \in C^- \ \widehat{att}_\Phi < \widehat{att}$, ce qui entraîne $\widehat{att}_{\Phi'} < \widehat{att}$, et contredit le lemme 4 puisque $\widehat{att}_{\Phi'}$ et $\widehat{att}$ appartiennent au même ensemble C^-.

□□□

Propriété 10 *En l'absence de prédicats de type assert ou produce,* $\forall \mathcal{P} \in COMPL(\mathcal{P}_0)$, $\forall \Phi, \Phi' \in \mathcal{P}$,

$$att_\Phi \prec att_{\Phi'} \Rightarrow att_\Phi \prec_{SI} att_{\Phi'}$$

Preuve :

Nous allons tout d'abord montrer que :

$$att_\Phi \dot{\prec} att_{\Phi'} \Rightarrow att_\Phi \dot{\prec}_{SI} att_{\Phi'}$$

Remarquons tout d'abord que nécessairement, comme on a rajouté la relation $att_\Phi \dot{\prec} att_{\Phi'}$ en appliquant la définition 14, $att_{\Phi'}$ est utilisé dans au moins un événement d'une tâche $\mathcal{T}$ du problème qui contient aussi une proposition temporelle sur l'attribut att_Φ. Il est clair que comme Φ' est un défaut d'un plan partiel issu de $\mathcal{P}_0$, $att_{\Phi'}$ est relevant pour $\mathcal{P}_0$. D'autre part, comme les événements expriment à la fois une condition et un effet sur l'attribut, il est aisé de voir que dès qu'un attribut associé à un événement est relevant dans une

tâche, il est aussi relevant dans toutes les tâches qui contiennent un événement sur ce même attribut. Ceci permet de voir que $att_{\Phi'} \in relevant(\mathcal{T}, \mathcal{P}_0)$. On a donc, d'après la définition 19, $att_\Phi \dot{\prec}_{SI} att_{\Phi'}$.

Montrons maintenant la propriété :

Si l'on a $att_\Phi \prec att_{\Phi'}$, c'est qu'il existe un chemin $att_\Phi \dot{\prec} att_1 \dot{\prec} ... \dot{\prec} att_n \dot{\prec} att_{\Phi'}$. Sur ce chemin, du fait de l'existence d'une relation $att \dot{\prec} att_i$, chacun des attributs att_i est un attribut d'état intervenant dans un événement d'une tâche. On peut alors facilement se convaincre dans ce cas que chacun de ces attributs att_i est relevant et que donc, on a aussi : $att_\Phi \dot{\prec}_{SI} att_1 \dot{\prec}_{SI} ... \dot{\prec}_{SI} att_n \dot{\prec}_{SI} att_{\Phi'}$, ce qui montre la propriété.

□□□

Références bibliographiques

[Alami 92] R. Alami, M. Borillo, F. Garcia, M. Ghallab, H. Laruelle, R. Mampey & T. Vidal. *Representation et algorithmes pour la planification et l'action.* 4emes Journees Nationales PRC-GDR Int. Art., Marseille (France), 1992, pp.495-545, Teknea, 1992.

[Allen 83a] J.F. Allen. *Maintaining knowledge about temporal intervals.* Comm. ACM, vol. 26, pages 832–843, 1983.

[Allen 83b] J.F. Allen & J. Koomen. *Planning using a Temporal World Model.* In Proceedings IJCAI-83, pages 741–747, 1983.

[Allen 84] J.F. Allen. *Towards a General Theory of Action and Time.* Artificial Intelligence, vol. 23, pages 123–154, 1984.

[Allen 91] J. Allen. Reasoning about Plans, chapter 1: Temporal Reasonning and Planning. J. Allen, H. Kautz, R. Pelavin and J. Tenenberg editors, Morgan Kaufmann San Mateo (CA), 1991.

[Arkin 87] E. Arkin & E. Silverberg. *Scheduling Jobs with Fixed Start and End Times.* Discrete Applied Mathematics, vol. 18, pages 1–8, 1987.

[Bacchus 91] F. Bacchus & Q. Yang. *The Downward Refinement Property.* In Proceedings IJCAI-91, pages 286–292, 1991.

[Bacchus 94] F. Bacchus & Q. Yang. *Downward refinement and the efficiency of hierarchical problem solving.* Artificial Intelligence, vol. 71, pages 43–100, 1994.

[Bäckström 95a] C. Bäckström. *Five Years of Tractable Planning.* In Proceedings 3rd European Workshop on Planning (EWSP-95), pages 27–41, 1995.

[Bäckström 95b] C. Bäckström & P. Jonsson. *Planning with Abstraction Hierarchies can be Exponentially Less Efficient.* In Proceedings IJCAI-95, pages 1599–1604, 1995.

[Bastié 96] C. Bastié & P. Régnier. *Planification et exécution en environnement dynamique : le suivi d'exécution dans SPEEDY.* In Actes RFIA-96, pages 879–887, 1996.

[Bensana 88] E. Bensana, G. Bel & D. Dubois. *OPAL: a multi-knowledge-based system for industrial job-shop scheduling.* International Journal of Production Research, vol. 26, no. 5, pages 795–819, 1988.

[Berge 70] C. Berge. Graphes et hypergraphes. Dunod, Paris, 1970.

[Berry 94] P. Berry, B. Chouery & L. Friha. *Distribuated Approach to Dynamic Resource Management Based on Temporal Influence.* Intelligent Systems Engineering, vol. 3, no. 2, pages 79–86, 1994.

[Bessière 92] C. Bessière. *Systèmes à Contraintes Evolutifs en Intelligence Artificielle.* PhD thesis, Université Montpellier II, Septembre 1992.

[Bylander 92] T. Bylander. *Complexity Results for Extended Planning.* In Proceedings of the First International Conference on AI Planning Systems, 1992.

[Carlier 82] J. Carlier & A. Rinnooy Kan. *Financing and Scheduling.* Operations Research Letters, vol. 1, no. 2, pages 52–55, 1982.

[Carlier 88] J. Carlier & P. Chretienne. Les problèmes d'ordonnancement: modélisation, complexité, algorithmes. Masson, Paris, 1988.

[Chapman 87] D. Chapman. *Planning for Conjunctive goals.* Artificial Intelligence, vol. 32, pages 333–377, 1987.

[Chater 95] M. Chater. *De la Planification Incrémentale vers la Planification Multi-Robots.* Rapport de Stage DEA Systèmes Intelligents, Université Paris-IX Dauphine, 1995.

[Chouery 95] B. Chouery. *Abstraction Methods for Resource Allocation.* PhD thesis, École Polytechnique Fédérale de Lausanne, 1995.

[Cooper 90] M. Cooper. *An Optimal k-Consistency Algorithm.* Artificial Intelligence, vol. 41, pages 89–95, 1990.

[Currie 91] K. Currie & A. Tate. *O-Plan: the open planning architecture.* Artificial Intelligence, vol. 52, pages 49–86, 1991.

[Dean 87] T. Dean & D. McDermott. *Temporal Database Management.* Artificial Intelligence, vol. 32, pages 1–55, 1987.

[Dechter 91] R. Dechter, I. Meiri & J. Pearl. *Temporal Constraint Networks.* Artificial Intelligence, vol. 49, pages 61–95, 1991.

[Drabble 94] B. Drabble & A. Tate. *The Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner.* In Proceedings AIPS-94, pages 243–248, 1994.

[El-Kholy 96] A. El-Kholy & B. Richards. *Resource Reasonning and Scheduling in* parc*PLAN.* In submitted to AIPS-96, 1996.

[Erol 92] K. Erol, D. Nau & V. Subrahmanian. *On the complexity of domain-independent planning.* In Proceedings AAAI-92, pages 381–386, 1992.

[Erschler 90] J. Erschler, P. Lopez & C. Thuriot. *Temporal reasoning under resource constraints: Application to task scheduling.* In Advances in Support Systems Research, pages 189–194. George E. Lasker and Robbin R. Hough (eds.), 1990.

[Falkenauer 91] E. Falkenauer & S. Bouffouix. *A Genetic Algorithm for Job Shop.* In Proceedings IEEE International Conference on Robotics and Automation, pages 824–829, 1991.

[Farreny 87] H. Farreny & M. Ghallab. Eléments d'intelligence artificielle. Editions Hermès, Paris, 1987.

[Fikes 71] R.E. Fikes & N.J. Nilsson. *STRIPS : a new approach to the application of theorem proving to problem solving.* Artificial Intelligence, vol. 2, pages 189–208, 1971.

[Fink 95a] E. Fink & M. Veloso. *Formalizing the PRODIGY Planning Algorithm.* In Proceedings 3rd European Workshop on Planning (EWSP-95), pages 279–289, 1995.

[Fink 95b] E. Fink & Q. Yang. *Planning with Primary Effects: Experiments and Analysis.* In Proceedings IJCAI-95, pages 1606–1611, 1995.

[Fox 84] M.S. Fox & S.F. Smith. *ISIS: a knowledge-based system for factory scheduling.* Expert Systems, vol. 1, no. 1, pages 25–49, 1984.

[Gallone 95] J.M. Gallone, F. Charpillet & F. Alexandre. *Une approche 'anytime' pour l'ordonnancement de tâches non-préemptives.* In Procceedings INRIA/IEEE Symposium on Emerging Technologies and factory Automation (ETFA-95), pages 509–520, 1995.

[Garcia 95] F. Garcia & P. Laborie. *Hierarchisation of the search space in temporal planning.* pages 235–249, 1995. Proceedings EWSP-95.

[Gavril 72] F. Gavril. *Algorithms for maximum coloring, maximum clique, minimum covering by cliques and maximum independant set of a chordal graph.* SIAM J. Comput., vol. 1, pages 180–187, 1972.

[Ghallab 83] M. Ghallab & D.G. Allard. *A_ε: an efficient near admissible heuristic search algorithm.* In Proceedings IJCAI-83, 1983.

[Ghallab 89a] M. Ghallab. *Représentation et gestion de relations temporelles.* In Actes 7ème Congrès RFIA, Paris, November 1989.

[Ghallab 89b] M. Ghallab & A. Mounir-Alaoui. *Managing Efficiently Temporal Relations Through Indexed Spanning Trees.* In Proceedings IJCAI-89, 1989.

[Ghallab 95a] M. Ghallab & T. Vidal. *Focusing on a Sub-graph for Managing Efficiently Numerical Temporal Constraints.* In Proceedings FLAIRS-95 (to appear), 1995.

[Ghallab 95b] M. Ghallab & T. Vidal. *Temporal Constraint in Planning: Free or not Free ?* In FLAIRS-95 Workshop on Constraint Reasonning (to appear), 1995.

[Glover 93] F. Glover. Tabu search. Blackwell Scientific Publications, 1993. Modern Heuristic Techniques in Combinatorial Problems, pp70-150.

[Golumbic 80] M. Golumbic. Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York, 1980.

[Golumbic 92] M. Golumbic & R. Shamir. *Complexity and Algorithms for Reasonning about Time.* In Proc. AAAI-92, San Jose, pages 741–747, 1992.

[Golumbic 93] M. Golumbic & R. Shamir. *Complexity and Algorithms for Reasonning about Time: A Graph-Theoric Approach.* Journal of the Association for Computing Machinery, vol. 40, no. 5, pages 1108–1133, November 1993.

[GOThA 93] GOThA. *Les problèmes d'ordonnancement.* Recherche Opérationnelle, vol. 27, no. 1, pages 77–150, 1993. GOThA: Groupe d'Ordonnancement Théorique et Appliqué.

[Granier 88] T. Granier. *Contribution l'étude du temps objectif dans le raisonnement.* Rapport technique RR 716-I-73, LIFIA, Grenoble, Février 1988.

[Grant 86] T.J. Grant. *Lessons for O.R. from A.I.: A Scheduling Case Study.* Journal of the Operational Research Society, vol. 37, no. 1, pages 41–57, 1986.

[Green 69] C. Green. *Application of theorem proving to problem solving.* In Proceedings IJCAI, 1969.

[Grötschel 81] M. Grötschel, L. Lovasz & A. Schrijver. *The Ellipsoid Method and its Consequences on Combinatorial Optimization.* Combinatorica, vol. 1, pages 169–197, 1981.

[Hayward 85] R. Hayward. *Weakly Triangulated Graphs.* Journal of Combinatorial Theory, vol. 39, pages 200–209, 1985.

[Hertzberg 93] J. Hertzberg & E. Rutten. *Temporal Planner = Nonlinear Planner + Time Map Manager.* AI-Communications, vol. 6, no. 1, pages 18–26, March 1993.

[Jegou 91] P. Jegou. *Contribution à l'étude des Problèmes de Satisfaction de Contraintes: Algorithmes de propagation et de Résolution; Propagation de Contraintes dans les Réseaux Dynamiques.* PhD thesis, Université des Sciences et Techniques du Languedoc, Montpellier, Janvier 1991.

[Kambhampati 93] S. Kambhampati. *On the utility of systematicity: understanding tradeoffs between redundancy and commitment in partial-ordering planning.* In Proceedings Spring Symposium AAAI, 1993.

[Kambhampati 95] S. Kambhampati & B. Srivastava. *Universal Classical Planner: An algorithm for unifying State-space and Plan-space planning.* In Proceedings 3rd European Workshop on Planning (EWSP-95), pages 81–94, 1995.

[Kautz 91] H. Kautz & P. Ladkin. *Integrating Metric and Qualitative Temporal Reasonning.* In Proceedings AAAI-91, pages 260–267, 1991.

[Keng 89] N. Keng & D. Yun. *A Planning/Scheduling Methodology for the Constrained Resource Problem.* In Proceedings IJCAI-89, pages 998–1003, 1989.

[Klein 95] I. Klein, P. Jonsson & C. Bäckström. *Tractable Planning for the Assembly Line.* In Proceedings 3rd European Workshop on Planning (EWSP-95), pages 333–344, 1995.

[Knoblock 90] C.A. Knoblock. *Learning abstraction hierarchies for problem solving.* In Proceedings AAAI-90, pages 923–928, 1990.

[Knoblock 91a] C.A. Knoblock. *Search Reduction in Hierarchical Problem Solving.* In Proceedings AAAI-91, pages 686–691, 1991.

[Knoblock 91b] C.A. Knoblock, J. Tenenberg & Q. Yang. *Characterizing Abstraction Hierarchies for Planning.* In Proceedings AAAI-91, pages 692–697, 1991.

[Knoblock 93] C.A. Knoblock & Q. Yang. *A comparison of the SNLP and TWEAK planning algorithms.* In Proceedings Spring Symposium AAAI, 1993.

[Knoblock 94] C.A. Knoblock. *Automatically generating abstractions for planning.* Artificial Intelligence, vol. 68, pages 243–302, 1994.

[Kondrak 95] G. Kondrak & P. van Beek. *A Theorical Evaluation of Selected Backtracking Algorithms.* In Proceedings IJCAI-95, pages 541–547, August 1995. also: technical report TR94-10, University of Alberta, june 1994.

[Korte 89] N. Korte & R. Möhring. *An Incremental Linear-Time Algorithm for Recognizing Interval Graphs.* SIAM J. Comput., vol. 18, no. 1, pages 68–81, 1989.

[Kushmerick 94] N. Kushmerick, S. Hanks & D. Weld. *An Algorithm for Probabilistic Least-Commitment Planning.* In Proceedings AAAI-94, 1994.

[Laborie 94] P. Laborie. *Planifier avec des contraintes de ressources.* In Actes Secondes Rencontres Nationales des Jeunes Chercheurs en Intelligence Artificielle, pages 239–247, 1994.

[Laborie 95] P. Laborie & M. Ghallab. *Planning with Sharable Resource Constraints.* In Proceedings IJCAI-95, pages 1643–1649, 1995.

[Laruelle 94] H. Laruelle. *Planification Temporelle et Exécution de Tâches en Robotique.* PhD thesis, Université Paul Sabatier, Toulouse, Avril 1994.

[Lever 94] J. Lever & B. Richards. parc*PLAN: a Planning Architecture with Parallel Actions, Resources and Constraints.* In Proceedings 8th ISMIS, 1994.

[Lopez 91] P. Lopez. *Approche Energétique pour l'Ordonnancement de Tâches sous Contraintes de Temps et de Ressources.* PhD thesis, Université Paul Sabatier, Toulouse, septembre 1991.

[MacCarthy 93] B.L. MacCarthy & J. Liu. *Addressing the gap in Scheduling Research: a review of Optimization and Heuristic Methods in Production Scheduling.* Int. Journ. of Production Research, vol. 31, no. 1, pages 59–79, 1993.

[Mackworth 77] A.K. Mackworth. *Consistency in Networks of Relations.* Artificial Intelligence, vol. 8, pages 99–118, 1977.

[McAllester 91] D. McAllester & D. Rosenblitt. *Systematic nonlinear planning.* In Proceedings AAAI-91, pages 634–639, 1991.

[Meiri 91] I. Meiri. *Combining Qualitative and Quantitative Constraints in Temporal Reasonning.* In Proceedings AAAI-91, pages 260–267, 1991.

[Miller 85] D. Miller. *Planning by Search Through Simulations.* PhD thesis, Yale University, Dept of Computer Science, 1985.

[Missiaen 91] L. Missiaen. *Localized Abductive Planning with the Event Calculus.* PhD thesis, Dept of Computer Science, K.U.Leuven, 1991.

[Mohr 86] R. Mohr & T. Henderson. *Arc and Path Consistency Revisited.* Artificial Intelligence, vol. 28, no. 2, pages 225–233, March 1986.

[Montanari 74] U. Montanari. *Networks of Constraints: Fundamental Properties and Applications to Picture Processing.* Information Sciences, no. 7, pages 95–132, 1974.

[Motwani 88] R. Motwani, A. Raghunathan & H. Saran. *Covering Orthogonal Polygons with Star Polygons: The Perfect Graph Approach.* In Proc. 4th. Annual ACM Symposium on Computational Geometry, pages 211–223, 1988. also: Technical Report, University of California, Berkeley, CSD-87-384.

[Mounir-Alaoui 90] A. Mounir-Alaoui. *Raisonnement Temporel pour la Planification et la Reconnaissance de Situations.* PhD thesis, Université Paul Sabatier, Toulouse, Octobre 1990.

[Muscettola 87] N. Muscettola & S. Smith. *A Probabilistic Framework for Resource-Constrained Multi-Agent Planning.* In Proceedings IJCAI-87, pages 1063–1066, 1987.

[Muscettola 93] N. Muscettola. *HSTS: Integrating Planning and Scheduling.* In Intelligent Scheduling. Morgan Kaufmann, March 1993.

[Nilsson 80] N.J. Nilsson. Principles of artificial intelligence. Tioga Publishing Company, 1980.

[Parrod 92] Y. Parrod. *OPTIMUM: A Planning and Scheduling Tool for Managing Complex Projects.* Rapport technique, 1992. Doc. Interne Matra Marconi Space.

[Pednault 89] E. Pednault. *ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus.* In Proceedings First Knowledge Representation Conference, pages 324–332, 1989.

[Penberthy 92] J.S. Penberthy & D.S. Weld. *UCPOP: A Sound, Complete, Partial Order Planner for ADL.* In Proceedings 3rd International Conference on Knowledge Representation and Reasonning, Cambridge, MA, pages 103–114, 1992.

[Penberthy 94] J.S. Penberthy & D.S. Weld. *Temporal Planning with Continual Change.* In Proceedings AAAI-94, 1994.

[Pinedo 95] M. Pinedo. SCHEDULING: Theory, Algorithms and Systems. Prentice Hall, New Jersey, 1995.

[Purdom 83] P.W. Purdom. *Search Rearrangement Backtracking and Polynomial Average Time.* Artificial Intelligence, vol. 21, no. 1,2, pages 117–133, 1983.

[Raghunathan 89] A. Raghunathan. *Algorithms For Weakly Triangulated Graphs.* Rapport technique CSD-89-503, University of California, Berkeley, 1989.

[Roy 69] B. Roy. Algèbre moderne et théorie des graphes, tomes 1 et 2. Dunod, Paris, 1969.

[Sacerdoti 74] E.D. Sacerdoti. *Planning in a Hierarchy of Abstraction Spaces.* Artificial Intelligence, vol. 5, pages 115–135, 1974.

[Sacerdoti 75] E.D. Sacerdoti. *A Structure for Plans and Behaviours.* technical note 109, SRI, 1975.

[Sadeh 91] N. Sadeh. *Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling.* PhD thesis, Carnegie Mellon University, march 1991.

[Shoham 88] Y. Shoham. Reasoning about change. The MIT Press, Cambridge, MA, 1988.

[Smith 86] S.F. Smith, M.S. Fox & P.S. Ow. *Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory.* AI Magazine, vol. 7, no. 4, pages 45–61, 1986.

[Tarjan 84] R. Tarjan & M. Yannakakis. *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs.* SIAM Journal of Computing, no. 13, pages 566–579, 1984.

[Tate 94] A. Tate, B. Drabble & R. Kirby. O-Plan2: An Architecture for Command, Planning and Control. Intelligent Scheduling. Morgan-Kaufmann Publishing, 1994.

[Tessier-Badie 88] C. Tessier-Badie. *Contribution à l'étude des problèmes d'affectation de ressources: application au domaine spatial.* PhD thesis, ENSAE,Toulouse, 1988.

[Thiébaux 95] S. Thiébaux. *Contribution à la planification sous incertitudes et en temps contraint.* PhD thesis, Université de Rennes I, 1995.

[van Beek 90] P. van Beek. *Reasonning about Qualitative Temporal Information.* In Proceedings AAAI-90, Boston, pages 728–734, August 1990.

[Vere 83] S. Vere. *Planning in Time: Windows and Durations for Activities and Goals.* IEEE Transaction on Pattern Analysis and Machine Intelligence, vol. 5, no. 3, pages 246–267, may 1983.

[Vidal 95] T. Vidal. *Le Temps en Planification et en Ordonnancement: Vers une Gestion Complète et Efficace de Contraintes Hétérogènes et Entachées d'Incertitude.* PhD thesis, Université Paul Sabatier, Toulouse, 1995.

[Vilain 86] M. Vilain & H. Kautz. *Constraint Propagation Algorithms for Temporal Reasoning.* In Proceedings AAAI-86, pages 377–382, 1986.

[Weld 95] D. Weld. *An introduction to least commitment planning.* AI Magazine, pages 27–61, 1995.

[Wilkins 88] D.E. Wilkins. Practical planning. Morgan Kaufmann, San-Mateo, CA, 1988.

[Wilkins 92] D.E. Wilkins & R.V. Desimone. Applying an AI Planner to Military Operations Planning. Intelligent Scheduling. Morgan-Kaufmann Publishing, 1992.

[Yang 90] Q. Yang & J.D. Tenenberg. *ABTWEAK : Abstracting a Nonlinear Least Commitment Planner.* In Proceedings AAAI-90, volume 1, pages 204–209, 1990.

[Yang 92] Q. Yang. *A Theory of conflict Resolution in Planning.* Artificial Intelligence, vol. 58, pages 361–392, 1992.