N° d'ordre : 212 Année 1997

THÈSE

présentée en vue de
l'obtention du titre de

DOCTEUR
de
L'ÉCOLE NATIONALE SUPÉRIEURE
DE L'AÉRONAUTIQUE ET DE L'ESPACE

SPÉCIALITÉ : INFORMATIQUE

par

Pierre DAGO

EXTENSION D'ALGORITHMES DANS LE CADRE DES PROBLÈMES DE SATISFACTION DE CONTRAINTES VALUÉS : APPLICATION À L'ORDONNANCEMENT DE SYSTÈMES SATELLITAIRES

Soutenue le 12 juin 1997 devant la Commission d'Examen :

MM.	M.	MINOUX	Président
	E.	BENSANA	Directeur de thèse
	J.	ERSCHLER	Rapporteur
	Ph.	JEGOU	Rapporteur
	B.	LECUSSAN	
	M.	RUEHER	
Mme	M.-C.	VILAREM	

Je remercie tout d'abord l'ensemble des membres du jury pour l'intérêt qu'ils ont manifesté pour ma formation doctorale. Je remercie spécialement mes directeurs de thèse, M. Bernard Lecussan et M. Eric Bensana, qui, par leur soutient "logistique" et administratif, ont rendu cette thèse possible.

Je remercie l'ensemble des ingénieurs et stagiaires du département d'automatique du CERT pour leur acceuil, leur convicialité et leur disponibilité.

Au sein du département d'automatique, je tiens tout particulièrement à remercier les secrétaires M. Pierre Guillemot, Mme Lilliane Arendo, Mme Michèle Dupuy ainsi que l'ingénieur système M. Michel Grau qui mettent tant de bonne volonté et d'énergie pour vous rendre la vie plus simple et plus agréable (et ils y parviennent).

Je remercie les membres du groupe "optimisation combinatoire" du CERT pour tout ce qu'ils m'ont apris et la bonne humeur qu'ils ont apporté à mon travail. Je remercie plus particulièrement Mme Caroline Thierry et Mlle Karine Rota du soutient moral et de l'ouverture sur le monde de la Gestion de Production qu'elles m'ont apporté.

Je remercie enfin, et surtout, M. Gérard Verfaillie, la seule personne a avoir participé à mon travail : je dois à la pertinence de ces critiques et suggestions l'aboutissement et le succes de cette thèse.

A mes femmes...

ma mère, ma mienne, ma môme.

Table des matières

1 Introduction 1

2 Problèmes de Satisfaction de Contraintes 5

2.1 CSP classiques . . . 6

2.1.1 Modèle CSP . . . 6

2.1.2 Algorithme standard . . . 7

2.1.3 Prévision des échecs . . . 10

2.1.4 Analyse des échecs . . . 14

2.1.5 Mémorisation des échecs . . . 17

2.1.6 Heuristiques . . . 27

2.1.7 Résumé . . . 28

2.2 CSP valués . . . 29

2.2.1 Modèle CSP valués . . . 29

2.2.2 Algorithmes . . . 31

2.2.3 Prévision des échecs . . . 34

2.2.4 Résumé . . . 43

2.3 Conclusion . . . 44

3 VCSP et contraintes induites 45

3.1 Contraintes induites 46
3.1.1 Définition des contraintes induites 46
3.1.2 Réduction et augmentation des contraintes induites 50
3.1.3 Agrégation des contraintes induites 55
3.1.4 VCSP induits 60
3.2 Cohérence locale et VCSP 63
3.2.1 Contraintes induites et prévision des échecs 63
3.2.2 Efficacité des prévisions 64
3.2.3 Définitions 66
3.2.4 Algorithme 67
3.2.5 Expérimentations 72
3.3 Nogood Recording pour VCSP 78
3.3.1 Algorithme 80
3.3.2 Expérimentations 83
3.4 Backtrack dynamique et VCSP 85
3.4.1 Algorithme 86
3.4.2 Terminaison 89
3.4.3 Expérimentations 98
3.5 Résultats généraux 99

4 Ordonnancement de systèmes satellitaires: définition du problème et solutions actuelles 103

4.1 Présentation du problème 103
4.1.1 L'observation de la Terre par satellite 103
4.1.2 Contraintes 106
4.1.3 Formalisation des objectifs 107

4.2 L'ordonnancement de systèmes satellitaires : solutions existantes . . . 108
4.2.1 Aide à la décision . . . 108
4.2.2 SPOT 1 . . . 108
4.2.3 OSCAR . . . 109
4.2.4 Recherche tabou . . . 110
4.2.5 Pseudo-programmation dynamique . . . 110
4.2.6 Positionnement de l'étude . . . 111

5 L'ordonnancement à court terme **113**
5.1 Ordonnancement journalier du satellite SPOT 5 . . . 113
5.1.1 Problème posé et objectifs . . . 113
5.1.2 Modélisation du problème journalier . . . 114
5.1.3 Jeux d'essai . . . 115
5.2 Contraintes locales . . . 116
5.2.1 VCSP particulier . . . 122
5.2.2 Propositions . . . 123
5.3 Contrainte globale . . . 128
5.3.1 Contrainte mémoire et optimisation multicritère . . . 128
5.3.2 Structure de valuation bicritère . . . 130
5.3.3 Propositions . . . 134
5.3.4 Coopération . . . 138

6 Conclusion, générale **143**

A Lexique **i**

B Espérance de solutions v

C Algorithmes détaillés vii

C.1 CSP Classiques vii

C.1.1 Le Backtrack vii

C.1.2 Le Forward-checking viii

C.1.3 Conflict Directed Backjumping viii

C.1.4 Le Nogood Recording xii

C.2 CSP Valués xii

C.2.1 Le Branch and Bound en profondeur d'abord xii

C.2.2 Le Forward-checking valué xii

Bibliographie xvii

Chapitre 1

Introduction

L'amélioration des méthodes d'ordonnancement automatique s'accompagne toujours de l'accroissement en difficulté et en taille des problèmes que l'on cherche à résoudre. L'ordonnancement des satellites d'observation ne fait pas exception ; le besoin associé est bien trop important au regard des possibilités actuelles.

Les satellites d'observation de la Terre ont pour mission de réaliser un ensemble de photographies à l'occasion du survol des zones considérées. Ces satellites sont semi-automatiques dans la mesure où ils reçoivent une fois par jour[1] des directives provenant des opérateurs qu'ils appliquent par l'activation précise de leur charge utile. Ces directives regroupées dans un plan de production définissent essentiellement les images à réaliser ainsi que leur mode de réalisation. Comme l'ensemble des demandes d'images est loin d'être réalisable sur l'horizon considéré, la production du plan revient à sélectionner un ensemble optimal de photographies sous la contrainte des limitations matérielles du système satellitaire.

La complexité des contraintes matérielles et la taille des problèmes interdisent une résolution manuelle réellement optimale ; d'où la nécessité d'introduire des outils automatiques pour soutenir le travail des opérateurs.

Le problème posé consiste donc, pour simplifier, à ordonnancer les tâches de réalisation des images en leur attribuant une date de réalisation. Mais il s'agit d'un ordonnancement auquel s'appliquent mal les théories classiques ("job-shop", etc.) car il est posé en des

1. La journée est un ordre de grandeur.

termes très spécifiques tant dans sa structure que dans ses objectifs :

- toutes les tâches ne sont pas réalisables ;
- les durées associées à l'ordonnancement des tâches n'interviennent pas dans les critères d'optimisation ;
- les opportunités de réalisation associées aux survols sont discrètes ;
- etc.

Nous considérons donc ce problème de manière plus générale, comme un problème d'optimisation combinatoire sous contraintes. Dans cette thèse, nous nous intéressons plus précisément à la résolution des "**Problèmes de Satisfaction de Contraintes valués**" (CSP valués ou VCSP). Cette modélisation extrêmement simple, intuitive et riche s'adapte facilement à une grande variété de problèmes d'optimisation. De nombreux travaux ont par ailleurs fourni toute une bibliothèque d'algorithmes de résolution.

Dans le cadre classique de la satisfaction des CSP, l'efficacité de ces méthodes n'est plus à démontrer et leur domaine d'application est assez bien connu. L'introduction d'un critère d'optimisation dans le modèle CSP qui donne naissance aux CSP valués est cependant trop récente pour que les possibilités et les limites d'efficacité soient clairement identifiées. L'ordonnancement des satellites d'observation fait partie des CSP valués à la frontière des compétences actuelles en matière d'optimisation. Cette application est pour nous un prétexte au développement d'outils de résolution nouveaux étendant le champ des CSP valués accessibles.

Cette étude se place donc volontairement dans le cadre de la résolution complète de problèmes d'optimisation modélisés sous forme de CSP valués.

Dans le deuxième chapitre, nous présentons la définition du modèle CSP ainsi que les acquis algorithmiques classiques pour la résolution de ces problèmes : backtrack, prévision des échecs et k-cohérence, analyse des échecs et mémorisation. Nous rapellons ensuite l'extension que constitue le modèle VCSP et les quelques algorithmes de résoultion définis à ce jour. On verra, en particulier, que les limitations actuelles proviennent essentiellement de la difficulté à mettre en œuvre des contraintes induites dans le cas général.

Le troisième chapitre expose les moyens de pallier cette carence, à savoir un modèle

de contraintes induites pour VCSP accompagné des propriété indispensables à leur utilisation. Puis nous proposons les extensions algorithmiques qui en découlent.

L'intérêt des contraintes induites pour les CSP valués est illustré dans le quatrième et le cinquième chapitres. Elles sont mises en œuvre pour résoudre les problèmes posés par l'ordonnancement à court terme du futur satellite SPOT 5.

Chapitre 2

Problèmes de Satisfaction de Contraintes

Notre travail s'inscrit dans le cadre des Problèmes de Satisfaction de Contraintes (CSP) [AS94]. Cette approche issue de l'Intelligence Artificielle offre une modélisation générale permettant de formuler simplement de nombreux problèmes de décision. Ce formalisme regroupe une large classe de problèmes, depuis les problèmes combinatoires ou les puzzles arithmétiques jusqu'aux problèmes classiques d'affectation ou d'ordonnancement. Tous se caractérisent par leur nature fortement combinatoire.

L'approche CSP consiste aussi en un ensemble de techniques de résolution exploitant les particularités de ce formalisme. Ces travaux ont également donné naissance à des langages génériques intégrant la programmation par contraintes (Ilog Solver, Chip, Prolog V pour les plus récents).

Plus récemment, le modèle CSP a fait l'objet de nombreuses extensions lui permettant de formuler une plus grande variété de problèmes [FDP95, FW92], dont les problèmes d'optimisation en particulier. Ces problèmes peuvent être regroupés sous le formalisme des CSP valués [SFV95] qui nous intéresse plus précisément.

Les deux sections qui suivent présentent le formalisme CSP et dressent un état de l'art concernant les méthodes de résolution complètes des CSP classiques et valués.

2.1 CSP classiques

2.1.1 Modèle CSP

Nous appelons CSP le couple $(\mathbb{V}, \mathbb{C})$ constitué d'un ensemble de variables $\mathbb{V}$, et d'un ensemble de contraintes $\mathbb{C}$.

Un domaine, ensemble fini de valeurs, est associé à chaque variable. Nous notons indifféremment x^i la valeur i de la variable x, ou l'affectation de la variable x à la valeur i $(x = i)$[1].

Une contrainte est un ensemble de clauses désignant pour un ensemble de variables $\{x_{c1}, x_{c2}, \ldots, x_{cn}\}$ les n-uplets interdits. En extension, elle prend la forme :

$$\begin{aligned} c \;=\; & [(x_{c1} \neq i_{11}) \vee \ldots \vee (x_{cn} \neq i_{n1})] \\ \wedge\; & [(x_{c1} \neq i_{12}) \vee \ldots \vee (x_{cn} \neq i_{n2})] \\ & \ldots \\ \wedge\; & [(x_{c1} \neq i_{1m}) \vee \ldots \vee (x_{cn} \neq i_{nm})] \end{aligned}$$

Nous notons $\tilde{c} = \{x_{c1}, x_{c2}, \ldots, x_{cn}\}$ l'ensemble des variables liées par la contrainte c.

Une *solution* du CSP est une affectation complète satisfaisant toutes les contraintes.

Nous appelons *affectation partielle* un sous-ensemble A de variables affectées $A = \{x_k = i_k\}$. Lorsque toutes les variables sont affectées, l'affectation est dite *complète*. Nous notons par $\tilde{A} = \{x_k\}$ le sous-ensemble des variables concernées.

Soit A une affectation partielle, et $x \notin \tilde{A}$ une variable libre, l'affectation $A \cup \{x^i\}$ est appelée *extension* de A avec la valeur x^i. Soit l'affectation partielle A' telle que $\tilde{A'} = \mathbb{V} - \tilde{A}$, $A \cup A'$ est une *extension complète* de A. Nous dirons enfin que deux affectations sont *compatibles* si elles ont une extension complète commune (une variable représentée dans les deux affectations est affectée à la même valeur).

Une affectation partielle qui satisfait toutes les contraintes qui reposent sur ses variables est dite *localement cohérente* : $\forall c \in \mathbb{C}/\tilde{c} \subset \tilde{A}, A \models c$. Elle est *globalement cohérente* s'il existe une extension complète cohérente (donc solution).

1. On trouvera en annexe un petit lexique des notations utilisées.

La figure 2.1 donne un exemple de CSP sans solution à 5 variables et 5 contraintes.

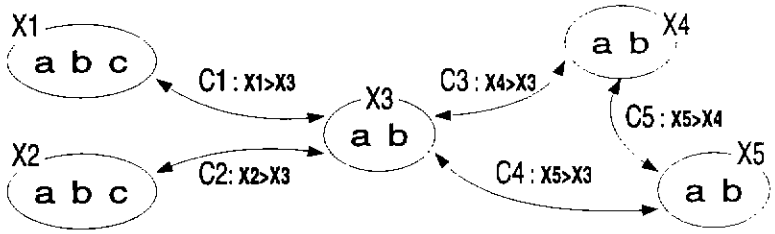

FIG. 2.1 – *Exemple.*

2.1.2 Algorithme standard

Les algorithmes auxquels nous nous intéressons visent à exhiber une solution du CSP (problème dit de satisfaction).

Le "**backtrack**" est l 'algorithme standard de résolution d'un problème de satisfaction de contraintes dont dérivent tous les algorithmes complets. Il s'agit du moins de la trame commune à toutes les résolutions qui nécessitent une énumération faute de pouvoir démontrer autrement l'existence ou non de solutions (en particulier lorsque les problèmes sont trop complexes pour permettre un calcul mathématique direct des solutions). Il s'appuie sur la notion d'arbre de recherche :

- la racine de l'arbre est une affectation vide ;
- les nœuds sont des affectations partielles (la racine est le premier) ;
- si un nœud ne correspond pas à une affectation complète, une variable non représentée dans l'affectation est désignée pour développer des branches vers autant de nœuds fils que d'extensions possibles avec les valeurs de cette variable ;
- les feuilles de l'arbre sont dès lors constituées de l'ensemble des affectations complètes possibles.

L'algorithme du backtrack construit et parcourt, en profondeur d'abord, un arbre de recherche. Afin d'éviter une exploration exhaustive de toutes les feuilles[2], le Backtrack

2. Pour des problèmes limités à quelques dizaines de variables et de contraintes, il faudrait compter plusieurs dizaines d'années.

s'appuie sur le lemme suivant :

Lemme 2.1 *Une affectation localement incohérente est globalement incohérente.*

En d'autres termes, les feuilles descendantes d'une affectation localement incohérente ne sont pas des solutions. L'algorithme du Backtrack, sous forme d'une fonction récursive prenant en argument l'affectation à étendre, est décrit par :

bt(A)

- Si A est complète alors A est une solution
- Sinon
  - Soit $x \notin \tilde{A}$
  - Pour chaque valeur i de x,
    Si **bt-Evalue**$(A \cup \{x = i\})$ alors **bt**$(A \cup \{x = i\})$

La fonction **bt-Evalue**$(A \cup \{x = i\})$ consiste ici à détecter une incohérence locale de $A \cup \{x = i\}$, soit à vérifier que cette affectation ne viole aucune contrainte. Elle renvoie $VRAI$ si l'affectation est cohérente.

Appelée avec l'affectation vide, cette fonction détecte toutes les solutions et termine lors de la fin de l'exploration de tout l'arbre de recherche. Il est important de noter que la terminaison de cette procédure provient du parcours arborescent : grâce à ce procédé, une branche ou affectation partielle n'est explorée qu'au plus une fois (et leur nombre est fini). On arrête généralement la procédure dès la première solution trouvée. Cette procédure doit son nom au retour arrière qu'elle effectue sur les choix d'affectation des variables à chaque échec de l'évaluation. On appellera plus généralement "backtrack" une procédure suivant ce schéma, construisant progressivement une affectation partielle par des instanciations successives de variables et revenant en arrière sur le choix des affectations lorsqu'une incohérence globale est constatée.

Exemple :

La figure 2.2 donne l'arbre produit et exploré sur l'exemple 2.1, en considérant un ordre statique pour le choix des variables candidates pour l'extension.

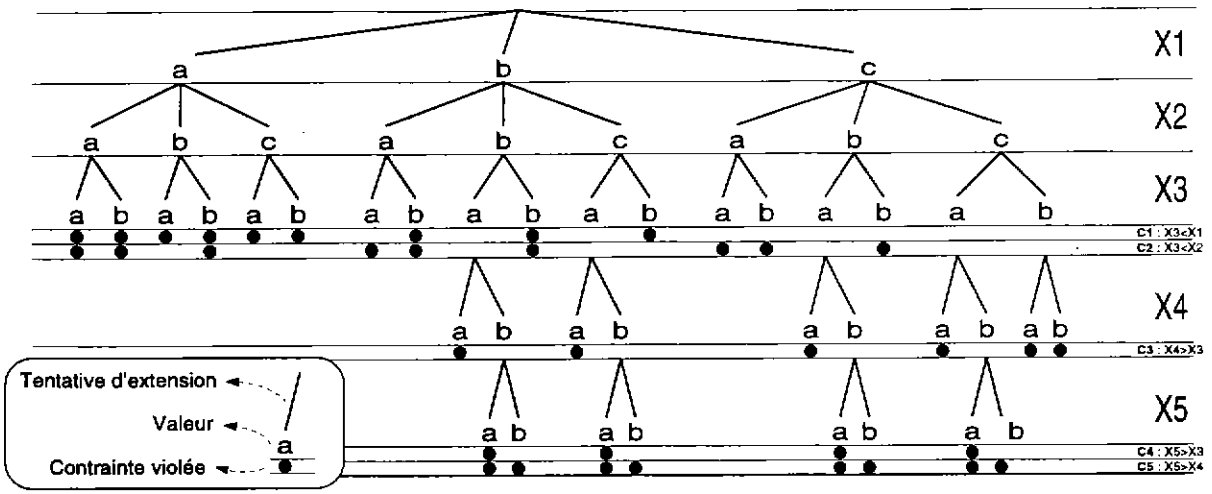

FIG. 2.2 – *Recherche arborescente du Backtrack.*

La rapidité d'obtention des solutions est directement liée à la portion de l'arbre de recherche effectivement parcourue par la recherche. Plus celle-ci est réduite, plus les solutions sont obtenues rapidement. Or la taille de l'arbre de recherche exploré dépend uniquement de deux points :

- de la capacité à détecter tôt l'incohérence globale de l'affectation partielle explorée ;
- de l'arbre de recherche effectivement construit, déterminé par le choix de l'ordre d'affectation des variables et de l'ordre d'examen de leurs valeurs.

Bien évidemment, il est impossible de prévoir quelles affectations mènent directement à une solution. De fait, les méthodes s'attachant au second point sont uniquement heuristiques. Par ailleurs, de nombreuses techniques ont été développées afin d'améliorer l'efficacité de la fonction **bt-Evalue**. Nous en donnons un rapide résumé.

Dans ce chapitre et celui qui suit, nous nous intéresserons à de nombreux algorithmes pour lesquels seule une description schématique sera donnée. Les implémentations pratiques peuvent s'écarter sensiblement de ce que nous décrivons. Les algorithmes conservent pour la majorité d'entre eux la structure de recherche arborescente de **bt** et ne diffèrent que par leur fonction d'évaluation des incohérences. Dans ces cas, seule cette fonction notée **xx-Evalue** sera présentée. Une description détaillée des principaux algorithmes est fournie en annexe.

2.1.3 Prévision des échecs

La technique consiste, à peu de choses près, à effectuer à chaque nœud des explorations partielles, d'une profondeur réduite, sur les variables encore non instanciées. De telles méthodes ont pour effet de détecter plus tôt les incohérences. En effet, lorsqu'une exploration partielle conduit à la constatation de l'incohérence du sous-problème correspondant, on peut conclure à l'incohérence globale de l'affectation de départ. Comme nous le verrons, la déduction de l'incohérence globale n'est pas toujours directe : des informations intermédiaires d'incohérence locale, similaires à des contraintes, sont déduites des explorations partielles.

Le Forward-checking

L'algorithme le plus utilisé du fait de sa simplicité et de son moindre coût en temps de calcul est celui du "**Forward-checking**" [HE80]. Il consiste à explorer toutes les variables non instanciées et s'appuie sur le lemme suivant pour détecter une éventuelle incohérence globale :

Lemme 2.2 *S'il existe une extension cohérente avec toutes les variables, il en existe avec chacune d'entre elles.*

Réciproquement, une affectation est globalement incohérente s'il existe une variable avec laquelle elle ne peut pas être étendue.

L'algorithme du Forward-checking s'obtient en remplaçant la fonction **bt-Evalue**(A) par **fc-Evalue**(A, x) appliquée à toutes les variables $x \notin \tilde{A}$:

fc-Evalue(A, x)

- Si pour toute valeur i de x, **bt-Evalue**$(A \cup \{x = i\}) = FAUX$ alors renvoie $FAUX$
- Sinon renvoie $VRAI$

Exemple :

La figure 2.3 donne un aperçu de l'efficacité de cette méthode sur l'exemple 2.1. Les explorations effectuées lors de l'évaluation sont cependant omises ; de leur importance comparée à la restriction de l'arbre de recherche dépendront les gains en temps de calcul.

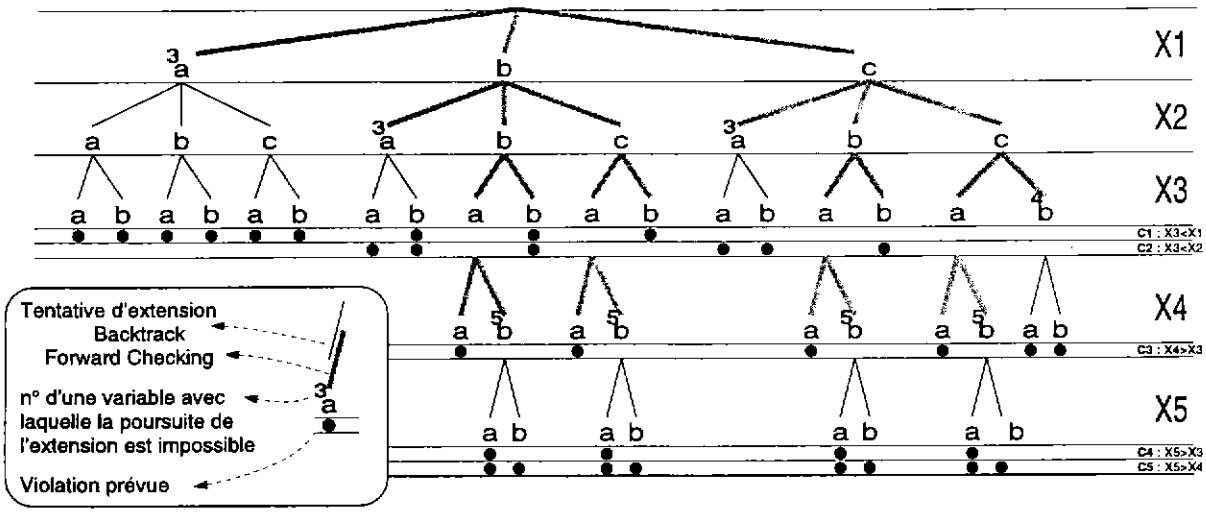

FIG. 2.3 – *Forward-checking.*

En particulier, lorsque l'échec est prévu avec la variable suivante dans l'ordre d'affectation, le Forward-checking n'apporte rien au Backtrack.

> *Suivons les premiers pas de la recherche :*
>
> *X_1 est tout d'abord affectée à a, et la cohérence globale de l'affectation est évaluée.* **fc-Evalue** *teste si, pour chaque variable, il existe au moins une valeur compatible avec $\{X_1 = a\}$. En l'occurrence, bien que localement cohérente, cette affectation ne peut pas être étendue avec X_3 : un retour arrière est effectué. On s'économise ainsi les explorations inutiles de toutes les valeurs de la variable X_2.*
>
> *Suite au retour arrière, X_1 est affectée à b. Cette fois, aucune incompatibilité n'est détectée (on peut considérer que les tests effectués le sont en pure perte).*
>
> *L'exploration est poursuivie par l'affectation de X_2 à a et la cohérence de $\{X_1 = b, X_2 = a\}$ testée.* **fc-Evalue** *constate l'incompatibilité de toutes les valeurs de X_3 avec cette affectation : un retour arrière est provoqué. Il faut remarquer que l'on n'a pas réellement réduit l'exploration puisque X_3 était de toute façon la prochaine variable avec laquelle l'extension devait être testée.*

Ce petit exemple, du fait de son faible nombre de variables, met surtout en évidence les risques du Forward-checking. En pratique, cette méthode s'avère généralement très efficace car elle prévoit longtemps à l'avance des violations inévitables.

Maintien de cohérence

D'une manière plus générale, la détection d'une incohérence globale se fait en exhibant un ensemble de variables avec lesquelles il est impossible d'étendre l'affectation considé-

rée sans produire une incohérence locale. Le Forward-checking se borne à explorer les ensembles constitués d'une unique variable. Les procédures examinant tous les ensembles de k variables sont les méthodes dites de renforcement de "**cohérence**" [Fre82]. Elles s'appuient sur la notion de contraintes induites.

Définition 2.3 *Nous dirons d'une contrainte qu'elle est induite si les n-uplets qu'elle interdit n'appartiennent à aucune solution.*

Par définition, on sait *a priori* qu'une affectation qui viole une contrainte ne peut pas conduire à une solution. On peut constater *a posteriori* qu'un n-uplet ne peut pas être étendu vers une solution. Alors, de même que pour une contrainte d'origine, on a l'assurance qu'une affectation contenant ce n-uplet (violant par conséquent la contrainte induite associée) ne peut pas non plus conduire à une solution. Une contrainte induite est donc équivalente à une contrainte d'origine, cependant, le n-uplet qu'elle interdit n'est pas explicitement rejeté par la définition des contraintes du problème, mais déduit de ces dernières.

Plus précisément, les méthodes de renforcement de cohérence qui examinent les ensembles de k variables procèdent par étapes : k-1 variables sont affectées et l'on cherche si la poursuite de l'extension est impossible avec une k^{è}*me* variable. Auquel cas une contrainte induite est posée pour interdite l'affectation de k-1 variables. Lorsque toutes les affectations de k-1 variables sont examinées, on passe aux affectations de k-2 variables et l'on vérifie la cohérence des extensions avec une k-1^{è}*me* variable (en s'aidant bien sûr des contraintes posées lors de l'étape précédente). Et ainsi de suite jusqu'à l'éventuelle preuve de l'incohérence globale de l'affectation vide. Ceci nous conduit naturellement à la définition de la "**k-cohérence**" :

Définition 2.4 (k-cohérence) *Un CSP est k-cohérent si, pour chaque affectation localement cohérente de k-1 variables, il existe une extension localement cohérente avec toute $k^{ème}$ variable.*

De même, lorsqu'en cours de recherche un sous-ensemble de variables est affecté, nous dirons que le CSP est k-cohérent si pour chaque extension cohérente avec k-1 variables il existe une extension localement cohérente avec toute k^{ème} variable.

En d'autres termes, tout développement cohérent de l'arbre sur k-1 noeuds doit pouvoir se poursuivre sur un k^{ème} noeud sans violer de contraintes. Lorsque cela s'avère impossible,

il convient de poser une contrainte interdisant la branche de k-1 noeuds. Le lemme 2.2 nous assure que cette contrainte est induite par le CSP car cette branche est globalement incohérente. Un telle procédure ne modifie donc pas l'ensemble des solutions.

Le Forward-checking correspond au maintient de 1-cohérence et conduit à l'interdiction de l'affectation courante.

Remarque 2.1 *Cette contrainte induite ne mérite pas d'être ajoutée dans la mesure où le parcours arborescent nous assure que son affectation ne sera jamais plus essayée.*

L'objectif est donc d'ajouter des contraintes induites[3] afin de renforcer progressivement la k-cohérence. Les contraintes posées par un renforcement de k-cohérence sont d'arité k-1 et peuvent servir à un renforcement de (k-1)-cohérence, jusqu'à la 1-cohérence. Si cette dernière conclut que l'affectation courante est globalement incohérente le mécanisme de retour arrière s'enclenche. Sinon, le développement de l'affectation courante se poursuit, avec le bénéfice des contraintes induites ajoutées au problème (plus il y a de contraintes, plus l'examen des violations détecte tôt une incohérence globale).

Nous pouvons donner une version simpliste des algorithmes afin de préciser le fonctionnement du renforcement d'une k-cohérence :

k-c(A)

- Tant qu'il est possible d'ajouter une contrainte :
  - S'il existe une affectation A' de $k-1$ variables, et une k^{ème} variable x telle que **fc-Evalue**$(A \cup A', x) = FAUX$
    alors pose une contrainte interdisant $A \cup A'$.
- Fin tant que

Ce traitement s'utilise de deux manières. La première consiste à effectuer un prétraitement du CSP de manière à le rendre k-cohérent (Appel de **k-c** avec l'ensemble vide comme argument). Le CSP résultant est *a priori* plus facile à résoudre. Dans le meilleur des cas, le renforcement de k-cohérence suffit à prouver l'absence de solution. Sinon, les contraintes supplémentaires produites permettent une détection anticipée des incohérences globales.

3. Lorsqu'une contrainte existe déjà entre les variables du n-uplet interdit, on peut se contenter d'ajouter le nouveau n-uplet à l'ancienne contrainte.

Une option plus lourde consiste à insérer ce traitement après chaque instanciation de variable de manière à ce que le problème résiduel constitué par l'affectation des variables libres soit toujours k-cohérent.

Remarque 2.2 *Il est important de noter que le mode de parcours arborescent n'examine qu'une fois chaque branche. Dès lors, une contrainte induite créée lors d'une affectation courante A et interdisant $A \cup A'$ ne sera plus jamais violée dès lors que la branche A aura été entièrement explorée.*

En pratique, de telles contraintes sont conservées uniquement tant que l'affectation A ayant servi à leur construction est en cours d'exploration.

L'algorithme de maintien de l'"**arc-cohérence**" (2-cohérence) [SF94, BR96], ainsi que de nombreuses variantes intermédiaires entre ce dernier et le Forward-checking (elles n'explorent qu'un sous-ensemble des paires de variables), sont réputés pour être de bons compromis entre le temps passé à détecter les incohérences globales et le nombre de noeuds développés. Les contraintes unaires qu'ils produisent sont aussi d'un traitement aisé : posées sur une variable, elles reviennent à supprimer une valeur de son domaine. Le problème du renforcement de la 2-cohérence présente en outre l'avantage d'être polynômial lorsque les contraintes sont binaires.

Sur notre exemple, un renforcement initial de l'arc-cohérence suffit à prouver l'absence de solution : lors de l'examen de $X_4 = a$, il apparaît qu'aucune extension n'est possible avec X_3, donc $X_4 = a$ est interdite par une contrainte unaire. De même, $X_4 = b$ est interdite car elle ne peut pas être étendue avec X_5. Finalement, X_4 ne peut pas être affectée, donc toutes les valeurs examinées seront interdites.

2.1.4 Analyse des échecs

La première technique consistait à prévoir l'échec de l'exploration. La seconde consiste à analyser un échec afin d'en découvrir les causes. En effet, lorsqu'une variable est sélectionnée pour étendre l'affectation courante, le Backtrack n'effectue un retour arrière qu'après avoir terminé l'exploration de toutes ses valeurs. Une analyse bien menée permettra un backtrack anticipé ("**backjumping**") s'il s'avère que la cause des échecs est en amont dans l'arbre de recherche.

Lorsque l'extension d'une affectation A avec une valeur x^i échoue, cela est dû à la violation d'une contrainte c. Cette extension restera impossible tant que les valeurs de A mises en cause dans cette contrainte sont présentes dans l'affectation courante.

Lorsque toutes les extensions avec les valeurs d'une variable x ont échoué, on dispose d'ensembles de variables affectées responsables chacun du rejet d'une valeur. Tant que le retour arrière n'a pas atteint l'une des variables de ces ensembles, une extension complète est impossible car aucune valeur ne peut être affectée à la variable x. Le backtrack fera donc un saut jusqu'à la valeur de la dernière variable affectée de l'union de ces ensembles. Cette union est la cause du backtrack : elle constituera l'ensemble responsable du rejet de la valeur sur laquelle se porte le backtrack.

L'algorithme du "**Graph-based backjumping**" [Dec90] se borne à considérer que les valeurs responsables sont celles de variables liées à x par des contraintes. Plus précis, le "**Conflict-directed backjumping**" [Pro93] ainsi que les méthodes s'appuyant sur la mémorisation des échecs considèrent les variables mises en jeu dans les contraintes effectivement violées.

L'algorithme du Conflict-directed backjumping que nous donnons s'appuie sur la procédure **cbj**(A) qui tente l'extension d'un affectation A et renvoie un ensemble de variables responsables de l'échec de l'extension. L'évaluation de la cohérence locale des affectations, **cbj-Evalue**, retourne un ensemble de variables provoquant l'éventuelle incohérence.

cbj(A)

- Si A est complète alors A est une solution
- Sinon
  - Soit $x \notin \tilde{A}$, $V = \emptyset$
  - Pour chaque valeur i de x,
    - $v \leftarrow$**cbj-Evalue**$(A \cup \{x = i\})$
    - Si $v = \emptyset$ alors $v \leftarrow$**cbj**$((A \cup \{x = i\}))$
    - Si $x \notin v$ alors renvoie v (backjumping)
    - $V \leftarrow V \cup v$
  - renvoie $V - \{x\}$ (échec de toutes les valeurs de x)

cbj-Evalue(A)

- Si **bt-Evalue**(A)= $VRAI$
  renvoie $\emptyset$
- Sinon
  Soit c une contrainte violée, renvoie $\tilde{c}$

Exemple :

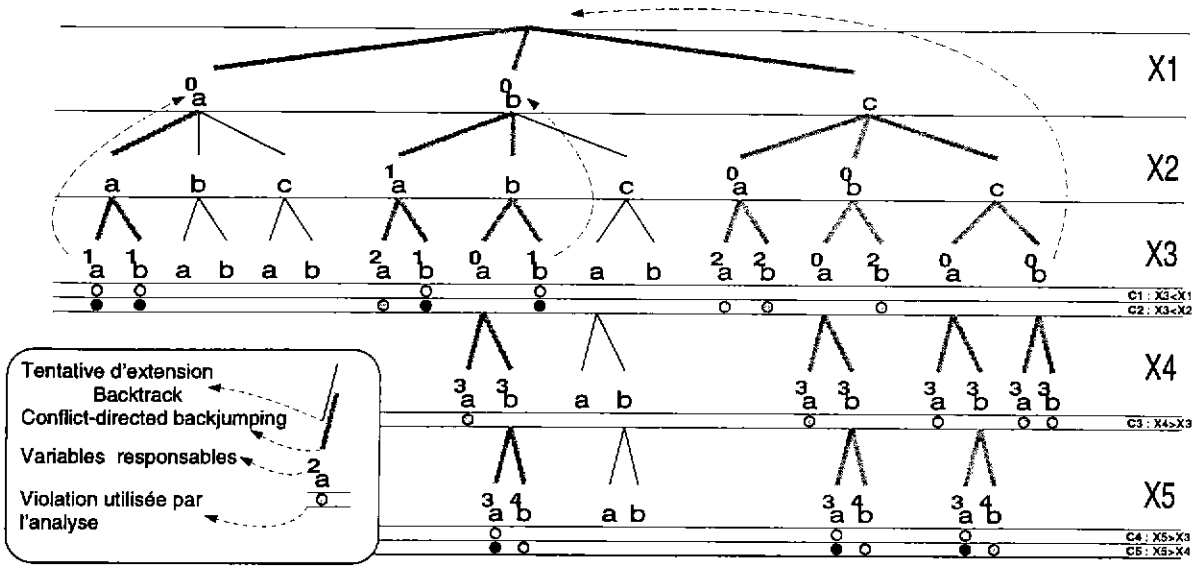

FIG. 2.4 – *Conflict-directed backjumping.*

La figure 2.4 illustre le Conflict-directed backjumping sur l'exemple. Pour plus d'efficacité, on cherche à identifier des variables responsables aussi proches de la racine que possible : lorsque plusieurs contraintes induisent une incohérence, on analyse la contrainte qui concerne les affectations les plus anciennes.

La première affectation partielle incohérente développée est l'affectation $\{X_1 = a, X_2 = a, X_3 = a\}$ *. L'analyse montre que cet échec peut être imputé à deux contraintes* C_1 *et* C_2*. Nous décidons de considérer* C_1 *comme seule responsable : les affectations des variables* X_1 *et* X_3 *sont déclarées responsables. Puisque la dernière affectation* $(X_3 = a)$ *est impliquée, la valeur suivante de* X_3 *est explorée.*

L'examen de $\{X_1 = a, X_2 = a, X_3 = b\}$ *conduit à un échec pour les mêmes raisons que précédemment. Cette fois, toutes les valeurs de* X_3 *ont été tentées sans succès;* X_3 *est retirée de l'ensemble des variables responsables. Un retour arrière naturel conduirait à revenir sur l'affectation de* X_2 *et à explorer ses autres valeurs. Or cette variable ne fait pas partie de l'ensemble des responsables : quelle que soit l'affectation de* X_2 *et tant que* X_1 *n'est pas modifiée, l'affectation courante est globalement incohérente.* X_2 *est désaffectée, mais l'exploration inutile de ses autres valeurs est ignorée. Le retour arrière remonte donc directement jusqu'à* X_1 *sur les valeurs de laquelle se poursuit l'exploration.*

2.1.5 Mémorisation des échecs

La dernière grande famille d'algorithmes complets de satisfaction cherche à mémoriser les causes des échecs afin d'éviter de reproduire dans la suite de l'exploration les conditions de ces échecs. Plus précisément, la connaissance d'une affectation partielle responsable d'une incohérence globale peut être réutilisée pour déduire l'incohérence globale de toutes ses extensions.

Les algorithmes d'analyse des échecs utilisent cette information pour interdire les extensions immédiatement après la découverte de l'échec et jusqu'à ce que l'affectation partielle responsable n'appartienne plus à l'affectation courante. Par la suite, cette affectation responsable pourra être reproduite : sur l'exemple du Conflict-directed backjumping, l'exploration de la branche $(x_1 = b, x_2 = a)$ montre que x_3 ne peut pas être affectée si $x_2 = a$. Cela n'empêchera pas l'exploration plus tard de l'extension de $(x_1 = b, x_2 = a)$ avec les valeurs de x_3. Les principes de mémorisation ont pour but de conserver les analyses effectuées (en ce sens ils sont une extension des méthodes précédentes) afin de les réutiliser dans la suite de l'exploration. En l'occurrence, $x_2 = a$ sera une affectation interdite jusqu'à la fin de la recherche.

Il y a cependant des moyens pour mémoriser les informations acquises lors d'une recherche autrement que sous forme d'une affectation inconsistante. En particulier, les diagrammes de décision binaires, que les avancées technologiques ont remis en lice, offrent une structure de données permettant de stocker intégralement une exploration [Bry86]. Plus précisément, ils construisent l'équivalent d'une fonction dont les paramètres sont les valeurs des variables et dont l'image est la cohérence ou non de l'affectation.

Le nogood

La notion centrale de ces algorithmes est celle du "**nogood**" [SV93, SV94a, Dec90] :

Définition 2.5 *Un nogood est une paire (A, C) où A est une affectation partielle appelée "affectation" du nogood, et C un ensemble de contraintes appelé "justification" du nogood.*

Un nogood (A, C) est induit par un CSP si toute extension complète de A viole au moins une contrainte de C. En d'autres termes, A est une affectation globalement incohérente du CSP réduit aux contraintes de C.

Considérons notre exemple. Aucune affectation complète contenant $\{X_3 = b\}$ *ne peut être solution car ni* C_3*, ni* C_4 *ne peuvent être satisfaites. Dès lors,* $(\{X_3 = b\}, \{C_3, C_4\})$*,* $(\{X_3 = b\}, \{C_3\})$ *ou* $(\{X_3 = b\}, \{C_4\})$ *sont des nogoods induits.*

Pour déterminer les incohérences globales dans un CSP, nous ne nous intéressons bien sûr qu'aux nogoods qu'il induit. Aussi, le terme "nogood" désignera implicitement un "nogood induit".

Les propriétés qui suivent vont nous permettre de manipuler les nogoods.

Propriétés 2.6 (classement) *Soit* $(A, \mathcal{C})$ *un nogood (induit),* $A' \supset A$ *et* $\mathcal{C}' \supset \mathcal{C}$*.*

$(A', \mathcal{C})$ *et* $(A, \mathcal{C}')$ *sont des nogoods (induits).*

Démonstration : (2.6) *Le premier point est évident : toute extension complète de* A' *est extension complète de* A*. Le second point aussi car si une contrainte de* $\mathcal{C}$ *est nécessairement violée, il en va de même pour* $\mathcal{C}'$

Propriété 2.7 (construction) *Quelle que soit l'affectation* A*, si* A *viole des contraintes dans un ensemble* $\mathcal{C}$*,* $(A, \mathcal{C})$ *est un nogood (induit).*

Démonstration : (2.7) *Il s'agit de l'équivalent de la propriété : incohérence locale implique incohérence globale.*

Propriété 2.8 (union) *Soit* x *une variable, et pour chaque valeur* i *de* x *un nogood* $(A_i \cup \{x = i\}, \mathcal{C}_i)$ *tel que* $x \notin \tilde{A}_i$*. Si les affectations* A_i *sont compatibles alors* $(\bigcup_i (A_i), \bigcup_i (\mathcal{C}_i))$ *est un nogood.*

Démonstration : (2.8) *Une extension complète de* $\bigcup_i (A_i)$ *affecte une valeur* k *à* x*. Par conséquent, elle est extension complète d'une affectation* $A_k \cup \{x^k\}$ *et viole une contrainte dans* $\mathcal{C}_k \subset \bigcup_i (\mathcal{C}_i)$*.*

Nous noterons $\bigvee_{x^i} (A_i \cup \{x = i\}, \mathcal{C}_i)$ le nogood résultant de l'application de cette propriété.

Propriété 2.9 (projection.1) *Soit* $(A, \mathcal{C})$ *un nogood et* C' *l'ensemble des contraintes de* $\mathcal{C}$ *dont tous les n-uplets interdits sont incompatibles avec* A*.* $(A, \mathcal{C} - C')$ *est un nogood.*

Propriété 2.10 (projection.2) *Soit (A, C) un nogood et A' l'ensemble des valeurs de A non représentées dans les n-uplets interdits par les contraintes de $\mathcal{C}$. $(A - A', C)$ est un nogood.*

Démonstration : (2.9,2.10) *Pour le premier point, il apparaît que toute extension complète de A ne viole pas les contraintes de $\mathcal{C}'$; elle en viole par conséquent une de $\mathcal{C} - \mathcal{C}'$. Pour le second, soit A'' une extension complète de $A - A'$ constituée de A'_{bis} une autre affectation des variables présentes dans A' et $\overline{A}$ une affectation des variables non représentées dans A. $A \cup \overline{A}$ viole une contrainte de $\mathcal{C}$, de même que $(A \cup \overline{A}) - A'$ car les valeurs de A' n'interviennent pas dans les violations des contraintes de $\mathcal{C}$. $A'' = ((A \cup \overline{A}) - A') \cup A'_{bis}$ viole donc aussi ces contraintes.*

Utilisation des nogoods, Nogood Recording

Un algorithme sur le modèle du Conflict-directed backjumping construit implicitement un nogood, lors de chaque échec, par la propriété de construction 2.7. Lorsque les extensions avec toutes les valeurs d'une variable ont échoué, il effectue l'union des nogoods construits pour chaque valeur grâce à la propriété d'union 2.8. Chaque nogood justifiant l'incohérence globale des extensions de leur affectation, l'algorithme effectue des retours arrière jusqu'à ce que l'affectation courante ne soit plus une extension du nogood. Pour plus d'efficacité, il convient de réduire l'affectation des nogoods à son strict minimum, afin qu'elle ne contienne que les seules valeurs conduisant à une incohérence globale. Ici interviennent les propriétés de projection 2.9 et 2.10. Il est cependant important de noter que ces projections sont inutiles si les nogoods sont construits de manière judicieuse lors des échecs : en ne considérant qu'une contrainte violée pour la justification du nogood, et le n-uplet mis en cause pour son affectation. De tels nogoods sont en fait projetés lors de leur construction.

La justification associée aux nogoods peut dès lors paraître superflue. Elle prendra de l'importance par la suite. Pour l'heure, elle n'est nécessaire que pour le traitement de problèmes dynamiques : la mémorisation présente un premier intérêt pour accélérer la recherche d'une solution initiale. Elle en présente un second si le problème devait être résolu à nouveau. De nombreuses applications réelles ont à résoudre des séries de problèmes relativement peu différents les uns des autres, par exemple lorsque des événements imprévisibles viennent régulièrement altérer les données ou les objectifs. Dans ces conditions,

il est intéressant de bénéficier des informations acquises lors des recherches antérieures si celles-ci restent valides pour le problème suivant. La justification associée aux nogoods assure que tant que ces contraintes sont présentes dans le problème à traiter, l'affectation du nogood est globalement incohérente. Elle permettra donc aux nogoods de survivre à une relaxation du problème [SV93, SV94a].

Le point supplémentaire apporté par les algorithmes de mémorisation consiste à conserver ces nogoods et à les réutiliser par la suite comme des contraintes.

La grande difficulté provient du nombre de nogoods qu'une recherche est susceptible de découvrir. En effet, le nombre de nogoods induits par un CSP augmente, en pire cas, de façon exponentielle avec la taille de ce dernier. Il est alors impératif de sélectionner judicieusement les nogoods que l'on souhaite mémoriser.

La stratégie adoptée par le plus simple des algorithmes de mémorisation, le "**Nogood-recording**" [SV94a], consiste à mémoriser les nogoods en fonction du cardinal de leur affectation. On considère qu'un nogood ayant une affectation de petite taille est plus susceptible d'être réutilisé puisqu'il concerne un plus grand nombre d'affectations.

Nous donnons ici une trame d'algorithme de backtrack amélioré par des mémorisations : **nr**(A) tente d'étendre l'affectation A et renvoie un nogood justifiant l'éventuel échec (soit un nogood inclus dans A) ; **nr-Evalue**$(A \cup \{x = i\})$ teste la cohérence globale de l'extension de A avec $x = i$ et, en cas d'incohérence, renvoie un nogood résultant de l'analyse de l'échec. Pour homogénéiser les notations, nous utilisons le nogood nul $n_\emptyset = (\emptyset, \emptyset)$ n'induisant aucune interdiction.

nr(A)

- Si A est complète alors A est une solution
- Sinon
  - Soit $x \notin \tilde{A}$, $N = \emptyset$
  - Pour chaque valeur i de x,
    - $n \leftarrow$**nr-Evalue**$(A \cup \{x = i\})$
    - Si $n = n_\emptyset$ alors $n \leftarrow$**nr**$((A \cup \{x = i\}))$
    - (Si $x^i \notin n$ alors renvoie n : backjumping)
    - $N \leftarrow N \cup \{n\}$
  - Mémorise et renvoie l'union des nogoods de N

nr-Evalue(A)

- Si **bt-Evalue**(A)= $FAUX$ alors
  Soit c une contrainte violée et A_c le n-uplet interdit,
  renvoie $(A_c, \{c\})$
- Sinon, S'il existe un nogood n violé par A alors
  renvoie n
- Sinon renvoie $n_\emptyset$

Exemple :

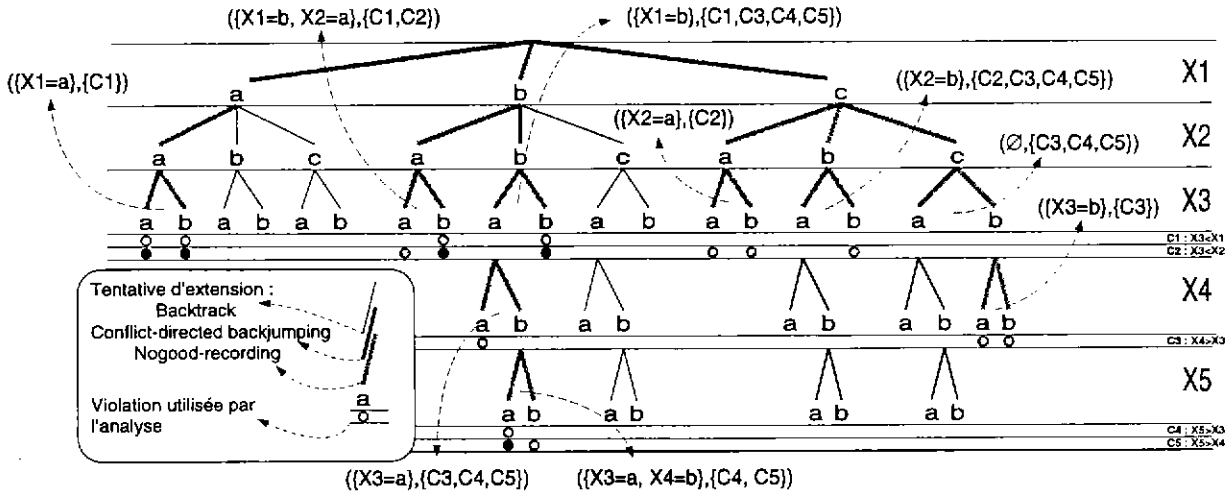

FIG. 2.5 – *Nogood-recording.*

La figure 2.5 donne le fonctionnement du Nogood-recording sur l'exemple, avec la

mémorisation de tous les nogoods construits par union.

> *Nous partons de l'exploration de la branche* $\{X_1 = b, X_2 = b, X_3 = a\}$. *Son extension impossible avec* $X_4 = a$ *donne lieu à un nogood* $n_{4a} = (\{X_3 = a, X_4 = a\}, \{C_3\})$ *qui ne sera pas mémorisé car il est redondant avec la définition de la contrainte* C_3 *(cf. remarque 2.3). Il est provisoirement conservé en vue d'une union lorsque toutes les valeurs de* X_4 *auront été explorées.*
>
> *L'extension avec* $X_4 = b$ *est a priori cohérente et est poursuivie sur* X_5. *Les deux valeurs de* X_5 *échouent et donnent lieu à deux nogoods (non mémorisés)* $n_{5a} = (\{X_3 = a, X_5 = a\}, \{C_4\})$ *et* $n_{5b} = (\{X_4 = b, X_5 = b\}, \{C_5\})$. *Toutes les valeurs de* X_5 *sont explorées. L'union de* n_{5a} *et* n_{5b} *donne le nogood* $n_{4b} = (\{X_3 = a, X_4 = b\}, \{C_4, C_5\})$ *justifiant le rejet de l'affectation courante* $\{X_1 = b, X_2 = b, X_3 = a, X_4 = b\}$. *Ce nogood plus pertinent (bien qu'inutile en l'occurrence) peut être mémorisé. Un retour-arrière sur* X_4 *est opéré.*
>
> *De même, toutes les valeurs de* X_4 *étant explorées, l'union des nogoods* n_{4a} *et* n_{4b} *qui justifient le rejet des valeurs est effectuée. Le résultat,* $(\{X_3 = a\}, \{C_4, C_5\})$ *est mémorisé et considéré comme une nouvelle contrainte. L'affectation de* X_3 *à* a *conduira désormais à la prédiction d'une incohérence globale: bien que localement cohérentes, les branches* $\{X_1 = c, X_2 = b, X_3 = a\}$ *et* $\{X_1 = c, X_2 = c, X_3 = a\}$ *ne seront pas développées.*

Les nogoods construits sont assimilables à des contraintes interdisant un unique n-uplet. En pratique, les algorithmes de mémorisation créent de nouvelles contraintes en cours de recherche et augmentent l'ensemble des contraintes du problème (ou l'ensemble des n-uplets interdits par une contrainte déjà existante).

Remarque 2.3 *Les nogoods produits par* **nr-Evalue** *ne méritent pas d'être mémorisés puisqu'ils sont redondants avec la définition du problème.*

La propriété d'union est à ce titre essentiel pour synthétiser les informations obtenues par l'exploration (donc non triviales).

Le Backtrack Dynamique

Le "**Backtrack Dynamique**" [Gin93, GM94] propose une option plus originale que le Nogood Recording: tout nogood est mémorisé, mais il est détruit dès que l'affectation courante est revenue sur deux valeurs mises en cause dans l'affectation du nogood.

Cette stratégie spécifique (quoique compatible avec le Nogood Recording habituel) offre des propriétés particulières qui font l'attrait du Backtrack Dynamique. Elle permet

de s'affranchir partiellement de la structure de recherche arborescente. En effet, dans le cas général, il est impératif d'effectuer les retours-arrière dans l'ordre inverse des affectations ; le backjumping est aussi astreint à cette règle puisqu'il détruit toutes les affectations jusqu'au point de retour arrière qu'il choisit. Or, le backtrack dynamique peut se dispenser de désaffecter les variables entre la dernière affectation et le point de backtrack qu'il choisit. Du fait de cette propriété, il n'est pas non plus tenu de revenir sur la dernière affectation impliquée dans l'échec de l'exploration mais peut sélectionner librement l'affectation conflictuelle à modifier. La seule restriction, importante tout de même, réside dans le respect d'un ordre partiel entre les variables qui impose une priorité dans les choix des points de backtrack. Cet ordre partiel repose sur les nogoods produits en cours de recherche et assure la terminaison du processus.

Plus formellement, à chaque valeur i de chaque variable x est associé un nogood N_{x^i}, initialisé à $n_\emptyset$, interdisant éventuellement l'extension d'une affectation A avec la valeur $x = i$ (ce nogood est une "mémoire" des anciennes tentatives infructueuses d'extension avec $x = i$).

Ces nogoods N_{x^i} impliquent des conditions de précédence $y < x$ entre chaque variable y de son affectation et la variable x à laquelle est associé le nogood. L'ensemble de toutes les conditions induites par les nogoods constitue, en l'absence de cycles, une relation d'ordre partiel sur l'ensemble des variables.

L'exploration utilise plusieurs fonctions :

- la procédure principale **dbt**(A) tente de développer une affectation partielle A ;
- en cas d'échec, cette-dernière fait appel à la procédure de retour arrière **dbt-Retour**(A, n) avec un nogood n justifiant l'échec de A ;
- la fonction **met-à-jour** met à jour les nogoods à l'occasion des retours. C'est la seule fonction qui modifie les nogoods.

L'algorithme est donné ci dessous :

dbt(A)

1. Si $\tilde{A} = \mathbb{V}$
   Si A est une solution FIN
   Sinon, soit n le nogood associé à une contrainte violée,**dbt-Retour**(A, n)
2. Sinon
   (a) Soit $x \in (\mathbb{V} - \tilde{A})$
   (b) S'il existe x^i tel que $N_{x^i} = n_\emptyset$
   dbt$(A \cup \{x = i\})$
   (c) Sinon, soit $n = \bigvee_i N_{x^i}$
   dbt-Retour(A, n)

dbt-Retour(A, n)

1. Si l'affectation de n est vide FIN
2. Sinon, soit $y^j \in \tilde{A}$ telle que aucune variable de l'affectation de n ne soit supérieure à y.
   (a) **met-à-jour**(A, n, y^j)
   (b) $A \leftarrow A - \{y = j\}$
   (c) **dbt**(A)

met-à-jour(A, n, y^j)

1. $\forall N_{z^k}$ dont l'affectation contient y^j, $N_{z^k} \leftarrow n_\emptyset$
2. $N_{y^j} \leftarrow n$

met-à-jour, appelée lors de chaque retour-arrière, assure que les nogoods N_{x^i} sont détruits dès qu'une variable de leur affectation change de valeur, c'est à dire dès que leur affectation n'est plus incluse dans $A \cup x^i$. Un nogood N_{x^i} s'applique donc toujours à l'extension de l'affectation courante avec la valeur considérée. Les propriétés des nogoods nous assurent que toutes les manipulations effectuées par cet algorithme produisent des nogoods valides. En particulier, le nogood n utilisé lors de l'appel de **dbt-Retour** constitue toujours une interdiction valide justifiant le rejet de l'affectation courante.

La validité de l'algorithme est donc acquise : il ne termine que par la mise en évidence d'une solution ou d'un nogood qui constate l'incohérence globale de l'affectation vide.

Par ailleurs, le respect de l'ordre partiel dans la sélection du point de backtrack (**dbt-Retour** ligne 2) nous assure que les conditions de précédence que le nogood produit par

met-à-jour ne produisent pas de cycles. Nous manipulons donc toujours un ordre partiel valide.

Terminaison :

La question réside dans la terminaison de cet algorithme qui n'hérite plus de l'exhaustivité des parcours arborescents. Dans le cas habituel, les rejets de valeurs produits par les retours-arrière descendent progressivement l'arbre de recherche jusqu'à la racine. Les branches coupées concernent progressivement des variables de plus en plus proches de la racine, donc plus petites selon l'ordre total imposé par la recherche, jusqu'à rejeter toutes les valeurs de la première variable. La preuve de la terminaison du Backtrack Dynamique s'appuie intuitivement sur le fait que les interdictions vont progressivement rejeter tout l'arbre de recherche en descendant, non plus vers la racine, mais vers les variables les plus petites selon l'ordre partiel.

Le test dans **dbt** ligne 2.b assure que le nogood nul est associé à toutes les valeurs de l'affectation courante. Lors d'un backtrack sur l'une d'entre elles, il est remplacé dans **met-à-jour** par un nogood interdisant toute nouvelle utilisation de cette valeur (jusqu'à la destruction du nogood). Un retour arrière sur une variable y provoque en quelque sorte une augmentation de l'interdiction de cette-dernière. Par contre, d'autres nogoods sont détruits par **met-à-jour**, mais ceux-ci concernent nécessairement des variables supérieures à y puisque y fait partie de la justification des nogoods. Tôt ou tard, mais assurément, on en viendra à interdire les variables minimum de l'ordre partiel avec une justification vide.

Plus formellement, on associe à chaque variable x_i le nombre n_i de ses valeurs qui sont interdites. Et on s'intéresse au nombre lexicographique obtenu en classant les variables selon un ordre total $>_O$: $N_O = (n_{i_1}, ..., n_{i_n})$, où $x_{i_k} >_O x_{i_{k'}} \iff k > k'$. Il faut alors constater que le minimum des nombres N_O dont l'ordre est compatible avec l'ordre partiel courant augmente strictement lors de chaque appel à **met-à-jour**. En effet, un n_y augmente strictement alors que ne diminuent que des n_z nécessairement supérieurs après n_y.

Exemple :

Il est difficile de présenter l'arbre de recherche produit par une recherche puisque la recherche n'est pas réellement arborescente. Pour illustrer cet algorithme, nous nous contentons de suivre les premier pas de la recherche obtenue en affectant les variables dans l'ordre naturel (sauf interdiction due à un nogood). Les retours arrière sont effectués

variable 1	2	3	4	5	n	nogood ajouté	ordre partiel	nogoods retirés
a	.	.	.	.				
a	a	.	.	.				
a	a	a	.	.				
a	a	a	a	.				
a	a	a	a	a	$\{X_1^a, X_3^a\}$	$\rightarrow N_{X_1^a}$	$X_1 > X_3$	
.	a	a	a	a				
b	a	a	a	a	$\{X_2^a, X_3^a\}$	$\rightarrow N_{X_2^a}$	$X_2 > X_3$	
b	.	a	a	a				
b	b	a	a	a	$\{X_3^a, X_4^a\}$	$\rightarrow N_{X_3^a}$	$X_3 > X_4$	$N_{X_1^a}, N_{X_2^a}$
b	b	.	a	a				
b	b	b	a	a	$\{X_1^b, X_3^b\}$	$\rightarrow N_{X_1^b}$		
.	b	b	a	a				
a	b	b	a	a	$\{X_1^a, X_3^b\}$	$\rightarrow N_{X_1^a}$		
.	b	b	a	a				
c	b	b	a	a	$\{X_2^b, X_3^b\}$	$\rightarrow N_{X_2^b}$		
c	.	b	a	a				
c	a	b	a	a	$\{X_2^a, X_3^b\}$	$\rightarrow N_{X_2^a}$		
c	.	b	a	a				
c	c	b	a	a	$\{X_3^b, X_4^a\}$	$\rightarrow N_{X_3^b}$		$N_{X_1^a}, N_{X_2^a}$
c	c	.	a	a	$\bigvee(N_{X_3^a}, N_{X_3^b}) = \{X_4^a\}$	$\rightarrow N_{X_4^a}$		$N_{X_3^a}, N_{X_3^b}$
c	c	.	.	a				
c	c	a	.	a				
c	c	a	b	a	*etc.*			

TAB. 2.1 – *Backtrack dynamique*

avec une priorité inverse mais toujours parmi les variables concernées par l'analyse de l'échec. Cette-dernière sélectionne les contraintes qui justifient un échec en fonction de cette même stratégie. Notre objectif est de mettre en évidence la souplesse qu'apporte cet algorithme.

Nous donnons dans le tableau 2.1 l'évolution de l'affectation courante, les raisons de son éventuel rejet, et les modifications entraînées par l'appel de **met-à-jour**.

> *Au moment où nous arrêtons la recherche sur notre exemple, l'ordre partiel donne* $X_4 < X_3 <$ (X_1 *et* X_2). *Les nogoods interdisent progressivement les valeurs des variables les plus petites selon l'ordre partiel. Déjà, le nogood placé sur* X_4^a *est définitif car son affectation est réduite à une seule valeur.*

On constate sur ce petit exemple la grande liberté dont on dispose pour orienter le backtrack et la souplesse de ce-dernier qui ne remet pas en cause les affectations qui ne sont pas sélectionnées comme responsables. Une fois la contrainte responsable choisie, et si l'on se cantonne aux variables mises en cause, l'ordre partiel peut s'avérer restrictif. En effet, excepté pour les trois premiers retours-arrière, le choix de la variable à modifier parmi la paire de variables impliquées est imposé par l'ordre partiel.

2.1.6 Heuristiques

Le schéma d'une recherche arborescente en profondeur ne laisse que deux degrés de liberté : celui de la variable, parmi les variables libres, avec laquelle une extension sera tentée puis celui de la valeur d'affectation parmi les valeurs ne conduisant pas à une incohérence globale. Cette liberté, bien que réduite, permet la mise en œuvre d'heuristiques sur l'ordre d'affectation des variables et des valeurs. Cet ordre peut être statique ie. déterminé une fois pour toutes avant le début de la recherche, ou dynamique, soit calculé avant toute nouvelle instanciation.

En ce qui concerne l'ordre des variables, la bonne stratégie consiste à détecter au plus tôt les incohérences globales, donc paradoxalement à rechercher les échecs. Ceci se comprend intuitivement et correspond au principe des prévisions : si l'affectation d'une variable doit s'avérer impossible, autant en tenir compte immédiatement. Les heuristiques à caractère général s'intéresseront, pour sélectionner une variable, au nombre de contraintes auxquelles elle participe, à la dureté de ces contraintes, à la taille de son domaine (nombre d'essais nécessaires pour prouver l'échec), etc. [MJPL92, WF95, HE80].

Il est possible de pousser plus loin cette stratégie en recherchant les variables critiques, c'est-à-dire celles ayant la plus grande influence sur la nature des solutions. Dans cette voie, les méthodes "**coupe-cycle**" [Dec90] mettent en évidence des sous-ensembles de variables qui sont critiques dans la mesure où, lorsque l'un d'entre eux est affecté, le sous-problème consistant à trouver une affectation pour les autres variables est polynômial.

L'ordre sur les valeurs n'a de sens que lorsque l'on ne s'intéresse qu'à la première solution trouvée (ou les premières). Les heuristiques consistent alors à choisir en premier lieu les valeurs ayant le plus de chances d'appartenir à une solution [FD94].

Ces heuristiques améliorent sensiblement les performances des algorithmes. Pour résoudre les problèmes les plus difficiles, elles s'avèrent cependant insuffisantes. En effet, on reste tenu de détruire les affectations dans l'ordre où elles ont été construites et de ne les détruire qu'après une exploration complète de toutes les possibilités d'extension ; la première variable affectée est la dernière à être désaffectée et elle n'est modifiée qu'après l'exploration complète du sous-problèmes constitué par l'affectation des autres variables. Lors de la résolution d'un problème de grande taille, les premiers choix doivent être considérés comme définitifs car la recherche n'aurra pas le temps de revenir dessus.

Le Backtrack Dynamique [GM94, McA93], algorithme plus récent et plus original, à été développé afin d'assouplir la recherche arborescente sans perdre la complétude. Il permet, en mémorisant à bon escient le travail effectué, de n'imposer qu'un ordre partiel sur les variables affectées. Un retour-arrière ne se fait plus nécessairement sur la dernière variable affectée, mais sur l'une des dernières variables d'un ordre partiel. Le caractère exhaustif de la recherche est assuré par la combinaison des mémorisations et de l'ordre partiel. Ceci permet de mettre en œuvre des heuristiques constructives et destructives, moins dépendantes de la structure arborescente de la recherche.

2.1.7 Résumé

Le schéma de base est celui d'une exploration arborescente par essai-erreur.

Il peut être amélioré par des explorations locales prévisionnelles. Ces techniques peuvent être très coûteuses car elles augmentent le travail d'évaluation de chaque nœud. En contrepartie, l'exploration partielle en largeur permet d'éviter les parcours inutiles d'un ordre d'affectation peu judicieux.

Les techniques d'analyse des échecs demandent moins de travail, mais sont souvent moins efficaces dans leur réduction du parcours arborescent. Ce sont en quelque sorte les versions paresseuses des techniques de prévision : elles attendent que l'incohérence locale soit atteinte pour déterminer depuis quand on pouvait la prévoir.

Enfin, lorsqu'une incohérence est prévue ou constatée, il est possible d'en mémoriser les causes sous forme d'une nouvelle contrainte. Ces méthodes sont particulièrement avantageuses pour le traitement de problèmes dynamiques puisqu'elles permettent de conserver d'une résolution à l'autre le bénefice des déductions.

Toutes ces techniques peuvent être hybridées à loisir et bénéficier d'heuristiques de choix des variables et des valeurs. Cependant, certaines méthodes peuvent s'avérer antinomiques. On retiendra premièrement que le besoin d'analyse des échecs est en partie devancé par les prévisions (on trouvera une analyse comparative intéressante dans [KB95]). Par ailleurs, mémorisation et prévision sont en partie redondantes (l'une se souvient de ce que l'autre prévoit). Enfin, l'efficacité des mémorisations est réduite lorsque l'ordre de sélection des variables n'est pas statique : dans ce cas, les violations peuvent se produire avant que les nogoods qui les prévoyaient ne soient violés.

2.2 CSP valués

2.2.1 Modèle CSP valués

Le cadre des CSP valués (VCSP) défini dans [SFV95] s'attache non plus à satisfaire toutes les contraintes, mais à minimiser une fonction liée à la satisfaction des contraintes.

Plus formellement, il faut se doter d'un CSP classique $(\mathbb{V}, \mathbb{C})$ et d'une structure de valuation $\mathbb{S}$ composée :

- d'un ensemble de valuations E (permettant de valuer contraintes et affectations), muni d'un ordre total $\preceq$ (permettant de comparer les valuations) et incluant un élément minimum $\bot$ (traduisant la satisfaction d'une contrainte ou la cohérence d'une affectation) et un élément maximum $\top$ (traduisant une insatisfaction inacceptable) ;
- d'une fonction de valuation ϕ, associant à chaque contrainte c de $\mathbb{C}$ un élément $\phi(c)$ de E traduisant l'importance de la satisfaction de c ;

- d'un opérateur d'agrégation des valuations $\oplus$, commutatif, associatif, monotone relativement à $\preceq$, tel que $\bot$ soit un élément neutre et *top* soit absorbant.

Nous nous plaçons toujours dans le cadre d'une minimisation où l'objectif est de trouver une solution aussi proche de $\bot$ que possible.

L'ordre sur les valuations n'est pas nécessairement total, un ordre partiel suffit ; nous aurons besoin de cette liberté lors de l'application concrète de cette modélisation. Cependant, dans cet état de l'art, nous nous placerons dans le cas d'un ordre total pour ne pas nous détourner de l'essentiel et clarifier nos propos. Les définitions ou propriétés pouvant être altérées dans le cas d'un ordre partiel seront marquées par le signe "†" et reprises lors de l'application présentée dans le cinquième chapitre.

D'autre part, il est souvent pratique d'associer à une seule contrainte plusieurs valuations selon le n-uplet interdit. Notre modélisation oblige à produire autant de contraintes que de valuations possibles, même si les n-uplets concernent les mêmes variables. On peut néanmoins préférer regrouper ces multiples contraintes en une seule. Dans ce cas, un paramètre implicite de la fonction ϕ est le n-uplet qui viole la contrainte, du moins est-ce ainsi qu'il faudra comprendre la notation $\phi(c)$. Cependant, nos développements nous amèneront à considérer la valuation des contraintes indépendamment de leur violation. Alors, une remarque, distinguée par le signe "‡", viendra proposer les adaptations nécessaires.

Pour le reste, on peut remarquer que les éléments $\bot$ et $\top$ n'ont un intérêt que descriptif. Par ailleurs, si la monotonie est une propriété indispensable pour le bon fonctionnement des algorithmes, la commutativité et l'associativité ne sont imposées que pour assurer l'unicité des résultats des agrégations (indépendemment de l'ordre des calculs).

Nous appelons alors *valuation (locale)* d'une affectation partielle A (notée $v(A)$) l'agrégation des pénalités des contraintes affectées par A ($\bar{c} \subset \tilde{A}$) et insatisfaites : $v(A) = \bigoplus_{c \in \mathbb{C}/A \models \neg c} \phi(c)$. Nous utiliserons la notation $v_{\mathcal{C}}(A)$ pour projetter la valuation sur un sous ensemble de contrainte $\mathcal{C}$: $v_{\mathcal{C}}(A) = \bigoplus_{c \in \mathcal{C}/A \models \neg c} \phi(c)$

Une solution du VCSP est une affectation complète de valuation strictement inférieure à $\top$. Il s'agit d'une *solution optimale* s'il n'existe pas de solution de valuation plus faible.

Nous appelons enfin *valuation globale* d'une affectation A le minimum des valuations des extensions complètes de A.

Plusieurs catégories de VCSP s'inscrivent dans ce cadre [FW92]. Nous en citons

quelques-unes :

- les VCSP *possibilistes*, $(E, \preceq, \bot, \top, \oplus) = ([0,1], \leq, 0, 1, \max)$;
- les VCSP *additifs*, $(E, \prec, \bot, \top, \oplus) = (\mathbb{N} \cup \{+\infty\}, \leq, 0, +\infty, +)$;
- les CSP *classiques*, $(E, \preceq, \bot, \top, \oplus) = (\{0,1\}, \leq, 0, 1, \vee)$. 0 pour satisfaction ou cohérence, 1 sinon; l'agrégation est un "ou" logique car une affectation est incohérente si l'une des contraintes est insatisfaite.

2.2.2 Algorithmes

La question n'est plus de trouver une solution, ce qui s'avère le plus souvent trivial, mais d'en trouver une de "bonne" qualité. Si l'on souhaite toujours obtenir une solution optimale, une solution proche de l'optimum reste en général satisfaisante. Cet état de fait a pour conséquence la mise en concurrence permanente des deux grandes classes d'algorithmes, selon qu'il effectuent une recherche locale incomplète ou une recherche complète.

Les méthodes approchées sont guidées par des heuristiques et font évoluer une solution courante de voisinage en voisinage. Elles sont de ce fait extrêmement sensibles à la richesse des voisinages que l'on est en mesure de produire. Les heuristiques sont essentiellement de type gradient [SK93, GW93, WF95] et sont améliorées par des méthodes d'extraction des *minima* locaux [Mor93, GL93] souvent stochastiques [KGV83, JKS95]. Nous ne nous étendrons pas sur les multiples formes que peuvent prendre les recherches locales ou, plus généralement, les recherches incomplètes. Retenons simplement que l'absence de toute contrainte d'optimalité et de preuve permet une grande liberté de l'exploration et la mise en œuvre d'heuristiques assurant une convergence rapide vers les solutions de "bonne" qualité. L'inconvénient est qu'il est en général impossible de juger la qualité des solutions obtenues.

A l'inverse, les méthodes exactes fournissent des solutions optimales mais se prêtent assez mal à l'utilisation d'heuristiques et peuvent facilement nécessiter des temps de résolution prohibitifs. En effet, la complétude exige d'être en mesure de parcourir exhaustivement tout l'espace de recherche et le seul principe qui permette d'aborder raisonnablement l'ensemble des VCSP sans restriction est celui du "**branch & bound**". Qui plus est, afin d'éviter, pour des raisons pratiques, l'utilisation d'un espace mémoire exponentiel, cette

méthode est utilisée en "profondeur d'abord". Cela consiste à développer un arbre et, à chaque nœud, à évaluer un minorant de toutes les feuilles filles. Lorsque ce minorant excède une borne maximale, généralement la valeur de la dernière solution trouvée, le sous-arbre descendant du nœud est abandonné.

Ainsi utilisé, le branch & bound est l'équivalent d'un backtrack où la fonction d'évaluation de la cohérence locale de chaque affectation explorée est remplacée par l'évaluation de la valuation locale. Il évite le parcours de toutes les feuilles possibles en utilisant le lemme suivant :

Lemme 2.11 *La valuation globale d'une affectation est supérieure ou égale à sa valuation locale.*

Soit α une borne que l'on peut fixer *a priori* à la valuation maximale, l'algorithme du Branch and Bound prend la forme :

BT(A)

- Si A est complète alors A est une solution (de valuation inférieure à α)
  $\alpha \leftarrow v(A)$
- Sinon
  - Soit $x \notin \tilde{A}$
  - Pour chaque valeur i de x,
    Si **BT-Evalue**$(A \cup \{x = i\}) \prec \alpha$ † alors **BT**$(A \cup \{x = i\})$

La fonction **BT-Evalue**$(A \cup \{x = i\})$ consiste ici à calculer la valuation locale de $A \cup \{x = i\}$.

Notez que les noms de procédures en majuscules font référence aux algorithmes du cadre valué, alors que les noms en minuscules sont réservés au cadre classique.

Cet algorithme explore un certain nombre de solutions sous-optimales. A chaque fois, la borne supérieure est diminuée, interdisant toute nouvelle solution d'une valuation supérieure ou égale à celle de la dernière solution obtenue. La conséquence directe est qu'une unique solution optimale est exhibée, même s'il en existe plusieurs de valuations équivalentes.

†. Avec un ordre partiel les deux formulations min $\prec$ max et $\neg$ min $\succeq$ max ne sont pas équivalentes. Tel qu'il est formulé, le test conduit au rejet de toute solution de valuation plus mauvaise que la borne maximum ou incomparable.

Exemple :

Si l'on considère notre exemple 2.1 comme un VCSP additif où toutes les contraintes ont une valuation unitaire, l'arbre développé par cet algorithme est donné par la figure 2.6.

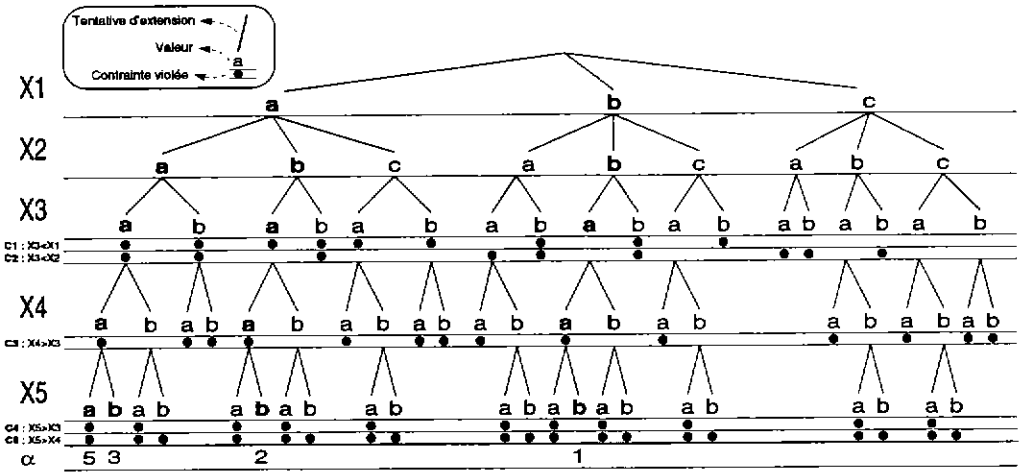

FIG. 2.6 – *Le Branch and Bound*

CSP valués et idempotence de l'agrégation

Le problème de décision "Existe-t-il une solution de valuation strictement inférieure à une borne α" est très proche du problème de satisfaction et les algorithme élémentaires de résolution sont quasiment identiques.

Remarque 2.4 *Lorsque l'opérateur d'agrégation est idempotent ($v \oplus v = v$), le Backtrack et le Branch and Bound en profondeur d'abord sont parfaitement équivalents.*

En effet, un opérateur d'agrégation idempotent correspond nécessairement à un opérateur "maximum" : $v \oplus v' = \max(v, v')$ [SFV95]. Le problème de décision est alors équivalent au problème de satisfaction réduit aux contraintes de valuation supérieure ou égale à α. Dès lors qu'il s'agit de trouver une solution de valuation inférieure à α, le respect de toutes ces contraintes est une condition nécessaire et suffisante. Le schéma d'optimisation proposé correspond alors à une série de résolutions de CSP classiques. Le premier CSP est le VCSP sans contrainte ; un nouveau CSP se déduit de la solution d'un précédent par l'ajout des contraintes ayant une valuation supérieure ou égale à la valuation de cette

dernière solution. Il faut par ailleurs remarquer que l'ajout de contraintes ne remet jamais en cause le travail fourni par une recherche arborescente : les échecs constatés ou prévus, ainsi que l'analyse qui en est faite et leur mémorisation restent valables car les contraintes qui en sont à l'origine sont toujours présentes.

Toutes les techniques de résolution appliquées aux CSP peuvent par conséquent être reproduites pour l'optimisation de VCSP dont l'opérateur est idempotent.

Par contre, lorsque l'opérateur n'est pas idempotent, il est en général impossible d'identifier un sous-ensemble de contraintes tel que la satisfaction du sous-ensemble soit suffisante et la satisfaction de chaque contrainte nécessaire afin d'obtenir une solution de meilleure valuation. En effet, toute violation de contrainte peut potentiellement participer au rejet d'une solution par agrégation avec d'autres violations. Par ailleurs, ces agrégations étant le plus souvent nécessaires pour conduire à un rejet, la satisfaction indépendante d'une contrainte n'est pas *a priori* nécessaire. Pour revenir à un problème de satisfaction, il conviendrait de considérer les sous-ensembles de contraintes qui ne doivent pas être toutes violées en même temps. Chacun d'entre eux induit un ensemble de n-uplets interdits (sous peine de violation de toutes les contraintes) et correspond donc à une contrainte dure de forte arité. Le nombre de ces contraintes et leur arité ont pour conséquence d'accroître la complexité des problèmes de satisfaction qui en résultent.

Avec un opérateur non idempotent, non seulement l'extension des techniques utilisées pour les CSP classiques n'est pas immédiate, mais en outre les problèmes associés sont plus difficiles à résoudre [SFV95]. Cependant, nous nous intéresserons plus particulièrement à cette catégorie de problèmes qui constitue l'essence des VCSP et englobe la catégorie la plus immédiate, celle des VCSP additifs. Nous présentons dans la partie qui suit les quelques améliorations apportées au Branch & Bound dans le cas général des VCSP.

2.2.3 Prévision des échecs

Comme dans le cas des CSP classiques, il s'agit d'explorer les variables non instanciées afin de prédire les violations ; cependant, on ne cherchera plus à savoir si une extension est possible mais quelle valuation elle implique. Nous cherchons maintenant un minorant de la valuation globale d'une affectation aussi élevé que possible (soit aussi proche que possible de la valuation globale). La valuation locale constitue un minorant rudimentaire que l'on souhaite améliorer grâce à une exploration partielle.

Le Forward-checking valué

L'équivalent du Forward-checking [FW92] consistera à déterminer, pour chaque variable non instanciée, la valuation minimale que l'on obtiendrait en l'affectant. Si l'une de ces valuations excède la borne maximale, le backtrack peut être enclenché. La fonction **BT-Evalue** est remplacée par la fonction **FC-Evalue** suivante :

FC-Evalue(A)

- Pour chaque variable $x \notin \tilde{A}$
  $v_x \leftarrow \min_i(v(A \cup \{x = i\}))$ †
- renvoie $\max_{x \notin \tilde{A}}(v_x)$

On peut faire mieux si l'on ne s'intéresse plus à la valuation des extensions mais au supplément de valuation entraîné par l'extension.

Le Forward-checking valué étendu

Soit A une affectation partielle, son extension avec une valeur x^i causera la violation d'un ensemble supplémentaire de contraintes $\mathcal{C}_{x^i}$ pour une valuation $\delta_{x^i} = \bigoplus_{c \in \mathcal{C}_{x^i}} \phi(c)$, avec $v(A \cup \{x = i\}) = v(A) \oplus \delta_{x^i}$.

Soit $\delta_x = \min_i(\delta_{x^i})$, le Forward-checking simple se borne à constater que la valuation de l'extension de A avec x (donc la valuation globale de A) est supérieure ou égale à $v(A) \oplus \delta_x$. Mais il faut remarquer que pour deux variables non affectées x et y distinctes, et quelle que soit leur valeur d'affectation, les ensembles $\mathcal{C}_{x^i}$ et $\mathcal{C}_{y^j}$ sont disjoints. En effet, pour que les affectations séparées de x et y violent la même contrainte, il faudrait que cette dernière puisse être violée sans que toutes les variables qu'elle lie ne soient affectées. Dès lors, l'agrégation des valuations des contraintes de ces deux ensembles est égale à $\delta_{x^i} \oplus \delta_{y^j} \succeq \delta_x \oplus \delta_y$. Les violations supplémentaires causées par l'extension simultanée avec x et y nous assurent donc que $v(A \cup \{x^i, y^j\}) \succeq v(A) \oplus \delta_x \oplus \delta_y$.

†. La définition du minimum d'un ensemble de valuations avec un ordre partiel mérite d'être reconsidérée. Il s'agit intuitivement d'une valuation que l'on souhaite aussi élevée que possible et qui soit inférieure à toutes les valuations de l'ensemble (sans être nécessairement égale à l'une d'entre elles).

En généralisant à l'ensemble des variables non instanciées, on obtient une procédure plus performante [FW92], en remplaçant la fonction **FC-Evalue** par la forme étendue suivante:

EFC-Evalue(A)

- Pour chaque variable $x \notin \tilde{A}$
  $\delta_x \leftarrow \min_i(\delta_{x^i})$, où δ_{x^i} est la valuation des contraintes violées par l'affectation de x à la valeur i.
- renvoie $v(A) \oplus (\bigoplus_x \delta_x)$

Exemple :

Le fonctionnement de cet algorithme sur notre exemple est présenté sur la figure 2.7. L'exemple est cependant trop petit pour permettre d'illustrer la supériorité du Forward-checking étendu sur son homologue simple.

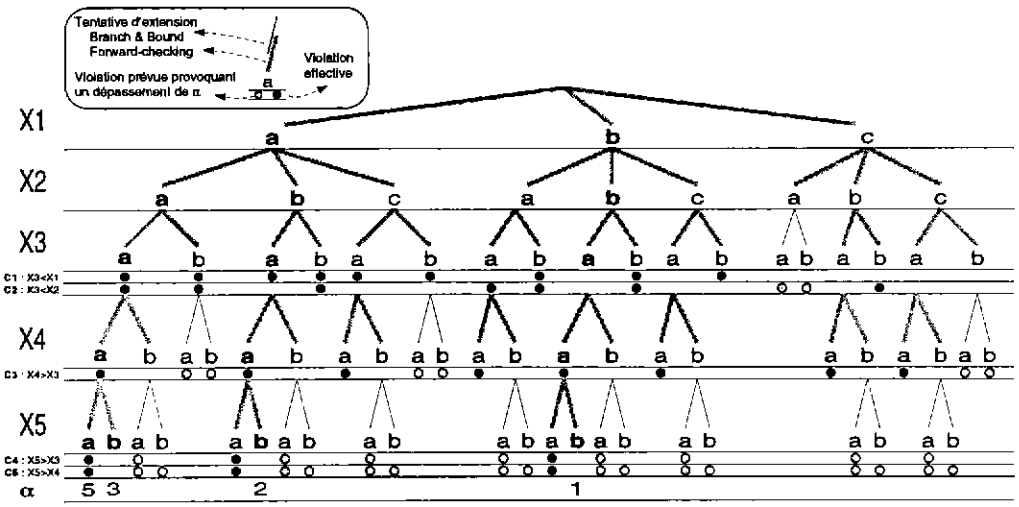

FIG. 2.7 – *Le Forward-checking valué.*

Suivons le début de la recherche.

La borne supérieure α est infinie, donc toutes les violations sont autorisées. La recherche s'enfonce donc jusqu'à la première feuille, obtenant une solution $\{X_1^a, X_2^a, X_3^a, X_4^a, X_5^a\}$ de valuation 5. α est fixée à 5, toutes les affectations de valuation supérieure seront rejetées, y compris l'affectation courante.

Retour-arrière sur X_5 et affectation à X_5^b. C'est une nouvelle solution de valuation 3, la borne supérieure est mise à jour. Toutes les valeurs de X_5 sont explorées.

Retour-arrière sur X_4, et examen de l'affectation $\{X_1^a, X_2^a, X_3^a, X_4^b\}$. Le Forward-checking examine la compatibilité des variables libres avec cette affectation. Plus précisément, il calcule la valuation impliquée par l'affectation de X_5 : elle serait de 3 pour une extension avec X_5^a et de 4 pour une extension avec X_5^b, donc quoi qu'il en soit au moins égal à 3. Cette valeur retournée par **FC-Evalue**$(\{X_1^a, X_2^a, X_3^a, X_4^b\})$ *conduit au rejet de l'affectation courante. Toutes les valeurs de X_4 sont explorées.*

Retour-arrière sur X_3 et examen de l'affectation $\{X_1^a, X_2^a, X_3^b\}$ par **FC-Evalue**. *Elle viole toujours deux contraintes. L'examen des extensions avec X_4 affiche une valuation minimale de 3. L'examen des extensions avec X_5 produirait exactement la même valuation.* **FC-Evalue** *renvoie donc la valuation 3 qui suffit à entraîner le rejet de l'affectation courante.*

Il est cependant possible de faire mieux en s'intéressant aux surcoûts. $\{X_1^a, X_2^a, X_3^b\}$ viole déjà des contraintes pour une valuation de 2. L'extension avec X_4 provoque un surcoût de 1 car la violation de C_3 est inévitable. De même, l'extension avec X_5 provoque un surcoût de 1 à cause de la violation nécessaire de C_4. **EFC-Evalue**$(\{X_1^a, X_2^a, X_3^b\})$ *renvoie la valuation $2+1+1$: la valuation globale de l'affectation est au moins égale à 4 car deux contraintes sont violées et deux autres le seront. Sur l'exemple, il n'est pas nécessaire de donner une estimation aussi élevée mais, pour de plus gros problèmes, la version étendue s'avère nettement plus performante.*

Vers l'arc-cohérence

Remarque 2.5 *Si l'opérateur n'est pas idempotent, on ne peut agréger des valuations que si ces valuations sont déduites des violations dans des ensembles de contraintes deux à deux disjoints.*

Prenons un petit exemple : vous possédez 300F sous plusieurs formes ; 100F en pièces, 100F en billets, 100F en Chèques de voyage. Il vous faut aller jusqu'à votre hôtel avec pour contraintes de conserver chaque pièce, chaque billet et chaque chèque dans vos poches sous peine d'une pénalité égale au montant perdu. Voyant l'ampleur de votre tâche, un oracle vient à votre secours et vous annonce : "En prenant le chemin de droite, vous perdrez au moins la moitié de vos pièces soit 50F, au moins la moitié de votre argent liquide soit 100F, et au moins la moitié de l'argent actuellement en votre possession soit 150F ; en

prenant le chemin de gauche, vous perdrez au moins la moitié de vos pièces soit 50F, au moins la moitié de vos billets soit 50F et au moins la moitié de vos chèques soit 50F.". A gauche, les valuations sont cumulables car elles portent sur des contraintes distinctes et l'on est certain de perdre au moins 150F. A droite, les intersections entre les différents ensembles interdisent de conclure à la perte de vos 300F.

La technique consiste donc à définir une partition de l'ensemble des contraintes et à identifier pour chaque sous-ensemble un minorant de la valuation des violations qu'une extension complète provoquera dans chacun d'entre eux. Le Forward-checking considère pour chaque variable non instanciée x, l'ensemble des contraintes qui lient x à l'affectation A considérée (cet ensemble se définit formellement comme $\{c/x \in \tilde{c} \subset \tilde{A} \cup \{x\}\}$).

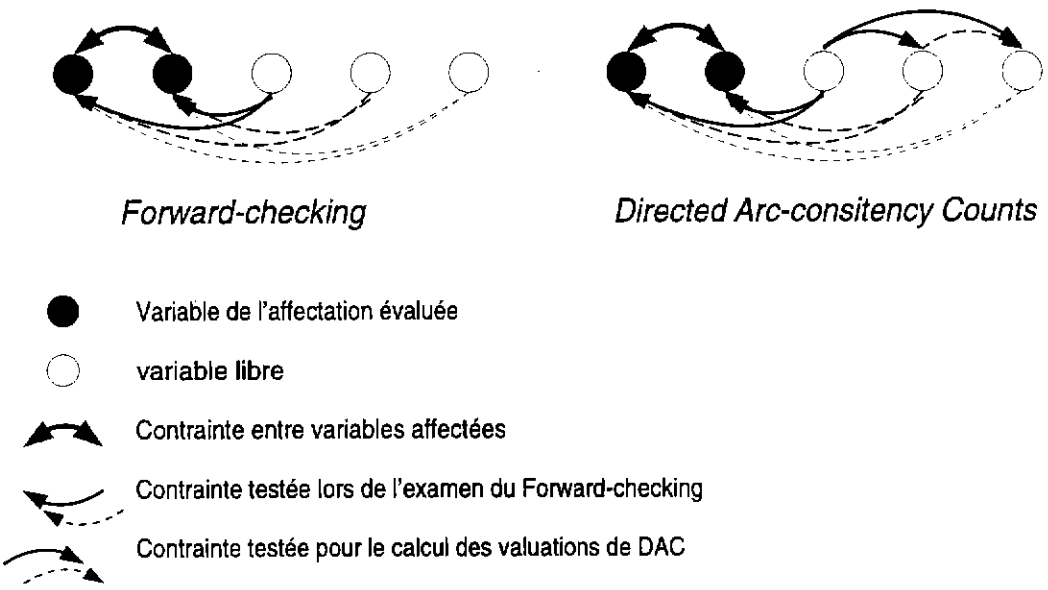

FIG. 2.8 – *Contraintes testées par les algorithmes de prévision.*

Dans cette optique, le plus puissant des algorithmes de prévision des échecs est certainement le "**Directed Arc-consistency Counts**" (DAC) [Wal96, LM96]. Il se rapproche de l'arc-cohérence dans la mesure où il examine si l'affectation d'une variable peut être suivie de l'extension avec une seconde et calcule l'équivalent d'une contrainte induite unaire (un "Count") dans le cas contraire. Plus précisément, il calcule le coût minimum de l'extension avec toute seconde variable, comme le ferait un Forward-checking. Cependant, toutes les paires de variables ne sont pas testées : un ordre total sur les variables est fixé et seules les paires (x, y) où x précède y sont examinées. On est alors certain que les violations prévues concernent des ensembles de contraintes disjoints (Figure 2.8). Les

valuations peuvent être agrégées entre elles, avec les contraintes violées par l'affectation et avec les violations considérées par le Forward-checking, sans risque de redondances.

Cet algorithme est proposé par Wallace comme prétraitement. Les valuations posées avant la recherche sur les valeurs ont pour but d'aider la recherche d'un Forward-checking.

La procédure d'examen correspondante, **DAC-Examen**, est décrite ci-dessous. Elle associe à chaque valeur x^i une valuation induite dac_{x^i}.

DAC-Examine

- Pour chaque variable x
  c_{x^i} =**DAC-Evalue**(x^i))

DAC-Evalue(x^i)

- Pour chaque variable y suivant x dans l'ordre total[4],
  $\delta_{x^i y} \leftarrow \min_j$(**BT-Evalue**$(x^i y^j)$)
- renvoie $\bigoplus_y \delta_{x^i y}$

DAC-Evalue(x^i) renvoie la valuation de ce que l'on peut considérer intuitivement comme une contrainte induite portant sur l'affectation x^i : si une affectation est étendue avec $x = i$, on est certain de produire des violations pour une valuation au moins égale à c_{x^i}. Ces valuations s'intègrent naturellement à l'évaluation du Forward-checking :

EFC+DAC-Evalue(A)

- Pour chaque variable $x \notin \tilde{A}$
  $\delta_x \leftarrow \min_i(\delta_{x^i} \oplus c_{x^i})$, où δ_{x^i} est la valuation des contraintes violées par l'affectation de x à la valeur i.
- renvoie $v(A) \oplus (\bigoplus_x \delta_x)$

Pour plus d'efficacité, il semble possible de passer du prétraitement à un maintien permanent des valuations pendant la recherche. Mais le caractère statique de cette méthode peut présenter un handicap sérieux.

Exemple :

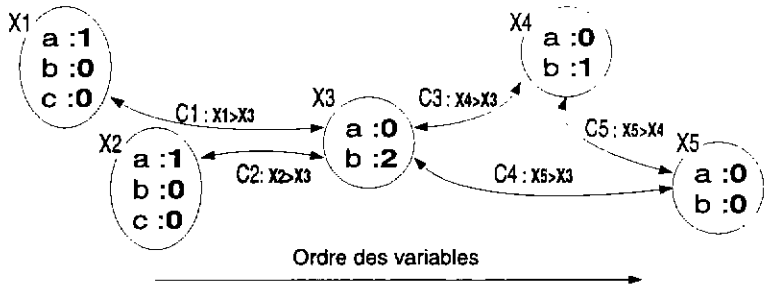

FIG. 2.9 – *DAC.*

Nous donnons pour notre petit exemple les valuations obtenues ainsi que l'arbre de recherche correspondant (figures 2.9 et 2.10).

> *Précisons la manière dont sont construites les valuations associées aux valeurs en suivant l'examen de* X_3.
>
> **DAC-Evalue**(X_3^a) *examine les extensions de* X_3^a *avec toute variable succédant* X_3 *(soit* X_4 *et* X_5*). Avec* X_4*, on peut espérer ne violer aucune contrainte, de même qu'avec* X_5*. La valuation associée à* X_3^a *est donc nulle.*
>
> *Par contre, lorsque l'on s'intéresse à* X_3^b*, il apparaît que son extension avec* X_4 *produit nécessairement des violations pour une valuation* 1*. Il en va de même pour les extensions avec* X_5 *(mais du fait de contraintes différentes). Une valuation* $1+1$ *est donc associée à* X_3^b*. Elle signifie que toute extension de cette valeur avec les variables suivantes entraîne un surcoût au moins égal à* 2.
>
> *Un exemple d'utilisation de ces valuations est donné par l'examen de* $\{X_1^a, X_2^c, X_3^a\}$ *:* **EFC+DAC-Evalue** *retourne une valuation* $1+1$ *constituée de la violation de* C_2 *par l'affectation explorée et de l'examen des extensions avec* X_4 *: on obtient un surcoût de* 1*, avec* X_4^a *à cause de* C_3 *et avec* X_4^b *à cause de la valuation qui lui est associée.*

Lors de la recherche, seule la valuation associée à X_4^b est utile. Elle permet plusieurs fois de stopper la recherche lors de l'examen des affectations contenant X_3^a.

Les Poupées Russes

Nous terminons cet état de l'art par un dernier algorithme plus récent, les "**Poupées Russes**" [VLS96]. La prévision des violations est ici plus poussée encore que celle effectuée par **DAC-Evalue**. En effet, l'estimation des violations à venir entre variables non affectées provient non pas d'un examen de paires de variables non affectées, mais d'une résolution complète du sous-problème constitué par l'ensemble des variables non affectées.

4. Il s'agit là de la seule différence avec **EFC-Evalue**.

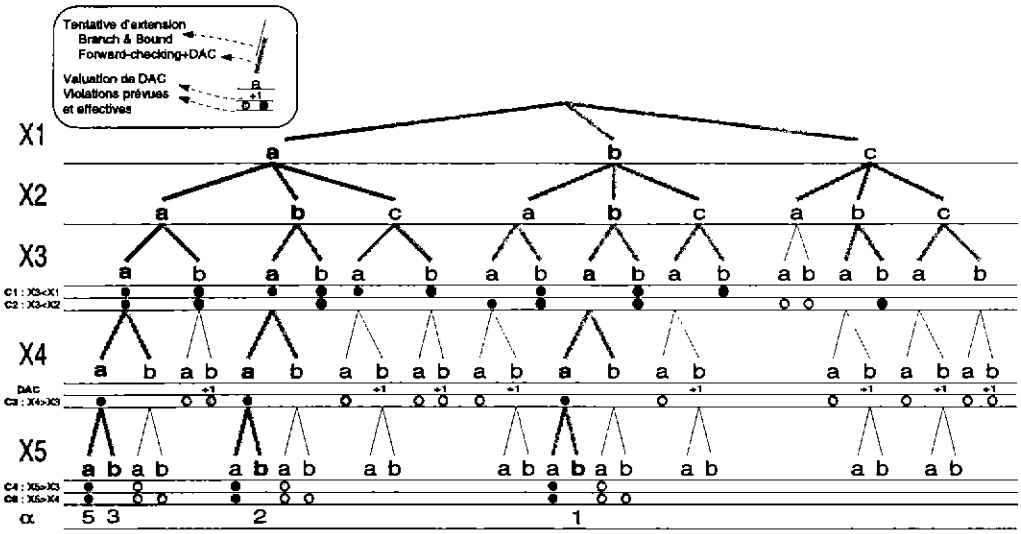

FIG. 2.10 – *DAC + Forward-checking.*

Il utilise un ordre statique et procède par résolutions successives des problèmes emboîtés constitués par la restriction du VCSP aux n dernières variables. Chaque résolution produit l'affectation et la valuation optimale pour les n dernières variables. Lors de la résolution des problèmes $n + k$, cette information peut être ajoutée au forward-checking pour donner une meilleure estimation des valuations globales (cf figure 2.11).

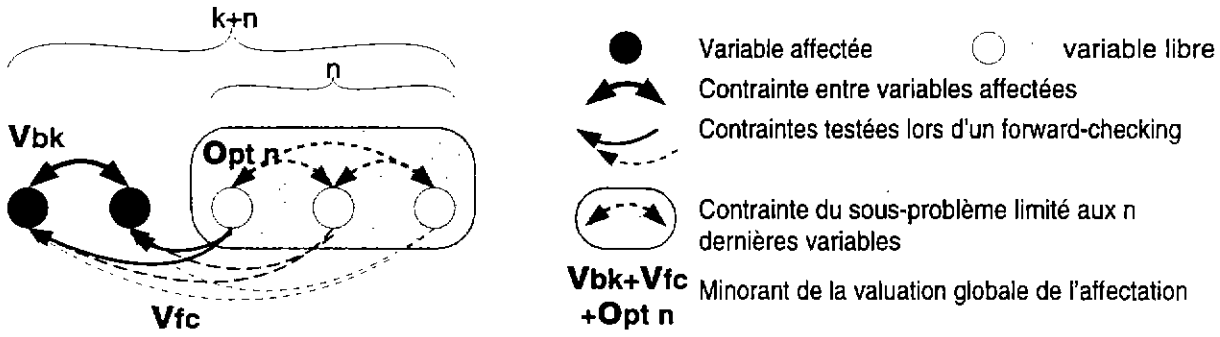

FIG. 2.11 – *Evaluation des valuations globales par les Poupées Russes*

L'algorithme a la forme suivante :

RDS[5]

- Soit n le nombre total de variables ordonnées de x_1 à x_n,
- Pour $k = n$ jusqu'à 1,
  - Soit $(\mathbb{V}_k, \mathbb{C}_k)$ le CSP restreint aux variables de $\mathbb{V}_k = \{x_k, ..., x_n\}$, Opt_k =**BT-RDS**$(\mathbb{V}_k, \mathbb{C}_k)$

(Opt_1 est la valuation optimale du problème)

La procédure **BT-RDS** est un branch and bound classique en profondeur d'abord qui affecte les variables selon l'ordre statique prédéfini. La seule particularité provient de sa fonction d'évaluation **RDS-Evalue** qui est un forward-checking amélioré par les valeurs Opt_k précédemment calculées :

RDS-Evalue(A)

- Soit x_i la dernière variable de $\tilde{A}$,
- Renvoie **EFC-Evalue**$(A) \oplus Opt_{i+1}$

Contrairement à ce que l'on pourrait penser, ni l'ordre statique, ni les résolutions redondantes des n sous-problèmes imbriqués ne constituent un handicap insurmontable. En effet, les Poupées Russes se distinguent nettement comme le meilleur algorithme sur les problèmes aléatoires dont les contraintes sont dures (peu de n-uplets autorisés), et ceux présentant une faible largeur de bande (étant donné un ordre total des variables, il s'agit de l'écart maximal entre deux variables connectées par une contrainte).

Exemple :

Nous présentons figure 2.12 uniquement la dernière résolution, après que tous les optimums intermédiaires aient été obtenus ($Opt_5 = 0$, $Opt_4 = 0$, $Opt_3 = 1$, $Opt_2 = 1$).

> *Remarquez comment, lors de l'examen négatif de l'affectation $\{X_1 = a, X_2 = c\}$, la valuation 1 prévue par le Forward-checking en cas d'extension avec X_3 est ajoutée à Opt_3, la valuation minimale prévue sur le sous-problème constitué par X_3, X_4 et X_5.*

5. "RDS" pour Russian Dolls Search.

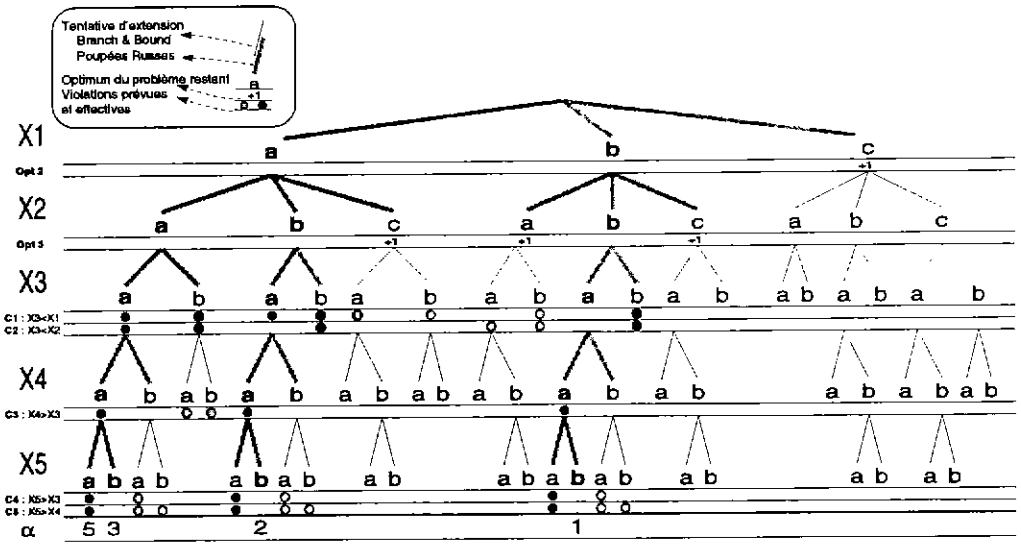

FIG. 2.12 – *Poupées Russes.*

2.2.4 Résumé

Les CSP valués constituent une extension extrêmement utile des CSP puisque de nombreux problèmes réels se posent en termes d'optimisation. Ce nouveau cadre conserve les même qualités que celui des CSP classiques (simplicité et expressivité) et hérite assez naturellement des algorithmes de résolution les plus simples.

A ce jour, le Forward-checking est le meilleur algorithme que les VCSP aient emprunté aux CSP. Associé à des heuristiques sur l'ordre des variables, son efficacité est difficilement égalée.

Ce formalisme est cependant très récent et n'a pas donné lieu à beaucoup de développements d'algorithmes de résolution complets. Parmi ceux-ci, seuls DAC et les Poupées Russes sont en mesure de surpasser le Forward-checking sur certaines catégories de problèmes.

Comparé aux CSP classiques, le faible nombre d'algorithmes dont on dispose s'explique aussi par les difficultés que posent les agrégations non idempotentes. Les quelques algorithmes proposés sont entravés par le risque de redondance dans les évaluations car ils évitent ce problème grâce à la rigidité de leur recherche (ordre statique sur les variables en particulier).

2.3 Conclusion

Lorsque l'on cherche à résoudre des CSP, valués ou non, la trame qui s'impose est celle de l'énumération arborescente par essais-erreur. Lorsque l'on développe une affectation, toute la question consiste à détecter les erreurs au plus tôt afin d'effectuer un minimum d'essais. Un bon compromis doit cependant être recherché entre le temps passé à essayer et celui passé à détecter.

La détection des échecs repose essentiellement sur deux principes : le premier est l'anticipation par de petites explorations locales autour de l'affectation en cours de construction. Le forward-checking ou l'arc-cohérence dans le cadre classique sont souvent donnés comme de bons compromis entre le temps de detection et la limitation des essais. Le second intervient *a posteriori* et consiste à analyser et mémoriser les causes des échecs en vue de prévisions ultérieures. Il n'est élaboré que pour les CSP classiques. Ce principe constitue plutôt une alternative du premier car leurs efficacités respectives sont difficilement compatibles.

On notera qu'en général les méthodes de résolution sont développées pour la résolution de CSP et VCSP aux contraintes binaires. Même si leur extension aux contraintes n-aires est possible, elle s'accompagne toujours d'une perte de performances.

Les CSP valués souffrent d'une certaine pauvreté dans le choix des méthodes de résolution complètes en raison de la nature plus complexe des échecs. Il ne s'agit plus, comme dans le cadre classique, de détecter indifféremment toute violation puisque toutes se valent et qu'une suffit, mais de détecter le cumul des valuations produites par des ensembles de violations. Cette difficulté théorique, liée à l'idempotence des opérateurs, est d'autant plus regrettable qu'elle conduit à une difficulté accrue des problèmes et qu'elle se pose pour la catégorie incontournable des VCSP additifs.

Chapitre 3

VCSP et contraintes induites

Les techniques de résolution complètes des VCSP ne vont pas au delà des quelques algorithmes présentés dans le chapitre précédent pour deux raisons liées à la propriété d'idempotence de l'opérateur d'agrégation.

Premièrement, le nombre éventuellement important de contraintes qu'il faut violer pour atteindre un rejet de l'affectation en cours d'exploration rend inopérante toute analyse de l'échec : il y a souvent trop de variables mises en cause. En d'autres termes, les contraintes induites par l'analyse des échecs ont une arité trop élevée pour que leur utilisation soit efficace à des fins de backjumping ou de mémorisation.

Deuxièmement, les risques de redondance interdisent en général l'utilisation de contraintes induites ou de nogoods. Du moins, ils imposent de prendre des précautions importantes qui se reflètent dans la rigidité extrême des procédures de recherche qui utilisent implicitement des contraintes induites. Ceci est d'autant plus regrettable que les applications réelles telles que la nôtre requièrent souvent des propriétés de résolution dynamiques : il est intéressant, après obtention d'une première solution, d'être capable d'en trouver une seconde rapidement, à la suite d'une légère modification des données du problème. La mémorisation du travail effectué sous la forme de contraintes induites est sans doute le meilleur moyen d'acquérir cette propriété [SV94a, VS95, BJ95].

Notre objectif est de mettre en place dans le cadre des CSP valués un outil de résolution équivalent aux contraintes induites connues dans le cadre classique. La richesse des possibilités offertes par les contraintes induites et le besoin ressenti dans les derniers algorithmes apparus pour résoudre les CSP valués nous incitent à explorer cette voie.

Dans une première section, nous proposons une modélisation simple qui nous permettra de prendre en compte les contraintes induites sans restrictions dans les CSP valués. Celle-ci permettra l'assimilation de toutes les contraintes (induites ou d'origine) sous une forme homogène puis la définition d'un opérateur d'agrégation idempotent. Les trois sections qui suivront explorent les possibilités de ce nouvel outil en étendant les principaux algorithmes du cadre des CSP classiques qui exploitent les contraintes induites : l'obtention et le maintien de k-cohérence, le Nogood Recording et le Backtrack Dynamique.

3.1 Contraintes induites

Le problème essentiel posé par les CSP valués provient de l'éventuelle non idempotence de l'opérateur d'agrégation. Dans ce cas, l'introduction d'une nouvelle contrainte, fût-elle induite par le CSP valué, risque de modifier la valuation des solutions : dupliquer une contrainte c revient à remplacer sa pénalité $\phi(c)$ lors de l'estimation des valuations des affectations par $\phi(c) \oplus \phi(c)$: sans la propriété d'idempotence, les valuations risquent donc d'être surévaluées.

Pallier ce problème impose d'éviter toute redondance lors des agrégations. Cet objectif nous conduit naturellement, en cas d'agrégation de nouvelles contraintes induites par un CSP, à nous intéresser aux contraintes sur lesquelles elles s'appuient.

Nous définissons donc les contraintes induites comme l'association d'un n-uplet de valeurs, d'une valuation signifiant l'importance de l'interdiction du n-uplet, ainsi que d'un ensemble de contraintes justifiant l'attribution de la valuation au n-uplet.

3.1.1 Définition des contraintes induites

Définition 3.1 *Une contrainte induite est un triplet $(A, v, \mathcal{C})$, où A est une affectation partielle, v une valuation et $\mathcal{C}$ un ensemble de contraintes (appelé "justification") tel que v est une borne inférieure de la valuation globale de A sur le problème réduit aux contraintes de $\mathcal{C}$ (pour toute affectation A' extension complète de A, $v \preceq v_{\mathcal{C}}(A')$).*

L'arité d'une contrainte induite continue de désigner le nombre de variables sur lesquelles elle porte, soit la taille de son affectation. Nous dirons qu'une contrainte induite est violée par une affectation partielle si cette dernière contient l'affectation de la contrainte.

Il faut noter que les contraintes induites ne font explicitement référence qu'à un unique n-uplet.

Par exemple, sur notre petit problème d'illustration, la contrainte induite $\zeta_0 = (\{\mathbf{X_1^a}, \mathbf{X_3^a}\}, \mathbf{2}, \{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_4}, \mathbf{C_5}\})$ *est valide : toute affectation qui contient* $\{X_1^a, X_3^a\}$ *possède une valuation globale supérieure ou égale à* 2.

Nous ferons fréquemment référence à notre exemple, et à certaines contraintes qu'il induit, pour illustrer les propriétés qui permettent de manipuler les contraintes induites.

Tout d'abord, une propriété suivie d'une définition tout d'abord pour homogénéiser les contraintes induites et originales :

Propriété 3.2 (construction) *Soit c une contrainte, et A_c un n-uplet de valeurs interdites par c, $(A_c, \phi(c), \{c\})$ est une contrainte induite.*

Nous appellerons ces contraintes induites particulières "**contraintes induites directes**". Il s'agit tout simplement de la traduction des contraintes initiales du problème sous forme de nogood.

La traduction de C_1 donne les trois contraintes induites :
$(\{\mathbf{X_1^a}, \mathbf{X_3^a}\}, \mathbf{1}, \{\mathbf{C_1}\})$
$(\{\mathbf{X_1^a}, \mathbf{X_3^b}\}, \mathbf{1}, \{\mathbf{C_1}\})$
$(\{\mathbf{X_1^b}, \mathbf{X_3^b}\}, \mathbf{1}, \{\mathbf{C_1}\})$

Nous assimilerons dans les sections suivantes les contraintes d'origine aux contraintes induites directes qui leur correspondent. La correspondance conduit à un éclatement des contraintes d'origine en autant de contraintes induites que de n-uplets interdits. Cependant, cette traduction est uniquement formelle et ne correspond à aucune transformation pratique des contraintes d'origine. L'essentiel consiste à identifier la justification des contraintes d'origine comme étant cette contrainte même. Notre objectif est de remplacer les contraintes d'un VCSP par la notion plus complète de "contraintes induites" sur laquelle nous pourrons définir un opérateur d'agrégation de remplacement idempotent. Les premières propriétés qui suivent sont les traductions dans le cadre des VCSP des propriétés concernant les nogoods.

Propriété 3.3 *Soit $(A, v, \mathcal{C})$ une contrainte induite, $A' \supset A$, $v' \preceq v$ et $\mathcal{C}' \supset \mathcal{C}$.*

$(A', v, \mathcal{C})$, $(A, v', \mathcal{C})$ et $(A, v, \mathcal{C}')$ sont des contraintes induites.

Démonstration : (3.3) *La première propriété est évidente car toute extension complète de A' est extension complète de A. La seconde l'est tout autant car si v est une borne inférieure, $v' \preceq v$ aussi. La troisième se déduit de la monotonie de l'agrégation : $\forall A'', \mathcal{C} \subset \mathcal{C}' \Rightarrow v_{\mathcal{C}'}(A'') \succeq v_{\mathcal{C}}(A'')$.*

Les contraintes induites qui suivent sont valides car elles ne sont que des répliques affaiblies de ζ_0.

$$\zeta_0' = (\{\mathbf{X_1^a}, \mathbf{X_2^a}, \mathbf{X_3^a}\}, \mathbf{2}, \{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_4}, \mathbf{C_5}\})$$
$$\zeta_0'' = (\{\mathbf{X_1^a}, \mathbf{X_3^a}\}, \mathbf{1}, \{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_4}, \mathbf{C_5}\})$$
$$\zeta_0''' = (\{\mathbf{X_1^a}, \mathbf{X_3^a}\}, \mathbf{2}, \{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_3}, \mathbf{C_4}, \mathbf{C_5}\})$$

Ces propriétés permettent de classer les contraintes induites :

- plus leur affectation est petite, plus elles ont de chances d'être violées donc utilisées ;
- plus leur valuation est élevée, plus l'interdiction est forte et leur utilité potentielle élevée ;
- plus la justification est petite, plus l'information est précise. Nous verrons plus tard l'importance que prend cette précision lorsque l'on cherche à éviter les évaluations redondantes.

Les deux propriétés qui suivent donnent les principes de production des contraintes induites.

Propriété 3.4 (construction) *Soit A une affectation partielle, et $\mathcal{C}$ un ensemble de contraintes qu'elle viole, alors $(A, \bigoplus_{c \in \mathcal{C}} \phi(c), \mathcal{C})$ est une contrainte induite.*

Il en découle en particulier que si $\mathcal{C}$ est l'ensemble de toutes les contraintes violées par une affectation A, $(A, v(A), \mathcal{C})$ est une contrainte induite. Cette contrainte correspond à l'évaluation de **BT-Evalue** dans le sens où elle ne traduit qu'un examen des contraintes violées par A et fournit la valuation locale résultante.

L'examen des violations de $\{X_1^a, X_2^b, X_3^a, X_4^b, X_5^a\}$ permet de déduire la contrainte induite :

$$\zeta_1 = (\{\mathbf{X_1^a}, \mathbf{X_2^b}, \mathbf{X_3^a}, \mathbf{X_4^b}, \mathbf{X_5^a}\}, \mathbf{3}, \{\mathbf{C_1}, \mathbf{C_4}, \mathbf{C_5}\})$$

Propriété 3.5 (union) †

Soit $x^1,\ldots,x^n$ les valeurs d'une variable $x \notin \tilde{A}$ et $(A_1 \cup \{x^1\}, v_1, \mathcal{C}_1),\ldots,(A_n \cup \{x^n\}, v_n, \mathcal{C}_n)$ des contraintes induites.
Si les affectations A_i sont compatibles alors $(\bigcup_i(A_i), \min_i(v_i), \bigcup_i(\mathcal{C}_i))$ est une contrainte induite.

Démonstration : (3.5) *Soit A une extension complète de $\bigcup_i(A_i)$, A affecte nécessairement x à une valeur x^k. $A_k \cup \{x^k\}$ est donc inclus dans A. Or, d'après les propriétés 3.3, $(A_k \cup \{v^k\}, v_k, \bigcup_i(\mathcal{C}_i))$ est une contrainte induite donc $v_{\bigcup_i(\mathcal{C}_i)}(A) \succeq v_k \succeq \min_i(v_i)$.*

Nous conserverons la notation "$\bigvee$" pour l'opérateur correspondant à cette propriété.

La propriété de construction permet de produire une nouvelle contrainte induite :
$\zeta_2 = (\{\mathbf{X_1^a}, \mathbf{X_2^b}, \mathbf{X_3^a}, \mathbf{X_4^b}, \mathbf{X_5^b}\}, \mathbf{2}, \{\mathbf{C_1}, \mathbf{C_5}\})$. *Comme ζ_1 et ζ_2 concernent les extensions d'une même affectation avec toutes les valeurs de X_5, leur union peut être réalisée :*
$\zeta_3 = \bigvee(\zeta_1, \zeta_2) = (\{\mathbf{X_1^a}, \mathbf{X_2^b}, \mathbf{X_3^a}, \mathbf{X_4^b}\}, \mathbf{2}, \{\mathbf{C_1}, \mathbf{C_4}, \mathbf{C_5}\})$.

Propriété 3.6 (projection.1) *Soit $(A, v, \mathcal{C})$ une contrainte induite et $\mathcal{C}'$ un sous-ensemble des contraintes de $\mathcal{C}$ que les extensions de A ne peuvent pas violer. $(A, v, \mathcal{C} - \mathcal{C}')$ est une contrainte induite.*

Démonstration : (3.6) *Soit A' une extension complète de A, A' ne viole aucune contrainte de $\mathcal{C}'$, donc $v_{\mathcal{C}-\mathcal{C}'}(A') = v_{\mathcal{C}}(A') \succeq v$.*

Propriété 3.7 (projection.2) *Soit $(A, v, \mathcal{C})$ une contrainte induite, et A' l'ensemble des valeurs de A absentes des n-uplets interdits par les contraintes de $\mathcal{C}$. $(A - A', v, \mathcal{C})$ est une contrainte induite.*

Démonstration : (3.7) *Soit A'' une extension complète de $A - A'$, et A''_{bis} l'extension complète de A égale à A'' excepté sur les valeurs des variables de A'. Comme les valeurs*

†. Cette propriété utilise la notion de minimum d'un ensemble de valuations. Avec un ordre partiel, cette notion est moins évidente. Il s'agit intuitivement d'une valuation que l'on souhaite aussi élevée que possible et qui soit inférieure à toutes les valuations de l'ensemble (sans être nécessairement égale à l'une d'entre elles).

de A' n'entrent pas en jeu dans les contraintes de $\mathcal{C}$, $v_{\mathcal{C}}(A''_{bis} - A') = v_{\mathcal{C}}(A''_{bis}) \succeq v$. Or A'' est une extension de $A''_{bis} - A'$, donc $v_{\mathcal{C}}(A'') \succeq v_{\mathcal{C}}(A''_{bis} - A') \succeq v$.

> *La contrainte induite $\zeta'_0 = (\{X_1^a, X_2^c, X_3^a\}, 2, \{C_1, C_2, C_4, C_5\})$ présente une contrainte inutile dans sa justification qui peut être détectée par la première propriété de projection. En effet, les extensions de $\{X_1^a, X_2^c, X_3^a\}$ ne peuvent en aucune manière violer C_2 (car elle est d'ores et déjà satisfaite, ou, plus finement, car X_2^c n'appartient à aucun n-uplet interdit).*
>
> $\zeta'_{0\mathbf{p}} = (\{\mathbf{X_1^a}, \mathbf{X_2^c}, \mathbf{X_3^a}\}, \mathbf{2}, \{\mathbf{C_1}, \mathbf{C_3}, \mathbf{C_4}\})$ *est aussi une contrainte induite.*

> *Or, maintenant, la contrainte induite ζ'_{0p} possède dans son affectation une valeur inutile. En effet, X_2 n'intervient dans aucune contrainte de sa justification.*
>
> $\zeta'_{0\mathbf{pp}} = (\{\mathbf{X_1^a}, \mathbf{X_3^a}\}, \mathbf{2}, \{\mathbf{C_1}, \mathbf{C_3}, \mathbf{C_4}\})$ *est donc une nouvelle contrainte induite.*

Notons que les projections sont tout de même limitées. Elles permettent de rétablir la dégradation de ζ_0 en ζ'_0, mais pas de remonter de $\zeta'_0\zeta'_0$ à ζ_0.

La définition des contraintes induites pour les VCSP et de ces quelques théorèmes répond au besoin de mémorisation d'informations construites durant l'exploration. L'efficacité de leur utilisation nécessite cependant quelques développements supplémentaires car les limitations constatées dans le chapitre précédent persistent.

Les contraintes induites construites par la propriété de construction 3.4 doivent choisir entre une petite justification et une valuation élevée. Dans le premier cas, le nombre de contraintes sur lesquelles elles reposent anihilent l'efficacité des projections : ces contraintes ne seront que rarement violées donc rarement utiles. Un premier objectif sera de rendre possible une réduction des nogoods plus efficace que celle offerte par les projections. Dans le second cas, la valuation des contraintes induites est trop faible pour conduire au rejet des affectations qui les violent. Un second objectif consistera à agréger les valuations des contraintes induites entre elles.

3.1.2 Réduction et augmentation des contraintes induites

Réduction

Lorsque la projection est inefficace, la seule solution consiste à éliminer autoritairement certaines contraintes de la justification au prix d'une réduction de la valuation. Dans

cette optique, la définition préalable d'un opérateur "$\ominus$" de soustraction des valuations est nécessaire. Nous le définissons rigoureusement comme suit :

Définition 3.8 (soustraction) *Soit v et v' deux éléments de l'ensemble E des valuations, $v'' = v \ominus v'$ est défini comme $v'' = \min(\epsilon \in E / \epsilon \oplus v' \succeq v)$* †.

L'idéal serait d'exhiber l'élément $\epsilon / \epsilon \oplus v' = v$. Dans le doute (inexistance ou multiplicité), on considère le plus petit des éléments permettant de reconstituer au moins la valuation v.

Propriété 3.9 (réduction) *Soit $\zeta = (A, v, \mathcal{C})$ une contrainte induite, et c une contrainte de $\mathcal{C}$. Soit $v' = v \ominus \phi(c)$* ‡*, $(A, v', \mathcal{C} - \{c\})$ est une contrainte induite (notée $\downarrow_c (\zeta)$).*

Démonstration : (3.9) *Soit A' une extension complète de A. Si A' ne viole pas c, $v_{\mathcal{C}}(A') = v_{\mathcal{C}-\{c\}}(A') \succeq v \succeq v'$* [1]*. Dans le cas contraire, $v_{\mathcal{C}}(A') = v_{\mathcal{C}-\{c\}}(A') \oplus \phi(c) \succeq v$. Donc $v' \preceq v_{\mathcal{C}-\{c\}}(A')$ d'après sa définition.*

Soit A une affectation et ζ une contrainte induite violée par A, nous noterons $\Downarrow_A (\zeta)$ le résultat des réductions successives de ζ avec toutes les contraintes violées par A, et $\Downarrow (\zeta)$ le résultat des réductions successives de ζ avec toutes les contraintes violées par son affectation.

> *Considérons la contrainte induite $\zeta_3 = (\{X_1^a, X_2^b, X_3^a, X_4^b\}, 2, \{C_1, C_4, C_5\})$, son affectation viole la contrainte C_1.*
>
> $\Downarrow (\zeta_3) = (\{\mathbf{X_1^a}, \mathbf{X_2^b}, \mathbf{X_3^a}, \mathbf{X_4^b}\}, \mathbf{2}, \{\mathbf{C_4}, \mathbf{C_5}\})$. *Après projection, les valeurs X_1^a et X_2^b disparaissent de l'affectation de cette contrainte induite (le retrait de X_1^a est exclusivement dû à la réduction). On déduit donc la contrainte induite :*
>
> $\zeta_3' = (\{\mathbf{X_3^a}, \mathbf{X_4^b}\}, \mathbf{1}, \{\mathbf{C_4}, \mathbf{C_5}\})$

Par la suite, l'idée sur laquelle nous nous appuyons pour réduire la justification d'une contrainte induite sans diminuer la valuation qu'elle induit est la suivante. Une contrainte induite donne une estimation de la valuation globale de son affectation. Cette estimation repose sur deux types d'informations : la valuation des contraintes d'origine violée par

†. Dans le cas d'un ordre partiel, on entend par min un élément inférieur à tous les autres.

‡. Si la valuation de c n'est pas unique et n'est pas déterminée par A, on prendra le cas le plus défavorable donc la valuation la plus élevée.

1. Comme $v \oplus \phi(c) \succeq v$, $v' = \min(\epsilon \in E / \epsilon \oplus \phi(c) \succeq v) \succeq v$.

l'affectation de la contrainte induite, et les valuations des contraintes dont on prévoit la violation en cas d'extension complète. Seule cette seconde information mérite d'être mémorisée puisque lorsque la contrainte induite est utilisable (donc violée) la première information peut être restituée par un simple constat des violations par une procédure du type de **BT-evalue**. On doit donc pouvoir se permettre de réduire la justification d'une contrainte induite, en retirant toutes les contraintes que son affectation viole, à condition d'être capable, par la suite, de lui agréger les valuations de contraintes violées.

Augmentation

Propriété 3.10 (augmentation) *Soit $\zeta = (A, v, \mathcal{C})$ une contrainte induite et c une contrainte violée par A. Si $c \notin \mathcal{C}$, $(A, v \oplus \phi(c), \mathcal{C} \cup \{c\})$ est une contrainte induite (notée $\uparrow^c(\zeta)$).*

Démonstration : (3.10) *Soit A' une extension complète de A. Si $c \notin \mathcal{C}$, comme A' viole c, $v_{\mathcal{C}\cup\{c\}}(A') = v_{\mathcal{C}}(A') \oplus \phi(c) \succeq v \oplus \phi(c)$.*

Soit A une affectation et $\zeta = (A', v, \mathcal{C})$ une contrainte induite violée par A, nous noterons $\Uparrow^A(\zeta)$ le résultat des augmentations successives de $(A, v, \mathcal{C})$[2] avec toutes les contraintes violées par A.

> *$A = \{X_1^a, X_2^b, X_3^a, X_4^b\}$ est une affectation qui viole ζ_3' et la contrainte C_1. Cette dernière n'appartenant pas à la justification de ζ_3', on a :*
> $\zeta_3'' = \Uparrow^{\mathbf{A}}(\zeta_3') = (\{\mathbf{X_1^a}, \mathbf{X_2^b}, \mathbf{X_3^a}, \mathbf{X_4^b}\}, \mathbf{2}, \{\mathbf{C_1}, \mathbf{C_4}, \mathbf{C_5}\})$. *On retrouve bien sûr la contrainte ζ_3.*

Les problèmes de taille de justification et de valuation sont extrêmement liés, autant que leurs solutions. Aussi cette propriété peut-elle être perçue comme un moyen d'agréger successivement les valuations des contraintes violées par une affectation avec la valuation d'une contrainte induite $(A, v, \mathcal{C})$ d'affectation compatible. Il est néanmoins important de remarquer que la valuation de $\Uparrow^A(\zeta)$ peut tout à fait être inférieure à $v(A)$: cette propriété nécessiterait quelques adaptations si elle devait servir à la définition d'un opérateur d'agrégation. Par contre, lors de l'évaluation d'une affectation partielle A, les seules informations fournies par **BT-evalue** et une contrainte induite ζ ne permettent pas de prouver la validité d'un minorant de sa valuation globale qui serait supérieur à

2. La propriété de classement assure que $(A, v, \mathcal{C})$ est une contrainte induite.

$\max\left(v(A), \phi(\Uparrow^A(\zeta))\right)$. L'augmentation est en ce sens maximale lorsque les contraintes induites viennent en support d'une recherche qui s'appuie sur **BT-evalue**.

Considérations sur l'opportunité des réductions et augmentations

Il apparaît tout d'abord que grâce à cette augmentation, une contrainte induite réduite offre toujours une estimation des valuations globales au moins aussi intéressante que celle qu'afficherait la contrainte induite d'origine.

Propriété 3.11 (équivalence) *Soit ζ une contrainte induite, et $\zeta' = \Downarrow(\zeta)$ sa réduction naturelle. Si A' est une affectation violant ζ, A' viole aussi ζ', et la valuation de $\Uparrow^{A'}(\zeta')$ est supérieure ou égale à celle de $\Uparrow^{A'}(\zeta)$.*

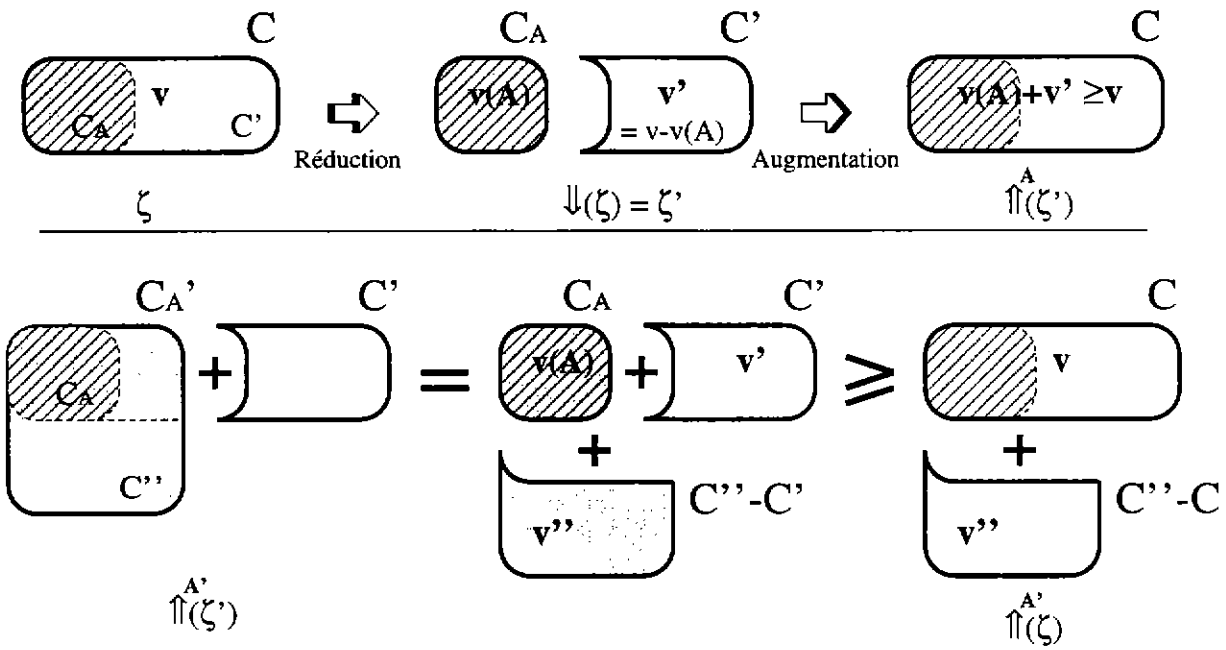

FIG. 3.1 – *Equivalence $\Uparrow^{A'}(\zeta)$, $\Uparrow^{A'}(\Downarrow(\zeta))$*

D'abord un lemme permettant de reconstruire par augmentation la contrainte induite d'origine (cf figure 3.1) :

Lemme 3.12 *Soit $\zeta = (A, v, C)$ une contrainte induite, et $\zeta' = \Downarrow(\zeta)$ sa réduction naturelle. Si C_A est le sous-ensemble des contraintes de C violées par A, $\uparrow_A^C(\zeta')$ possède la même affectation et la même justification que ζ, et une valuation supérieure ou égale.*

Démonstration : (3.12) *ζ' est la réduction de ζ avec toutes les contraintes de $\mathcal{C}_A$. Après chaque réduction avec une contrainte c de $\mathcal{C}_A$, l'ancienne valuation est par définition inférieure ou égale à l'agrégation de la nouvelle avec $\phi(c)$. En final, la valuation v' de ζ' est telle que $v \preceq v' \oplus \bigoplus_{c \in \mathcal{C}_A} \phi(c)$. Or, puisque toutes les contraintes de $\mathcal{C}_A$ ont étés retirées, elles n'appartiennent pas à la justification de ζ'. Par conséquent, la valuation de l'augmentation est exactement $v' \oplus \bigoplus_{c \in \mathcal{C}_A} \phi(c)$. Par ailleurs, les contraintes restituées à la justification sont exactement celles qui furent supprimées par la réduction.*

La propriété se démontre alors facilement (figure 3.1).

Démonstration : (3.11) *Soit $\mathcal{C}_{A'}$ l'ensemble des contraintes violées par A', il se compose d'une part de $\mathcal{C}_A$, l'ensemble des contraintes violées par A et incluses dans $\mathcal{C}$, et d'autre part de l'ensemble $\mathcal{C}''$ des contraintes restantes. $\Uparrow^{A'}(\zeta') = \uparrow^{\mathcal{C}_{A'}}(\zeta') = \uparrow^{\mathcal{C}''}(\uparrow^{\mathcal{C}_A}(\zeta'))$ et $\Uparrow^{A'}(\zeta) = \uparrow^{\mathcal{C}''}(\zeta)$. D'après le lemme, $\uparrow^{\mathcal{C}_A}(\zeta')$ est identique à ζ en tous points sauf la valuation qui est supérieure ou égale. La monotonie de l'agrégation nous assure que l'augmentation avec les contraintes de $\mathcal{C}''$ conservera cette supériorité.*

Lorsque l'on ne s'intéresse aux contraintes induites que de manière individuelle, on a toujours intérêt à effectuer une réduction naturelle des contraintes induites que l'on construit. On sera toujours en mesure, le moment venu, de rétablir au moins la valuation initiale par une augmentation. En outre, la réduction de la justification permettra de plus importantes projections et par conséquent une plus grande utilité des contraintes induites.

Une autre manière de voir consiste à dire que l'augmentation permet de générer tout un ensemble de contraintes induites à partir d'une seule contrainte induite, par augmentation avec l'ensemble des affectations partielles. On s'intéresse alors à l'ensemble des contraintes engendrées. Ce-dernier définit une relation d'ordre partiel entre les contraintes, selon l'inclusion de leur ensembles engendrés. En ce sens, la réduction produit la contrainte induite maximale engendrant la contrainte que l'on souhaite réduire : toute contrainte engendrant $(A, v, \mathcal{C})$ est elle même engendrée par $\Downarrow_A (A, v, \mathcal{C})$. Non seulement la réduction n'est jamais néfaste, mais en outre, compte tenu de l'opérateur d'augmentation que nous avons défini, elle est optimale.

Cependant, rien n'interdit de réduire les contraintes induites de manière plus forte ou simplement différente de la réduction naturelle. Couplée avec la projection, la réduction permet de jouer sur tous les paramètres des contraintes induites. On peut sélectionner

précisément les contraintes que l'on souhaite conserver dans la justification, donc il est aussi possible de sélectionner les valeurs concernées par la justification. Enfin, mais cela ne présente aucun intérêt, la valuation peut être diminuée à volonté.

Un embryon de réponse au problème de l'agrégation des valuations de contraintes avec des contraintes induites est donné par la propriété d'augmentation. Il s'avère suffisant dans les cas où il n'est pas nécessaire de tirer la meilleure valuation possible d'un ensemble de contraintes violées, induites ou non. Dans le cas contraire, la définition d'un opérateur plus complet s'impose.

3.1.3 Agrégation des contraintes induites

Posons maintenant les bases d'un opérateur d'agrégation des contraintes induites. Ce-dernier doit être idempotent pour permettre l'adjonction de contraintes induites. Or il ressort de l'étude de [SFV95] qu'un opérateur idempotent et valide pour une structure de valuation est nécessairement équivalent à l'opérateur "max". Un tel opérateur est parfaitement inutile lorsqu'il est nécessaire de cumuler des valuations pour estimer les solutions puisque justement l'opérateur "max" ne fait aucun cumul. Il y a donc une incompatibilité entre le besoin d'idempotence de l'opérateur et celui de cumul des valuations. Tout compromis nous impose de sacrifier une des propriétés "mineures" de l'agrégation, la commutativité ou l'associativité. En l'occurrence, ce sera l'associativité.

La définition de l'agrégation que nous allons donner est relativement indépendante des contraintes induites. Elle consiste à sommer des valuations en tenant compte de leur justification mais sans réellement s'intéresser aux violations[‡].

Pour plus de clarté, nous procéderons en deux temps. Nous commencerons par définir provisoirement un opérateur binaire simple et intuitif, puis nous passerons à un opérateur plus complet et plus précis en terme de valuation résultante.

‡. Ceci pose problème pour l'agrégation des contraintes (d'origine) dont la valuation est déterminée par la nature de la violation. Il faut noter que, par la suite, l'agrégation sera toujours utilisée pour agréger des contraintes violées par une affectation donnée. La valuation à prendre en compte est alors fixée.

Opérateur binaire minimal

On peut assez facilement définir un premier opérateur d'agrégation binaire par le simple examen de l'intersection des justifications. Ce-dernier consiste à soustraire la valuation des contraintes redondantes.

Définition 3.13 *Soit ζ_1 et ζ_2 deux contraintes induites, et, $\mathcal{C}_1$ et $\mathcal{C}_2$ leurs justifications, $\zeta_1 \odot \zeta_2 = max(\phi(\zeta_1), \phi(\zeta_2), \phi(\zeta_1) \oplus \phi(\zeta_2) \ominus \bigoplus_{c \in \mathcal{C}_1 \cap \mathcal{C}_2} \phi(c))$*‡.

Propriété 3.14 *D'autre part, si ζ_1 et ζ_2 sont violées par une affectation A, $(A, \zeta_1 \odot \zeta_2, \mathcal{C}_1 \cup \mathcal{C}_2)$ est une contrainte induite.*

Démonstration : (3.14) *Soit A' une extension complète de A. A' viole ζ_1 et ζ_2, donc $v_{\mathcal{C}_1}(A') = v_{\mathcal{C}_1 - \mathcal{C}_1 \cap \mathcal{C}_2}(A') \oplus v_{\mathcal{C}_1 \cap \mathcal{C}_2}(A') \succeq \phi(\zeta_1)$ et $v_{\mathcal{C}_2} = v_{\mathcal{C}_2 - \mathcal{C}_1 \cap \mathcal{C}_2}(A') \oplus v_{\mathcal{C}_1 \cap \mathcal{C}_2}(A') \succeq \phi(\zeta_2)$. En sommant les deux équations, on obtient: $\phi(\zeta_2) \oplus \phi(\zeta_2) \preceq v_{\mathcal{C}_1 \cup \mathcal{C}_2}(A') \oplus v_{\mathcal{C}_1 \cap \mathcal{C}_2}(A') \preceq v_{\mathcal{C}_1 \cup \mathcal{C}_2}(A') \oplus \bigoplus_{c \in \mathcal{C}_1 \cap \mathcal{C}_2} \phi(c))$. Enfin, la définition de la soustraction nous assure que $v_{\mathcal{C}_1 \cup \mathcal{C}_2}(A') \succeq (\phi(\zeta_2) \oplus \phi(\zeta_2)) \ominus \bigoplus_{c \in \mathcal{C}_1 \cap \mathcal{C}_2} \phi(c))$. Donc, dans tous les cas, $v_{\mathcal{C}_1 \cup \mathcal{C}_2}(A') \succeq \phi(\zeta_2) \odot \phi(\zeta_2)$.*

En d'autre termes, pour dériver une nouvelle contrainte induite il suffit d'examiner les justifications afin de ne compter qu'une fois les contraintes communes aux justifications des contraintes induites agrégées. Pour assurer la validité du résultat, on se place en pire cas ie. on considère la valuation maximale qui puisse être prise en compte de manière redondante. On peut ainsi cumuler, du moins partiellement, les valuations d'un ensemble de contraintes induites.

Cet opérateur simple n'est cependant pas associatif: chaque agrégation augmente la justification et hypothèque les agrégations ultérieures. De fait, pour le calcul des valuations locales, il est important d'agréger en priorité les contraintes induites directes si l'on veut s'assurer d'un résultat au moins équivalent à celui obtenu avec sans contraintes induites.

Dans tous les cas, l'importance des justifications des contraintes induites est mise en évidence : elles influent directement sur la valuation que l'on peut attendre d'une agrégation. On aura intérêt à les réduire au maximum afin de limiter autant que possible leurs intersections.

‡. Pour une contrainte qui peut prendre plusieurs valuations, on considère le cas le plus pessimiste, à savoir la valuation maximale.

En effet, le cas de justifications disjointes, tel que celui des contraintes induites directes (propriété 3.17), est idéal dans la mesure où la valuation résultante est la somme des valuations des contraintes induites. Dans le cas contraire, elle serait probablement inférieure.

A l'opposé, si les justifications des contraintes induites agrégées sont identiques, la valuation résultante correspond simplement au maximum des valuations des contraintes induites, sans qu'aucune agrégation de ces valuations ne soit réalisable. Or, les propriétés de classement 3.3 montrent que l'on peut toujours augmenter la justification d'une contrainte induite.

Par ailleurs, on ne dispose pas d'un théorème permettant de restreindre *a posteriori* la justification d'une contrainte au minimum nécessaire pour justifier sa valuation. Il existe par conséquent un risque important de "**noyade**" des justifications. Il est donc capital, lors de la création des contraintes induites, de gérer avec soin leur justifications.

Pour contourner le problème que pose l'ordre des agrégations, il nous semble nécessaire de définir un opérateur plus complet. Nous ne nous étendrons pas sur l'opérateur simple que nous venons de définir car il ne constitue qu'un cas particulier d'utilisation de l'opérateur qui suit.

Opérateur ensembliste

Cette fois, nous comptons agréger simultanément un ensemble de contraintes. Nous utilisons la notion "**ensemble suffisant**" de contraintes d'origines que l'on peut rapprocher de la notion de "configuration prometteuse" [JB97] définie dans le cadre de la satisfaction. Une contrainte induite est la prévision de violations dans un sous-ensemble de contraintes d'origine, et plus précisément, c'est une prévision par défaut de la valuation qu'il faut attendre. Soit un ensemble de contraintes induites, une affectation complète qui les viole toutes respecte nécessairement toutes les prévisions (à savoir que la valuation des violations dans chaque justification est supérieure ou égale à la valuation prévue). Un ensemble suffisant est intuitivement un ensemble de violations pouvant potentiellement être celui d'une affectation complète qui violerait toutes les contraintes induites; nous l'appelons "suffisant" car il suffit à la satisfaction de toutes les prévisions associées aux contraintes induites. Le plus petit de ces ensembles possibles donne un minorant des valuations que l'on doit attendre et tient compte simultanément de plusieurs contraintes

induites (d'où notre intérêt pour cette notion d'ensemble suffisant).

Définition 3.15 *agrégation Soit* $\{\zeta_i = (A_i, v_i, \mathcal{C}_i), 0 \leq i \leq n\}$ *un ensemble de contraintes induites. Nous appelons* "**ensemble suffisant**" *un ensemble de contraintes* $\mathcal{C}$[‡] *tel que* $\forall i, \bigoplus_{c \in (\mathcal{C} \cap \mathcal{C}_i)} \phi(c) \succeq v_i$.

L'opérateur d'agrégation "$\bigodot$*" d'un ensemble de contraintes induites se définit alors par :*

$\bigodot_i \zeta_i = \min_{\mathcal{C} \in \mathbb{C}s} \left(\bigoplus_{c \in \mathcal{C}} \phi(c) \right)$[†] *, où* $\mathbb{C}s$ *est l'ensemble de tous les ensembles suffisants possibles pour les contraintes agrégées.*

Il faut noter que l'opérateur binaire que nous avons défini plus haut est identique à cet nouvel opérateur utilisé comme un opérateur binaire sur des paires de contraintes induites.

Si l'on considère les deux contraintes induites $\zeta_0 = (\{X_1^a, X_3^a\}, 2, \{C_1, C_2, C_4, C_5\})$ et $\zeta_3 = (\{X_1^a, X_2^b, X_3^a, X_4^b\}, 2, \{C_1, C_4, C_5\})$, *des ensembles suffisants pour ces deux contraintes sont* $\{C_1, C_4\}$, $\{C_3, C_4, C_5\}$, $\{C_1, C_2, C_3, C_4\}$, *etc. Ils contiennent au moins deux contraintes et il existe un moyen plausible de ne violer que deux contraintes, donc* $\bigodot(\zeta_0, \zeta_3) = 2$.

Propriété 3.16 *Soit* $\mathcal{I}$ *un ensemble de contraintes induites* $(A_i, v_i, \mathcal{C}_i)$ *telles que les affectations* A_i *soient compatibles. Alors* $(\bigcup_i A_i, \bigodot_{\zeta_i \in \mathcal{I}} \zeta_i, \bigcup_i \mathcal{C}_i)$ *est une contrainte induite.*

Démonstration : (3.16) *Soit* A *une extension complète de* $\bigcup_i A_i$ *et* $\mathcal{C}$ *l'ensemble des contraintes de* $\mathcal{C}_\mathcal{I}$ *violées par* A. *La définition des contraintes induites impose que* $\forall (A_i, v_i, \mathcal{C}_i) \in \mathcal{I}, v_{\mathcal{C}_i}(A) = \bigoplus_{c \in (\mathcal{C} \cap \mathcal{C}_i)} \phi(c) \succeq v_i$: $\mathcal{C}$ *est un ensemble suffisant. Donc, par construction,* $\bigoplus_{c \in \mathcal{C}} \phi(c) \succeq \bigodot_{\zeta_i \in \mathcal{I}} \zeta_i$. *Par suite,* $v_{\bigcup_i \mathcal{C}_i}(A) = v_\mathcal{C}(A) \succeq \bigodot_{\zeta_i \in \mathcal{I}} \zeta_i$.

La valuation de l'agrégation des contraintes induites portant sur une affectation partielle fournit donc un minorant de la valuation globale de cette dernière.

‡. Dans le cas de valuations multiples pour une même contrainte, il convient de sélectionner en même temps que les contraintes la valuation qui leur est attribuée. L'énumération des possibilités est cette fois inévitable et conduit, du moins pour le calcul, à l'éclatement implicite des contraintes en fonction des valuations possibles.

†. A préciser dans le cas d'un ordre partiel.

Propriété 3.17 *Soit $\mathcal{I}$ un ensemble de contraintes induites directes violées par une même affectation partielle, et $\mathcal{C}$ l'ensemble des contraintes d'origine correspondant. Alors* $\bigodot_{\zeta \in \mathcal{I}} \zeta = \bigoplus_{c \in \mathcal{C}} \phi(c)$.

Démonstration : (3.17) *Un ensemble suffisant contient obligatoirement l'unique contrainte de la justification de chaque contrainte induite directe ; il contient au moins $\mathcal{C}$, union de toutes les justifications. Or, $\mathcal{C}$ est un ensemble de contraintes suffisant. La monotonie de $\oplus$ nous assure donc que la valuation des contraintes de tout autre ensemble suffisant est supérieure ou égale à* $\bigoplus_{c \in \mathcal{C}} \phi(c)$ [†] .

Si l'on remarque que A viole toutes les contraintes de $\mathcal{C}$ comme de $\mathcal{I}$, cette propriété permet un parallèle entre l'agrégation des valuations des contraintes et la valuation de l'agrégation des contraintes induites directes.

Ce nouvel opérateur $\odot$ présente quelques propriétés équivalentes à celles de $\oplus$, et l'idempotence en plus :

- la valuation $\bot$ est neutre, au sens où, lors de l'agrégation d'un ensemble de contraintes induites, la prise en compte (ou non) des contraintes induites de valuation $\bot$ ne modifie pas la valuation du résultat. En effet, leur présence n'impose aucune condition supplémentaire sur les ensembles de contraintes suffisants ;
- la commutativité et l'associativité sont contournées par le fait que l'opérateur n'est pas binaire, mais concerne simultanément un ensemble de contraintes. Cependant, l'agrégation binaire des contraintes induites et plus généralement l'agrégation par parties n'est pas associative ;
- l'agrégation est monotone : soit $\mathcal{I}$ un ensemble de contraintes induites et $\mathcal{I}' \subset \mathcal{I}$, alors $\bigodot_{\zeta \in \mathcal{I}'} \zeta \preceq \bigodot_{\zeta \in \mathcal{I}} \zeta$. Augmenter le nombre de contraintes induites a pour conséquence la diminution de l'ensemble des ensembles suffisants ;
- l'opérateur est idempotent : la duplication des contraintes induites dans un ensemble ne modifie par la contrainte induite résultant de l'agrégation.

†. Ici, même dans le cas d'un ordre partiel, l'ensemble des valuations des ensembles suffisants possède un élément inférieur à tous les autres.

Nous n'en faisons pas la démonstration car il faudrait commencer par donner un signification à toutes ces propriétés pour un opérateur non binaire. En fait, ces propriétés ne revètent que peu d'importance, il s'agit surtout de comprendre la démarche. En ce qui concernent la validité de $\odot$, seules les propriétés le positionnant par rapport à l'opérateur $\oplus$ sont nécessaires, ce dernier servant alors de "tuteur". Nous précisons cela dans la section qui suit.

3.1.4 VCSP induits

Définition 3.18 *Soit un VCSP* $(\mathbb{V}, \mathbb{C}, \mathbb{S})$, *un VCSP* induit *est un triplet* $(\mathbb{V}, \mathbb{I}, \mathbb{S},)$ *où* $\mathbb{I}$ *est un ensemble de contraintes induites par le VCSP contenant au moins l'ensemble de toutes les contraintes induites directes découlant de* $\mathbb{C}$.

Nous appelons alors *valuation induite (locale)* d'une affectation partielle A (notée $v_{\odot}(A)$) la valuation de l'agrégation des contraintes induites violées par A.

Une solution du VCSP induit est une affectation complète de valuation induite strictement inférieure à $\top$. Il s'agit d'une *solution optimale* s'il n'existe pas de solution de valuation induite plus faible.

Théorème 3.19 $\forall A$ *et* $\forall A'$ *extension complète de* A, $v_{\odot}(A) \preceq v(A')$.

Démonstration : (3.19) *Le résultat de l'agrégation de toutes les contraintes induites est une contrainte induite violée par* A, *et qui respecte donc leur propriété essentielle : leur valuation est inférieure ou égale à celle de toute extension complète.*

Théorème 3.20 $\forall A$, $v(A) \preceq v_{\odot}(A)$.

Démonstration : (3.20) *Soit* $\mathcal{I}$ *l'ensemble de toutes les contraintes induites directes violées par* A, $\mathcal{C}_{\mathcal{I}}$ *l'ensemble de contraintes d'origine correspondant et* $v_{\mathcal{I}} = \bigodot_{\zeta \in \mathcal{I}} \zeta$ *leur agrégation, d'après la propriété 3.17,* $v_{\mathcal{I}} = \bigoplus_{c \in \mathcal{C}_{\mathcal{I}}} \phi(c)$.

Comme toutes les contraintes induites directes sont présentes, $\mathcal{C}_{\mathcal{I}}$ *est l'ensemble des contraintes violées par* A.

Donc $v(A) = v_{\mathcal{I}}$.

La monotonie de l'agrégation des contraintes induites assure finalement que $v(A) \preceq v_{\odot}(A)$

En particulier, il découle de ces deux théorèmes que si A est une affectation complète, $v(A) = v_{\odot}(A)$: un VCSP et un VCSP induit ont les mêmes solutions (affectations et valuations identiques). Le VCSP induit devrait en outre être plus facile à résoudre par un algorithme de type Branch and Bound. En effet, la portion de l'arbre de recherche explorée est d'autant plus réduite que les valuations locales (sources de la borne inférieure du Branch and Bound) sont élevées.

Considérations pratiques

Le calcul proposé fournit réellement une estimation optimale (aussi élevée que possible) de la valuation globale d'une affectation à partir des contraintes induites qui reposent sur elle. Avec la seule donnée des contraintes agrégées, il est impossible de prouver qu'une valeur plus élevée peut satisfaire la propriété 3.16. En effet, l'ensemble de contraintes suffisant minimum exhibé correspond à un contre-exemple : il peut exister une affectation complète qui viole exactement cet ensemble de contraintes. Cependant, il faut noter que tout calcul de valuation locale entre cet optimum $v_{\odot}(A)$ et la valuation $v(A)$ des contraintes d'origine violées satisferait les deux théorèmes 3.19 et 3.20 qui assurent l'équivalence entre un VCSP et un VCSP induit. Tout opérateur monotone et idempotent qui calcule une valuation entre ces deux bornes est correct (la section qui suit en donne un exemple).

Il est important de préciser la marge de liberté dont on dispose dans la définition de cet opérateur car, tel que nous l'avons défini, le calcul de la valuation du résultat est en général NP-complet ! Dans le cadre des VCSP additifs, par exemple, le problème prend la forme classique d'un problème de recouvrement :

Nous considérons donc une structure de valuation $(E, \preceq, \bot, \top, \oplus) = (\mathbb{N}, \leq, 0, \infty, +)$. Précisons dans ce cadre le calcul de la valuation d'une agrégation.

Soit $\mathcal{I} = \{\zeta_1 ... \zeta_i ... \zeta_m\}$ l'ensemble des contraintes à agréger, et $\mathcal{C} = \{c_1 ... c_j ... c_n\}$ l'union des justifications des contraintes de $\mathcal{I}$. Le problème se met aisément sous la forme d'un problème linéaire en nombres entiers :

- n variables $x_{c_1} ... x_{c_n}$ binaires, une par contrainte présente dans les justifications

des contraintes induites mais non dans celles des contraintes induites directes. On cherche un ensemble suffisant de valuation minimale. $x_{c_i} = 1$ si la contrainte c_i appartient à l'ensemble, 0 sinon.

- m contraintes, une par contrainte ζ_i de $\mathcal{I}$ dont la justification est J_{ζ_i}, de la forme $\sum_{c_j \in J_{\zeta_i}} \phi(c_j) \times x_{c_j} \geq \phi(\zeta_i)$.
- un critère à minimiser : $v = \sum_j \phi(c_j) \times x_{c_j}$. ‡

Comme nous l'avons vu lors de la démonstration de la propriété 3.17 , ce sont les contraintes induites indirectes qui rendent l'agrégation complexe. En effet, lorsque l'on cherche un ensemble de contraintes suffisant, l'inclusion des justifications des contraintes induites directes dans ce-dernier est une condition nécessaire et suffisante pour que les inégalités associées à ces contraintes induites directes soient vérifiées. On peut par conséquent fixer à 1 les variables correspondant à des contraintes d'origine.

Malgré tout, cette procédure, comme le problème qu'elle résout n'est pas polynômiale. Il peut être dangereux de s'appuyer sur un tel opérateur. Les quelques expérimentations que nous avons poussées dans ce sens à l'aide du logiciel CPLEX se sont avérées infructueuses du fait de la lenteur de l'agrégation. Non pas que les problèmes à résoudre fussent difficiles (ils étaient en général réductibles à moins de deux équations linéaires), mais le nombre des opérations d'agrégations effectuées dans une recherche amplifie le préjudice causé par les lourdeurs de mise en œuvre du logiciel de programmation linéaire.

Nous chercherons donc, autant que possible, à mettre en œuvre un mode de calcul intermédiaire entre la qualité des solutions et le temps nécessaire à leur obtention. Les remarques de cette section ont pour objet de préciser les limites dans lesquelles il est possible de réduire cette complexité.

Nous pouvons maintenant étendre les algorithmes basés sur des principes de mémorisation de contraintes induites au cas général des VCSP. Nous détaillons ici les algorithmes correspondant d'abord à la k-cohérence. Puis, nous nous intéressons à l'analyse et la mémorisation des échecs sous sa forme la plus générale. Cette trame d'algorithme s'avèrera particulièrement adaptée à notre application. Enfin, nous présentons une extension du

‡. Ceci n'est possible que dans le cas où une seule valuation est associée à chaque contrainte. Dans le cas contraire, une solution sous-optimale consiste à considérer les valuations les moins favorables : la valuation maximale dans les contraintes et la minimale dans le critère.

Backtrack Dynamique qui compense relativement bien sa complexité par des propriétés d'une grande utilité pour la résolution de VCSP.

3.2 Cohérence locale et VCSP

3.2.1 Contraintes induites et prévision des échecs

Avant d'aborder la cohérence locale proprement dite, nous redéfinissons les procédures de prévision des échecs présentées dans le chapitre précédent (**BT-Evalue**, **FC-Evalue**, **DAC-Evalue**). La valuation v, qu'une de ces procédures fournit après examen d'une affectation A, peut nécessairement donner lieu à une contrainte induite interdisant A avec une valuation v. En effet, la validité de ces procédures assure que $(A, v, \mathbb{C})$ est une contrainte induite. Ceci doit nous permettre d'intégrer ces procédures de prévision au cadre des contraintes induites que nous venons de définir. Il reste cependant nécessaire de produire une justification plus raisonnable que l'ensemble de toutes les contraintes. Le principe consiste simplement à ne prendre en compte que les contraintes mises en jeu pour la construction de la valuation. Nous donnons ci-dessous les nouvelles versions qui remplaceront leurs homologues.

BT-induite(A)

- Soit $\mathcal{I}$ l'ensemble des contraintes violées par A
- Soit $v = \bigodot_{\zeta \in \mathcal{I}} \zeta$
- Soit $\mathcal{C}$ l'union des justifications des contraintes de $\mathcal{I}$
- Renvoie $(A, v, \mathcal{C})$

La propriété 3.16 prouve la validité du résultat.

FC-induite(A, x)

- Pour chaque valeur i de x
  Soit ζ_i =**BT-induite**$(A \cup \{x = i\})$
- Renvoie $\bigvee_i \zeta_i$

EFC-induite(A)

- Pour chaque variable $x \notin A$
  Soit $(A_x, v_x, \mathcal{C}_x)$ =**FC-induite**(A, x)
- Soit $v = \bigodot_x (A_x, v_x, \mathcal{C}_x)$
- Renvoie $(A, v, \bigcup_x \mathcal{C}_x)$

Il n'est plus vraiment nécessaire de s'intéresser aux surcoûts car la définition du nouvel opérateur prend en compte les redondances. Dans tous les cas, le découpage en surcoûts portant sur des ensembles de contraintes disjoints n'est plus possible. En effet, les disjonctions entre justifications ne sont plus aussi sûres qu'auparavant car la propriété 2.5 n'est plus vérifiée : des contraintes induites portant sur des variables distinctes peuvent parfaitement posséder des contraintes en commun dans leurs justifications.

D'autre part, l'opérateur que nous avons défini n'est pas associatif et rend très incertaine la notion même de "surcoût". En effet, pour être optimale, l'agrégation des contraintes induites doit être simultanée. Or, le surcoût est par définition une agrégation partielle des violations. En l'occurrence, la version de **EFC-Evalue** que nous proposons n'est pas optimale, et l'adjonction de contraintes induites peut produire l'effet inverse de celui désiré, à savoir une diminution de la valuation retournée !

Bien que les prévisions soient parfaitement compatibles avec les contraintes induites, il apparaît que leur mise en œuvre doit être entourée de précautions que nous allons préciser.

3.2.2 Efficacité des prévisions

Noyade des justifications

Tout d'abord, il convient de prendre en compte les remarques concernant les risques de noyade en limitant au strict nécessaire les justifications des contraintes induites lors de leur création. Il n'existe que trois manières de créer des contraintes induites, dont découlent toutes les autres procédures ; elles correspondent aux propriétés de construction 3.4 , d'agrégation 3.16 et d'union 3.5.

L'utilisation de la première correspond à la procédure **BT-Evalue**(A). A ce niveau, la seule politique raisonnable consiste à justifier la valuation $v(A)$ par l'ensemble des contraintes violées par A. Toute autre contrainte serait superflue et préjudiciable.

Noyade des justifications et agrégation

La seconde propriété (agrégation 3.16) effectue une union rudimentaire des justifications des contraintes induites agrégées pour justifier la valuation résultant de leur agrégation. L'identification de contraintes dont la présence n'est pas nécessaire dans la justification dépend nécessairement du mode de construction de la valuation agrégée.

Un opérateur "max" sélectionnera uniquement la justification de la contrainte induite de valuation maximale. L'opérateur $\odot$ utilisé comme un opérateur binaire (propriété 3.14) sélectionnera selon le cas les deux justifications si le résultat est supérieur aux deux valuations, une seule si le résultat est le maximum des deux valuations.

Dans cette voie, on peut toujours se dispenser de prendre en compte les justifications des contraintes dont l'agrégation est inutile pour atteindre la valuation résultat. Dans le cas général, l'analyse de la valuation retournée par l'opérateur $\odot$ ensembliste ne permet malheureusement pas d'identifier les contraintes induites inutiles. On aura certainement intérêt à effectuer un travail de sélection *a priori* des contraintes induites à agréger.

Noyade des justifications et union

Le dernier mode de création d'une contrainte induite passe par la propriété d'union 3.5.

Chaque contrainte induite utilisée pour une union porte sur une valeur de la variable x sur laquelle se fait l'union. Dans le cas particulier où une contrainte induite n'implique pas la valeur sur laquelle elle est censée porter (après une projection par exemple), on peut immédiatement considérer cette contrainte comme le résultat de l'union : son affectation ne concerne pas la variable x, sa valuation est au moins égale au minimum des valuations, sa justification est plus réduite que l'union de toutes les justifications.

A l'exception de ce cas particulier, toutes les contraintes utilisées sont nécessaires à l'obtention du résultat. Cependant, dans la mesure où l'union ne s'intéresse qu'au minimum des valuations, certaines contraintes induites utilisées peuvent être "surdimensionnées" en terme de valuation, donc de justification. Les conditions idéales sont remplies lorsque toutes les contraintes utilisées ont la même valuation.

Cette situation peut être forcée grâce à la propriété de réduction 3.9. Le plus efficace reste néanmoins de créer, si possible, des contraintes "sur mesure" : lorsque **FC-induite**

réalise une union, il n'est pas nécessaire que **BT-induite** s'efforce de produire une valuation maximale, la valuation correspondant à l'union suffit. Cette stratégie est efficace pour réduire les justifications, bien qu'elle se mette difficilement en œuvre puisque le minimum des valuations n'est connu qu'*a posteriori*.

Notion de surcoût

A ces réflexions générales concernant la génération des contraintes induites et de leurs justifications, il faut ajouter des remarques concernant la gestion des surcoûts particulière aux procédures de prévision étendues (**EFC-Evalue**, **DAC-Evalue**). Comme nous l'avons vu, les agrégations partielles sont désormais inutiles voire néfastes car les disjonctions ne sont plus assurées et le résultat peut s'avérer sous-optimal.

Malheureusement, l'exploration de chaque variable nécessite une union (**FC-induite**) et chaque union constitue une agrégation partielle des contraintes relatives à l'extension considérée. Le résultat de l'agrégation des ces agrégations partielles que produit **EFC-induite** est nécessairement sous-optimal. On souhaiterait du moins obtenir une valuation au moins égale à celle de **EFC-Evalue**. Elle ne sera obtenue dans le cas général qu'à condition de considérer séparément et en premier lieu les contraintes induites directes.

Ces remarques quelque peu négatives conservent leur pertinence pour tous les algorithmes de prévision. Cependant, lorsque les problèmes ne comptent que des contraintes d'origine, les versions qui utilisent $\odot$ et l'union sont parfaitement équivalentes, en termes de valuations, à leur homologues s'appuyant sur $\oplus$ et min.

3.2.3 Définitions

La définition des contrainte induites doit permettre de développer des méthodes de prévision plus puissantes que la 1-cohérence du forward-checking ou la 2-cohérence partielle du Directed Arc-consistency Counts. Commençons par établir rigoureusement la définition de la k-cohérence dans le cadre des CSP valués[BFM+96].

Définition 3.21 (k-cohérence) *Un VCSP induit est dit* k-cohérent *si pour toute affectation partielle A de cardinal $k-1$ et pour toute variable $x \notin \tilde{A}$, $v_{\odot}(A) \succeq \min_i v_{\odot}(A \cup \{x = i\})$*

Dans le cadre classique, il faut qu'une affectation de k variables localement incohérente soit décelée dès l'affectation de $k-1$ variables. Dans le cadre valué, la valuation locale de l'affectation doit être prévisible lorsque que $k-1$ variables seulement sont affectées.

3.2.4 Algorithme

Dans la mesure où nous disposons maintenant d'un opérateur d'agrégation idempotent, il nous suffit de copier les algorithmes existants dans le cadre de la satisfaction. Nous présentons ici une version triviale destinée principalement à illustrer le principe du renforcement de la k-cohérence dans le cadre valué. La méthode est simple : tant qu'une inégalité n'est pas respectée, l'algorithme ajoute une contrainte induite pour compenser la différence.

Cet algorithme ne fait pas, comme le forward-checking, de distinction entre les deux options, simple ou étendue, de cohérence locale. Et il n'y a pas lieu d'en faire car, contrairement au forward-checking, les contraintes induites créées sont immédiatement introduites dans le problème et participeront automatiquement aux agrégations suivantes.

Il existe néanmoins diverses manière de satisfaire la propriété. Celle que nous proposons génère beaucoup de contraintes induites plutôt que d'effectuer des agrégation partielles non imposées par les unions. L'autre possibilité consiste à créer une contrainte par affectation examinée (à l'aide de **EFC-induite**) plutôt qu'une contrainte pour l'examen de chaque extension de l'affectation (comme nous le faisons avec **FC-induite**) ; **EFC-induite** renvoie l'agrégation des résultats des appels à **FC-induite**. On préférera l'une ou l'autre des versions selon le nombre de contraintes souhaité et les risques de noyade.

K-C(A)

- Soit $\mathcal{A}_{k-1}$ l'ensemble de toutes les affectations A' de cardinal $k-1$ telles que $\bar{A} \cap \bar{A}' = \emptyset$.
- Pour chaque $A' \in \mathcal{A}_{k-1}$,
  - Pour chaque $x \notin \bar{A} \cup \bar{A}'$
    1. Soit $\zeta_x =$**FC-induite**$(A \cup A', x)$
    2. si $\phi(\zeta_x) \succ v_\odot(A \cup A')$, alors ajouter ζ_x au problème
- Si une contrainte a été ajoutée recommencer la boucle

Le test de la ligne 2 correspond à la propriété de k-cohérence. Par conséquent, lorsque l'algorithme n'effectue plus de modification, le VCSP est un VCSP induit k-cohérent.

Terminaison :

En ce qui concerne la terminaison, il faut noter que :

- Lorsque des contraintes sont ajoutées pour interdire une affectation A, leur valuation est strictement supérieure à l'ancienne valuation de A et par conséquent strictement supérieure à la valuation de toute ancienne contrainte interdisant A (d'après la monotonie de l'agrégation).
- Les valuations des contraintes induites proviennent de la somme des valuations d'un ensemble de contraintes d'origine. Il n'existe qu'un nombre fini de tels ensembles et les contraintes induites ne peuvent donc prendre qu'un nombre fini de valeurs : la procédure termine toujours.

Unicité :

L'obtention de la k-cohérence ne permet pas d'assurer l'unicité du VCSP résultant. Nous avons en effet insisté sur le rôle des justifications dans les agrégations et sur l'impossibilité de maîtriser complètement les noyades. Selon l'ordre des examens et l'ordre des agrégations, les contraintes induites produites peuvent varier en nombre et en nature.

Complexité :

Dans le cadre classique, de nombreuses variantes de l'arc-cohérence s'affrontent sur le critère de complexité en pire cas. Bien que notre algorithme vise essentiellement la clarté et non l'efficacité, nous en donnons la complexité.

Si l'on considère l'appel de **K-C**($\emptyset$), chaque boucle examine les $C_n^{(k-1)}$ affectation de $k-1$ variables parmis les n variables du problème. Puis, pour chaque affectation toutes les extentions possibles avec une variable sont examinées, soit $(n-k+1)$ examens. Chaque boucle effectue donc $C_n^{(k-1)}.(n-k+1)$ appel à **FC-induite** avec une affectation de cardinal $k-1$.

FC-induite effectue, au pire, d évaluations d'une affectation de cardinal k, où d est la taille maximale des domaines des variables. Soit $E(k)$ la complexité de l'évaluation d'une affectation de cardinal k, la complexité en pire cas d'une boucle du programme est en $o\left(C_n^{(k-1)}.(n-k+1).d.E(k)\right)$.

Dans le cas de l'arc-cohérence (k=2), on peut se ramener à une seule contrainte induite

par valeur (seule la dernière produite est nécessaire). L'évaluation consiste donc à agréger, en plus des contraintes d'origine, au maximum 2 contraintes induites. Soit c le nombre maximum de contraintes d'origine pesant sur une variable, il suffit d'au plus $2 \times (2.c)$ tests d'inclusion (appartenance des contraintes aux justifications) et $2.c + 2$ agrégations par $\oplus$. Si l'on suppose cette agrégation en $o(1)$, la complexité de l'évaluation est en $o(c)$.

Finalement la complexité en pire cas d'une boucle de l'algorithme d'arc-cohérence est en $o(n^2.d.c)$. Le nombre de boucles succeptibles d'être effectuées est limité par le nombre fini de valeurs que les valuations des contraintes induites peuvent prendre. Soit v ce nombre, puisque chaque boucle augmente la valuation d'une contrainte induite, au pire il faudra en effectuer $n.d.v$. Cette estimation en pire cas est extrêmement pessimiste et donne une complexité globale en $o(n^3.d^2.c.v)$.

Exemple :

Nous donnons sur l'exemple habituel les contraintes induites produites par un algorithme d'arc-cohérence valuée. Ces contraintes sont donc unaires. La valuation produite par **DAC-evalue** est donnée pour comparaison (colonne "DAC"). Les tableaux 3.1 et 3.2 donnent la valuation et la justification des contraintes produites lorsque les variables sont examinées dans l'ordre $(X_1 ... X_5)$ et dans l'ordre inverse $(X_5 ... X_1$, pour illustrer les risques de noyade), ainsi que la valuation induite maximale obtenue pour chaque valeur (colonne "Arc").

> *Suivons dans le cas inverse l'examen de X_3^b. D'abord, on examine ses extensions avec X_5 : avec X_5^a, C_4 est violée, ainsi que les deux contraintes induites ζ_1 et ζ_2. L'agrégation de ces trois contraintes correspond à la contrainte $(\{X_3^b, X_5^a\}, 2, \{C_4, C_5\})$. Avec X_5^b, seuleC_4 est violée, ce qui correspond à la contrainte $(\{X_3^b, X_5^b\}, 1, \{C_4\})$. L'union de ces deux contraintes donne ζ_6 (on notera au passage l'intervention inutile de C_5 dans la justification).*
>
> *L'examen de l'extension avec X_4^a donne trois violations : C_3, ζ_3 et ζ_6 précédemment construite, donc une agrégation $(\{X_3^b, X_4^a\}, 2, \{C_3, C_4, C_5\})$. L'extension avec X_4^b viole C_3, ζ_4 et ζ_6 d'où la contrainte induite $(\{X_3^b, X_4^b\}, 2, \{C_3, C_4, C_5\})$. L'union de ces deux contraintes, ζ_7 est mémorisée car sa valuation est supérieure à $v(\{X_3^b\})$ actuellement donnée par ζ_6.*

Les valuations produites sont identiques selon les deux ordres pour l'arc-cohérence, mais non pour l'algorithme **DAC** : l'arc-cohérence n'est pas dépendante d'un ordre qui limite ses évaluations. Par ailleurs, l'arc-cohérence fournit des contraintes plus intéressantes. En particulier, il ressort du renforcement que la solution optimale possède une valuation supérieure ou égale à 1. Cependant, d'un ordre à l'autre, les phénomènes de

Affectation		Extension testée X_1	X_2	X_3	X_4	X_5	*Arc*	*DAC*
X_1	a	–		$(1,\{C_1\})$			1	0
	b	–					0	0
	c	–					0	0
X_2	a		–	$(1,\{C_2\})$			1	0
	b		–				0	0
	c		–				0	0
X_3	a			–			0	0
	b			–	$(1,\{C_3\})$	$(2,\{C_3,C_4\})$	2	2
X_4	a			$(1,\{C_3,C_4\})$	–		1	1
	b				–	$(1,\{C_5\})$	1	0
X_5	a			$(1,\{C_3,C_4\})$	$(2,C_3,C_4,C_5)$	–	2	2
	b				$(1,\{C_3,C_4,C_5\})$	–	1	0
X_1	a	–			$(2,\{C_1,C_3,C_4,C_5\})$		2	0
	b	–			$(1,\{C_3,C_4,C_5\})$		1	0
	c	–			$(1,\{C_3,C_4,C_5\})$		1	0
X_2	a	$(2,\{C_1,C_2,C_3,C_4,C_5\})$	–				2	0
	b	$(1,\{C_1,C_3,C_4,C_5\})$	–				1	0
	c	$(1,\{C_1,C_3,C_4,C_5\})$	–				1	0
X_3	a	$(1,\{C_1,C_3,C_4,C_5\})$		–			1	0

TAB. 3.1 – *Arc-cohérence valuée*

Affectation		Extension testée X_5	X_4	X_3	X_2	X_1	*Arc*	*DAC*
X_5	a	–	$\zeta_1 = (1, \{C_5\})$	$\zeta_2 = (2, \{C_4, C_5\})$			2	0
	b	–					0	0
X_4	a		–	$\zeta_3 = (1, \{C_3\})$			1	0
	b	$\zeta_4 = (1, \{C_5\})$	–				1	1
X_3	a		$\zeta_5 = (1, \{C_3, C_5\})$	–			1	0
	b	$\zeta_6 = (1, \{C_4, C_5\})$	$\zeta_7 = (2, \{C_3, C_4, C_5\})$	–			2	2
X_2	a		$\zeta_8 = (1, \{C_3, C_5\})$	$\zeta_9 = (2, \{C_2, C_3, C_4, C_5\})$	–		2	1
	b		$\zeta_{10} = (1, \{C_3, C_5\})$		–		1	0
	c		$\zeta_{11} = (1, \{C_3, C_5\})$		–		1	0
X_1	a		$\zeta_{12} = (1, \{C_3, C_5\})$	$\zeta_{13} = (2, \{C_1, C_3, C_4, C_5\})$		–	2	1
	b		$\zeta_{14} = (1, \{C_3, C_5\})$			–	1	0
	c		$\zeta_{15} = (1, \{C_3, C_5\})$			–	1	0
X_5	b	–	$\zeta_{16} = (1, \{C_3, C_5\})$				1	0

TAB. 3.2 – *Arc-cohérence valuée, ordre inverse*

noyade n'ont pas la même ampleur et illustrent la dépendance de l'agrégation aux choix d'agrégations partielles, donc à l'ordre des évaluations. En particulier, l'ordre direct produit des contraintes induites aux justifications nettement trop surchargées. Exemples :

- la contrainte induite posée sur X_4^a lors de l'évaluation de l'extension avec X_3 pourrait être plus pertinente. Une analyse rudimentaire montre que la contrainte C_3 suffit à justifier la valuation ;
- la contrainte induite posée lors de la seconde passe sur X_2^b est liée au fait que l'examen de l'extension avec X_1 est effectué en premier. Chaque examen est une agrégation partielle, et l'utilisation nécessaire de l'union interdit de les effectuer simultanément : un ordre, avec ses conséquences, est inévitable.

Cet exemple illustre aussi l'intérêt d'utiliser le renforcement de k-cohérence en combinaison avec les cohérences d'ordre inférieur. A la suite de l'examen de l'arc cohérence pour la variable X_4, un forward-checking sur cette valeur produirait une contrainte induite $(\emptyset, 1, \{C_3, C_5\})$. Cette contrainte pertinente produite une fois pour toutes éviterait la duplication implicite de l'information lors de chaque test d'extension avec X_4.

Nous donnons comme d'habitude l'arbre parcouru sur notre exemple (figure 3.2). Les

contraintes induites produites par un prétraitement avec l'arc-cohérence dans l'ordre inverse sont utilisées par un algorithme de Branch & Bound. Seul un ensemble minimal de contraintes induites suffisant pour apporter les coupes dans l'arbre est représenté. Pour comparaison, la figure donne aussi le parcours obtenu par la combinaison du Forward-checking avec le prétraitement effectué par DAC.

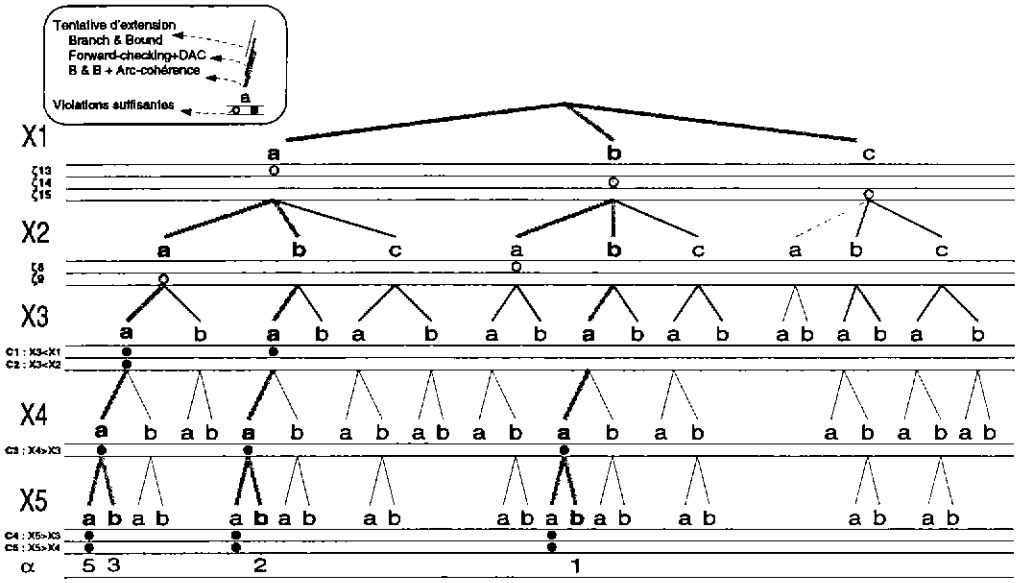

FIG. 3.2 – *Arc-cohérence valuée.*

Sur ce petit exemple, il semble difficile de faire mieux. Il ne faut cependant pas perdre de vue les calculs importants effectués par le prétraitement ainsi que lors de la recherche, pour agréger les nombreuses contraintes induites.

3.2.5 Expérimentations

Jeux d'essai

Les jeux d'essai sont constitués par des VCSP additifs binaires aléatoires. Le comportement des algorithmes de recherche est généralement très différent sur les problèmes aléatoires et sur les problèmes réels. Cependant, la génération aléatoire permet de produire facilement des jeux d'essai homogènes en fonction de paramètres dimensionnants simples [Smi94]. Ceci est indispensable si l'on souhaite tirer des conclusions générales sur le comportement des algorithmes.

Les paramètres utilisés pour produire des instances de problèmes sont :

- le poids des contraintes que nous fixons toujours à 1 ;
- le nombre de variables n ;
- la taille de leurs domaines d ;
- la dureté commune de toutes les contraintes *dur*. La dureté d'une contrainte est le rapport du nombre de n-uplets interdits sur le nombre total de n-uplets possibles ;
- la connectivité du graphe de contraintes *con*, nombre de contraintes sur le nombre maximal de contraintes possibles.

Nous conserverons l'ensemble de ces notations pour toutes les expérimentations sur les VCSP aléatoires.

Nous devons par ailleurs tenir compte de la difficulté des problèmes générés. Trop faciles, il ne nécessitent pas la mise en œuvre d'algorithmes d'optimisation (un algorithme de satisfaction suffirait à l'obtention de solutions) et ne permettent donc pas d'illustrer les capacités d'optimisation des algorithmes. Trop difficiles, ils mettent en échec les algorithmes de recherche complets et interdisent leur évaluation quantitative.

Les jeux d'essai que nous utilisons sont constitués de problèmes intermédiaires en termes de difficulté. Or, celle-ci réside en grande partie dans le nombre de contraintes qu'il faut violer pour produire un échec. Elle est donc liée à la valuation des solutions optimales comparée aux valuations des contraintes. Dans le cadre classique, une formule simple donne l'espérance du nombre de solutions en fonction des paramètres caractéristiques d'un problème. En suivant la même démarche, nous nous intéressons à l'espérance du nombre de solutions d'une valuation donnée (figure 3.3). En fonction du nombre de variables et de la taille de leur domaine, nous fixons les deux autres paramètres (dureté et connectivité) de manière à ce que l'espérance de solutions de valuation 2 soit unitaire (le détail des calculs est donné en annexe). Cela signifie que les problèmes que nous générons possèdent un optimum probable entre 2 et 3.

Le nombre de variables est fixé à 20, et leur domaine à 5 valeurs. Pour chaque jeu de paramètres (connectivité / dureté), 100 instances aléatoires de problèmes sont générées.

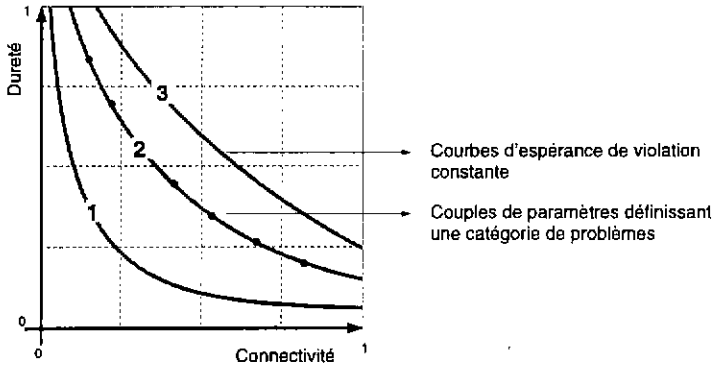

FIG. 3.3 – *Espérance du nombre de solutions*

Versions testées

Comme toujours lorsque l'on teste des algorithmes, la question du codage et de son optimalité se pose. Le code que nous utilisons[3] dans cette thèse est certainement loin d'être optimal et a plutôt vocation à permettre la cohabitation d'une grande variété de méthodes de résolution et la mise en œuvre de multiples options. La trame est assez proche des algorithmes décrits à titre pédagogique et offre une grande modularité aux dépens des possibilités d'optimisation particulières à chaque version. Tous les algorithmes testés présentent néanmoins l'avantage d'être issus d'une structure de données et d'un squelette de programmation communs, et en définitive d'un programme unique dont ils ne sont que des variantes. Les algorithmes standards ont aussi été reprogrammés et intégrés au schéma. Aussi les comparaisons faites entre les divers algorithmes sont-elles tout à fait valides.

Les versions de l'arc-cohérence valuée que nous utilisons prennent en compte la majorité des remarques émises juqu'ici :

- pour des questions de rapidité de calcul, l'opérateur d'agrégation est utilisé comme un opérateur binaire pour agréger progressivement et une par une les contraintes induites, dans l'ordre où elles se présentent. Une variante économe consistera à ne considérer, pour chaque valeur[4], que la dernière contrainte induite créée ;
- pendant le renforcement de l'arc-cohérence, après chaque examen d'une variable, l'union des contraintes posées sur ses valeurs est effectuée (procédure **FC-induite**) ;

3. Le langage de programmation est C++ ; les ordinateurs utilisés sont des stations SUN Sparc 4.
4. L'arc-cohérence crée des contraintes induites unaires.

- les contraintes d'origine sont agrégées en premier puis ensuite les autres contraintes (éventuellement dans l'ordre décroissant des rapports valuation sur taille de justification). Lorsqu'un forward-checking est utilisé, une première borne inférieure est fournie par l'examen des contraintes d'origine seules ;
- la justification des agrégations est limitée autant que possible. Lorsque l'agrégation ne fournit pas une valuation supérieure à celle des contraintes agrégées, on considère que le résultat correspond à la contrainte de plus forte valuation (et, à valuation égale, celle de plus petite justification).

Nous comparons plusieurs algorithmes :

- "**DAC+FC**" : prétraitement par Directed Arc-consistency Counts suivi du Forward-checking valué ;
- "**AC+FC**" : prétraitement par renforcement de l'arc-cohérence suivi du Forward-checking valué ;
- "**AC+AC**" : maintien permanent d'un VCSP arc-cohérent par renforcement après chaque affectation.

L'ordre d'examen des variables correspond toujours à l'ordre statique. Lorsque l'option consistant à ne considérer que la dernière contraintes induite produite sur chaque valeur est utilisée, le nom de l'algorithme est suivi du signe "-".

Résultats

Les résultats donnent des informations sur le temps de résolution et la taille de l'espace de recherche exploré au moyen du nombre de nœuds produits par la recherche. Pour chaque problème généré, les performances relatives[5] avec le Forward-checking des différentes versions proposées sont calculés. La figure 3.4 contient les moyennes de ces valeurs pour chaque catégorie de problème. Le nombre de contraintes induites produites est aussi donné. Les données concernant la version "AC+AC" ne font apparaître que les résultats sur le sous-ensemble des problèmes résolus par le Forward-checking en moins de 5 secondes (environ 25% des problèmes). En raison des faibles performances du maintien d'arc-cohérence, il est difficile de procéder à des essais plus complets. Les résultats ne font pas apparaitre les temps de résolution de cette version qui sont à peu près 100 fois plus longs que ceux de "FC"!

5. $\frac{version_x}{forward_checking}$

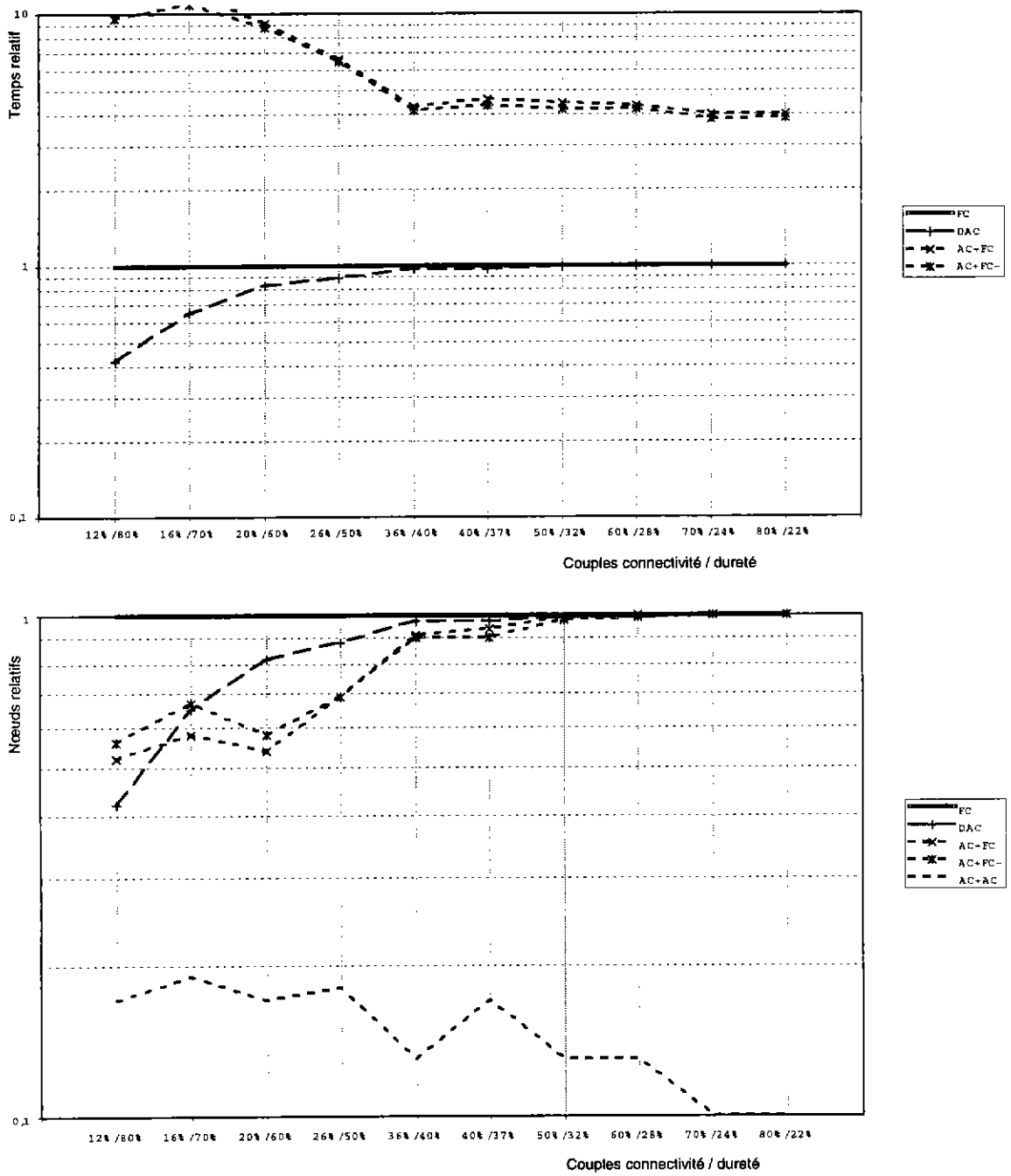

FIG. 3.4 – *Arc-cohérence valuée : résultats*

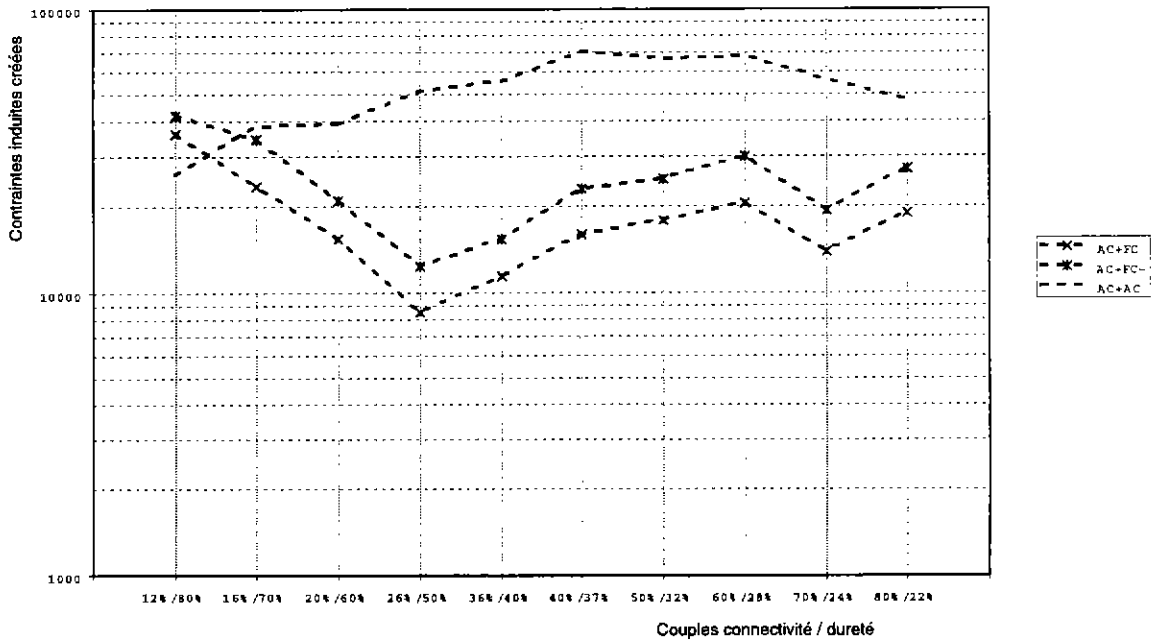

FIG. 3.5 – *Arc-cohérence valuée : résultats*

Au vu de ces quelques résultats, la première constatation qui s'impose est la lenteur du mécanisme utilisé. Malgré les restrictions, parfois importantes, de l'espace exploré, les mécanismes d'arc-cohérence s'avèrent au moins 4 fois plus lents. La faute ne revient pas seulement au mode de production des contraintes induites (particulièrement rustique) car même les versions n'utilisant l'arc-cohérence qu'en prétraitement sont pénalisées. La gestion et, en particulier, l'agrégation des contraintes induites semblent être la cause majeure des lenteurs. La perte de rapidité, lorsque les problèmes donnent lieu à une utilisation efficace des contraintes induites efficace est à ce titre significative. Sans qu'elle n'apportent aucune réduction de l'espace de recherche, les contraintes induites rendent les algorithmes 4 fois plus lent; ils peuvent être jusqu'à 10 fois plus lents lorsque les contraintes induites interviennent réellement. Les multiples simplifications auxquelles nous avons procédé, jusqu'à la réduction des contraintes induites à une par valeur, ne suffisent pas à produire une rapidité de calcul raisonnable. Le code que nous avons utilisé est certes loin d'être optimal (il est très proche du schéma proposé), mais il semble difficile de combler un tel retard.

En ce qui concerne l'efficacité de la réduction de l'espace exploré, il est possible d'atteindre des améliorations intéressantes avec un simple prétraitement. Les gains réalisés sont directement liés au rapport connectivité/dureté. Il est naturel que des contraintes

trop faciles à satisfaire ne permettent pas de produire des contraintes induites par le simple examen des paires de variables. Sur ce point, la sensibilité de l'algorithme **DAC** est la même. On remarquera cependant que l'arc-cohérence en prétraitement est généralement plus efficace, et se montre performante sur une plus large gamme de problèmes. Enfin, les performances du maintien d'arc-cohérence sont remarquables et d'une grande constance sur le spectre des problèmes. La lenteur du mécanisme ne permet malheureusement pas de prendre cette possibilité au sérieux.

Il faut enfin, pour que les algorithmes soient comparés sur un pied d'égalité, insister sur le fait que **DAC** nécessite un ordre total et ne peut donc pas bénéficier de l'apport extrêmement important des heuristiques dynamiques de choix des variables (la supériorité de **DAC** par rapport au Forward-checking est plus marquée lorsque les heuristiques dynamiques sont mises de côté).

Globalement, on peut dire que les contraintes induites sont efficaces pour anticiper les échecs, mais que leur gestion est trop lourde pour apporter un gain en temps de résolution. Plus précisément, la comparaison de **DAC** et de **AC+FC** montre que la source de l'inefficacité vient de la gestion permanente des redondances pendant la résolution.

La notion de CSP induits s'étend parfaitement au cadre valué, et avec elle celle de k-cohérence. Mais elle entraîne une trop grande complexité de la recherche. Outre la difficulté qu'il y a à contrôler les contraintes induites produites, la lenteur de l'agrégation, ou son inefficacité, suffisent à annihiler l'utilité du renforcement de k-cohérence. Il semble plus réaliste de se contenter d'une utilisation limitée, contrôlée et opportuniste des contraintes induites. Le Nogood Recording que nous présentons ci-dessous est une trame suffisamment générale et ouverte pour permettre une utilisation plus efficace des contraintes induites.

3.3 Nogood Recording pour VCSP

Le Nogood Recording dans le cadre valué [DV96, Dag96] est identique en tout point à celui utilisé pour les CSP classiques à la différence qu'il dispose d'un plus grand nombre de variantes : il fallait précédemment choisir les nogoods à mémoriser ; maintenant, il faut aussi déterminer l'éventuelle réduction des nogoods avant mémorisation, et le mode d'aug-

mentation ou d'agrégation lors de leur utilisation. Nous ne proposons ici qu'une possibilité d'utilisation des contraintes induites ou nogoods (dans ce contexte, nous parlerons plutôt de nogoods) pour bien fixer les notions importantes.

Pour le premier point, l'introduction d'une valuation et l'importance de la justification rendent difficile la sélection des nogoods "utiles". Il faut arbitrer entre les trois critères reconnus :

- avec une affectation de petite taille, le nogood est plus réutilisable ;
- avec une valuation élevée, les coupures dans l'arbre de recherche sont plus efficaces ;
- avec une justification de petite taille, les augmentations et agrégations donnent des valuations plus intéressantes. Pour un fonctionnement dynamique, une petite justification est aussi un gage de survie aux relaxations du problème.

Le choix est difficile, mais nécessaire, si l'on veux rester dans des limites raisonnables d'utilisation d'espace mémoire. Pour illustrer le principe du Nogood Recording, nous considérons que tous les nogoods non triviaux sont mémorisés. Un nogood $(A, v, \mathcal{C})$ est trivial si $v_{\mathcal{C}}(A) \succeq v$.

Le second point portant sur la réduction des nogoods peut être résolu par les développements de la section précédente. Ils permettent de jouer sur la taille et la composition de la justification, et par conséquent sur la taille et la composition de l'affectation. Les possibilités sont multiples et il est difficile de juger de leur intérêt dans le cas général. Cependant, la réduction naturelle d'un nogood par toutes les contraintes que son affectation viole apparaît comme une réduction minimale toujours avantageuse. En effet, les propriétés 3.17 ou 3.11 montrent qu'il suffit, lors de l'utilisation du nogood, de l'agréger — ou de l'augmenter — avec les contraintes violées pour que le nogood réduit apporte une valuation au moins équivalente à celle du nogood original. Notre algorithme se limitera à cette seule réduction (suivie d'une projection).

Le troisième point concerne l'utilisation des nogoods. Lorsque l'on cherche à évaluer une affectation, on dispose d'un ensemble de contraintes violées, induites ou non. La méthode la plus sûre consiste à agréger toutes ces contraintes comme nous l'avons vu dans la section 3.3. Nous optons pour une solution intermédiaire agrégeant chaque contrainte induite avec toutes les contraintes d'origine (propriété d'augmentation 3.10). On s'intéresse alors à la valuation prévue la plus élevée.

3.3.1 Algorithme

Nous donnons ici une trame d'algorithme de backtrack amélioré par des mémorisations : **NR**(A) tente d'étendre l'affectation A et renvoie un nogood justifiant l'éventuel échec (soit un nogood inclus dans A) ; **NR-induite**$(A \cup \{x = i\})$ évalue la valuation globale de l'extension de A avec $x = i$ et renvoie un nogood correspondant.

NR(A)

- Si A est complète alors $\alpha \leftarrow v(A)$
- Sinon
  - Soit $x \notin \bar{A}$, $N = \emptyset$
  - Pour chaque valeur i de x,
    - $n \leftarrow$**NR-induite**$(A \cup \{x = i\})$
    - Si $\phi(n) \prec \alpha$ alors $n \leftarrow$**NR**$(A \cup \{x = i\})$
    - (Si $x^i \notin n$ alors renvoie n : backjumping)
    - $N \leftarrow N \cup \{n\}$
  - $n_{union} \leftarrow \bigvee_{n \in N} n$
  - Mémorise $\Downarrow (n_{union})$ et $\bigvee_{n \in N} (\Downarrow (n))$
  - renvoie $\bigvee_{n \in N} n$

NR-induite(A)

- Si $\phi($**BT-induite**$(A)) \succeq \alpha$
  Renvoie **BT-induite**(A)
- Sinon, soit $N_A = \{\Uparrow^A (n), A \text{ viole } n\}$
  Renvoie un nogood de N_A de valuation maximale

Il faut remarquer que le choix a été fait, lors de la mémorisation, de considérer les deux possibilités consistant à réduire les nogoods avant[6] ou après l'union. La différence est que le premier nogood mémorisé n'inclut aucune contrainte violée par A, ni par aucune de ses extensions avec les valeurs x^i de la variable en cours d'exploration ; alors que le second n'exclut que les contraintes violées par A. Le premier nogood est donc plus réduit. Le revers de la médaille est bien sûr la possible diminution de sa valuation.

Remarque 3.1 *Les nogoods produits par* **BT-induite** *ne méritent pas d'être mémorisés puisqu'ils sont redondants avec la définition courante du problème.*

6. Réduire avant l'union devient naturel si l'on choisit de justifier les échecs par des nogoods systématiquement réduits.

L'algorithme proposé tient automatiquement compte de cette remarque puisque les mémorisations interviennent toujours après une union. On ne mémorise jamais de nogoods triviaux obtenus par évaluation, mais toujours des nogoods induits obtenus par exploration.

Remarque 3.2 *Les nogoods produits dont l'affectation est égale à l'affectation courante peuvent aussi être ignorés puisque le parcours arborescent ne reproduit jamais deux fois la même affectation.*

L'hybridation de cette technique avec les méthodes de prévision s'obtient simplement par le remplacement dans **NR-induite** de **BT-induite** par toute autre procédure de prévision (**FC-induite**, **EFC-induite**, etc.).

Exemple :

La figure 3.7 donne le fonctionnement du Nogood Recording valué sur notre exemple. Afin d'illustrer l'intérêt des réductions, la trace du même algorithme, sans réduction avant mémorisation, est donnée par la figure 3.6. Seuls les nogoods mémorisés non triviaux sont représentés (cf les remarques 3.1 et 3.2) ; ceux qui seront réutilisés sont en gras.

L'exploration se déroule comme pour le nogood recording classique ; les échecs sont traduits en nogoods et, lorsque toutes les valeurs d'une variable échouent, l'union des nogoods traduisant les échecs fournit l'explication du rejet de l'affectation courante.

L'exemple devient intéressant lors de l'exploration de l'affectation $\{X_1^a, X_2^a\}$. *Son extension avec* X_3^a *et* X_3^b *conduit aux nogoods*

$\zeta_{\mathbf{a}} = (\{\mathbf{X_1^a}, \mathbf{X_2^a}, \mathbf{X_3^a}\}, \mathbf{3}, \{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_3}, \mathbf{C_4}, \mathbf{C_5}\})$ *et*

$\zeta_{\mathbf{b}} = (\{\mathbf{X_1^a}, \mathbf{X_2^a}, \mathbf{X_3^b}\}, \mathbf{3}, \{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_3}\})$. *La procédure d'exploration retourne leur union :*

$\zeta = \bigvee(\zeta_{\mathbf{a}}, \zeta_{\mathbf{b}}) = (\{\mathbf{X_1^a}, \mathbf{X_2^a}\}, \mathbf{3}, \{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_3}, \mathbf{C_4}, \mathbf{C_5}\})$.

Ce nogood est un minimum pertinent puisqu'il ne se déduit pas directement de la définition du problème. Cependant, la projection ne permet pas de réduire son affectation qui est égale à l'affectation courante. Il ne doit pas être mémorisé puisque le parcours arborescent ne reproduira jamais son affectation.

Sa réduction naturelle est elle aussi sans intérêt puisque l'affectation du nogood ne viole aucune contrainte. Par contre, la réduction de ζ_a *et* ζ_b *est intéressante et permet le retrait de* C_1 *et* C_2, *donc celui de* X_1 *et* X_2. *Leur union conduit au nogood extrêmement précieux :*

$\zeta' = \bigvee(\Downarrow \zeta_{\mathbf{a}}, \Downarrow \zeta_{\mathbf{b}})(\emptyset, \mathbf{1}, \{\mathbf{C_3}, \mathbf{C_4}, \mathbf{C_5}\})$. *Ce dernier est bien sûr mémorisé.*

Il sera utilisé régulièrement et augmenté par chaque violation de C_1 *ou* C_2 *(indépendamment des valeurs de* X_1 *et* X_2*).*

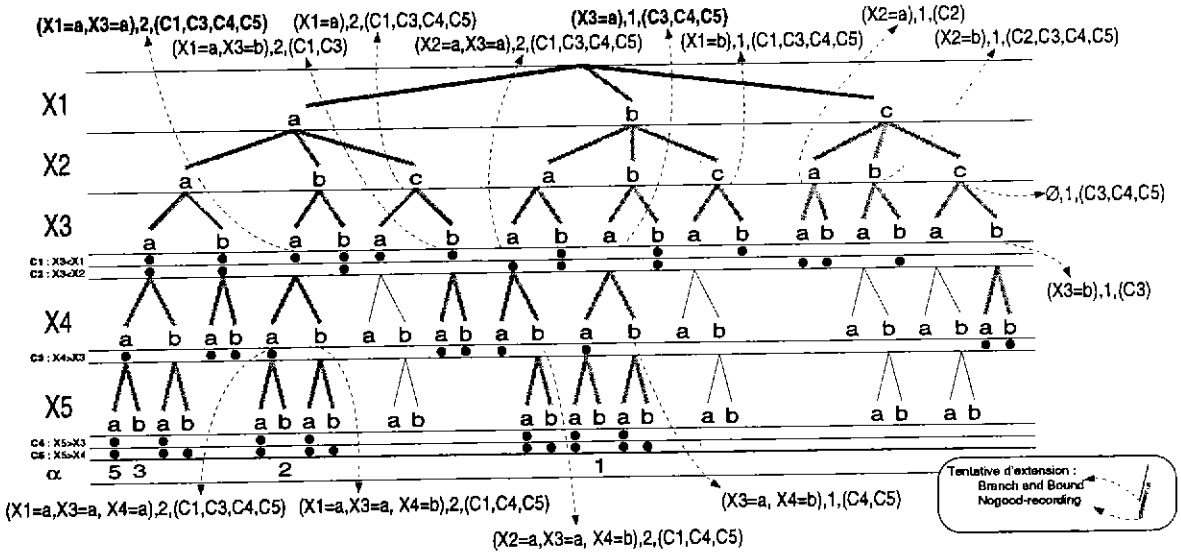

FIG. 3.6 – *Nogood-recording valué sans réduction.*

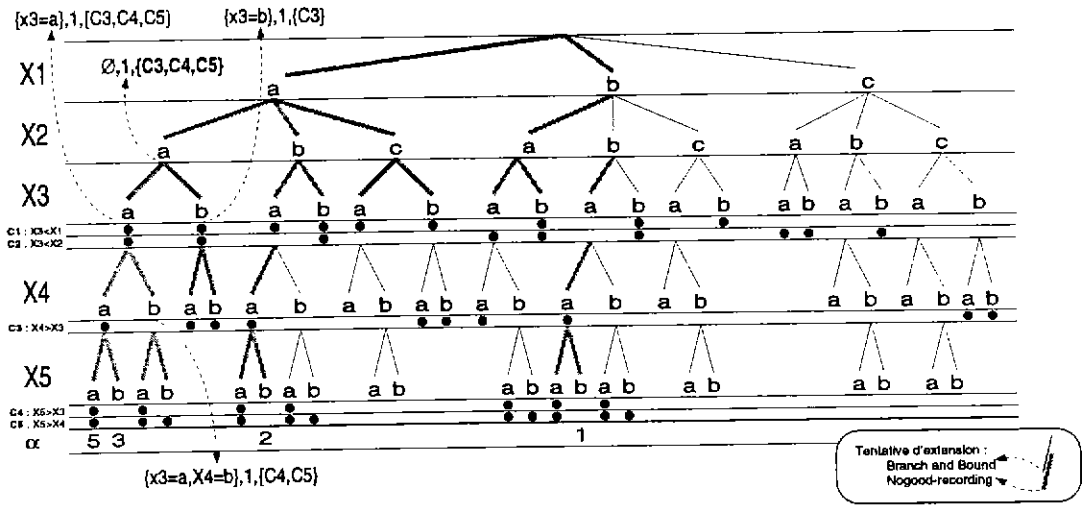

FIG. 3.7 – *Nogood-recording valué.*

3.3.2 Expérimentations

Versions testées

Nous reprenons les jeux d'essai précédents et comparons les versions "**NRx+FC**" du Nogood Recording valué mémorisant tous les nogoods pertinents d'affectation inférieure à x et s'aidant de **EFC-induite** pour prévoir les échecs.

Il est important de noter que, contrairement aux algorithmes d'arc-cohérence, le forward-checking, lorsqu'il est utilisé, ne tient compte que des contraintes d'origine. La valuation apportée par les nogoods et celle fournie par les prévisions sont ici complètement dissociées. Il faut aussi garder à l'esprit que les prévisions nuisent à l'intérêt de l'analyse des échecs et qu'il serait plus avantageux de comparer le Nogood Recording sans forward-checking au branch and bound.

Résultats

La figure 3.8 donne en valeurs relatives à celles du Forward-checking étendu les temps d'exécution et les nombres de nœuds développés. Apparaît aussi que le nombre de nogoods produits.

Comme précédemment pour l'arc-cohérence, on retrouve la préférence des contraintes induites pour les problèmes où la dureté des contraintes prime sur la connectivité du graphe de contraintes. Le Nogood Recording est à peu près deux fois plus lent que le Forward-checking pour parcourir le même nombre de nœuds. Ce rapport est suffisamment faible pour que cette technique constitue une alternative réaliste. Le cap d'une réduction de 50% est atteint pour les catégories de VCSP aléatoires les plus favorables. Le Nogood Recording apporte alors des gains substantiels sur le temps de résolution. Les lenteurs sont moins dues aux augmentations et réductions permanentes lors des évaluations qu'au mode de production des nogoods. Sans nogoods mémorisés, le Nogood Recording est presque deux fois plus lent que le Forward-checking, alors que ce même rapport entre les versions n'est obtenu que lorsque le nombre de nogoods mémorisés varie du simple au centuple.

Le problème de la sélection des nogoods pertinents est en partie résolu. La méthode proposée semble convenir puisque la version mémorisant le moins de nogoods (**NR2+FC**) est à peine moins, sinon plus efficace que les autres (en temps de résolution). Il est clair cependant que la faiblesse de la version mémorisant le plus de nogoods (**NR6+FC**) pro-

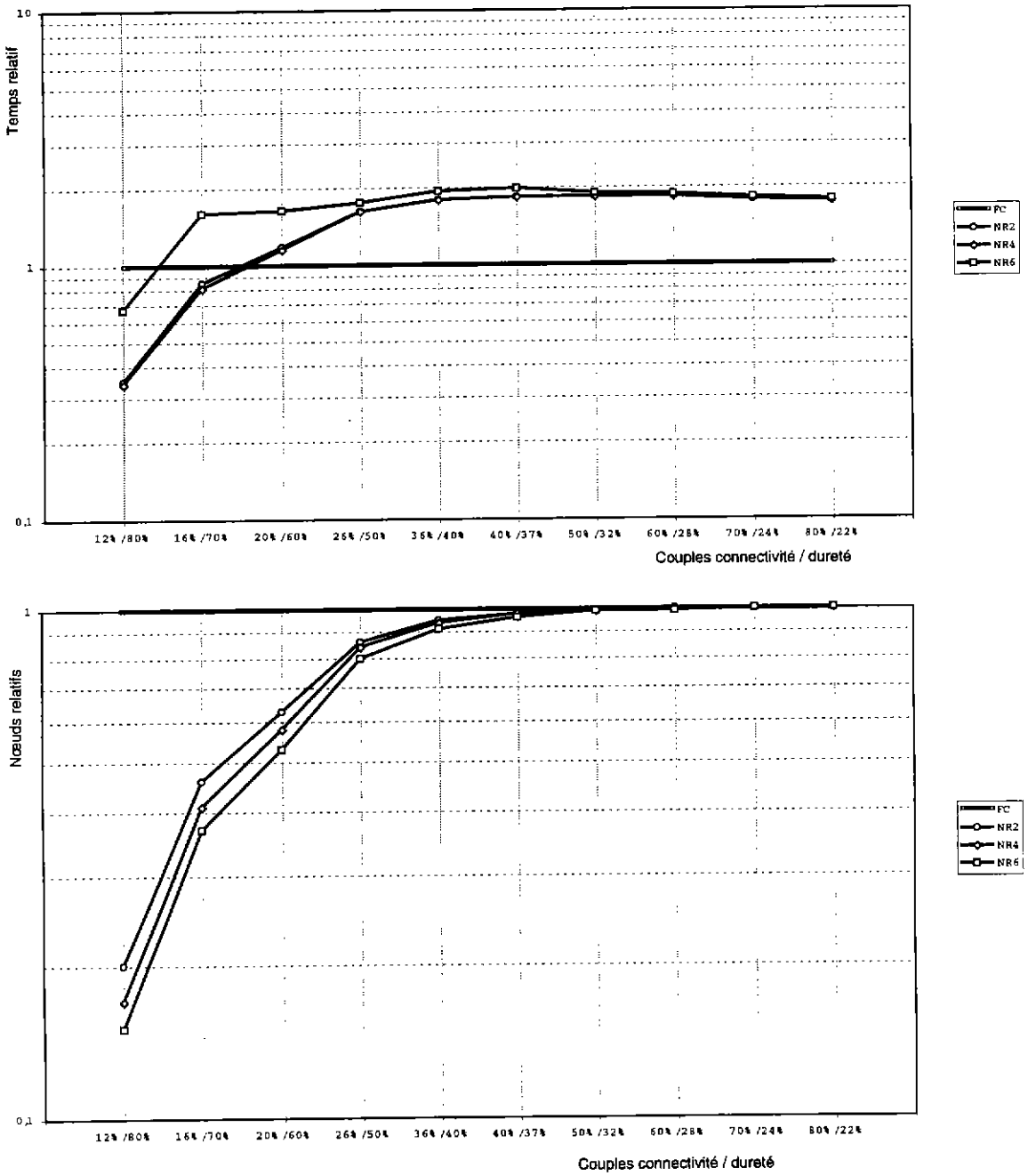

FIG. 3.8 – *Nogood Recording valué : résultats*

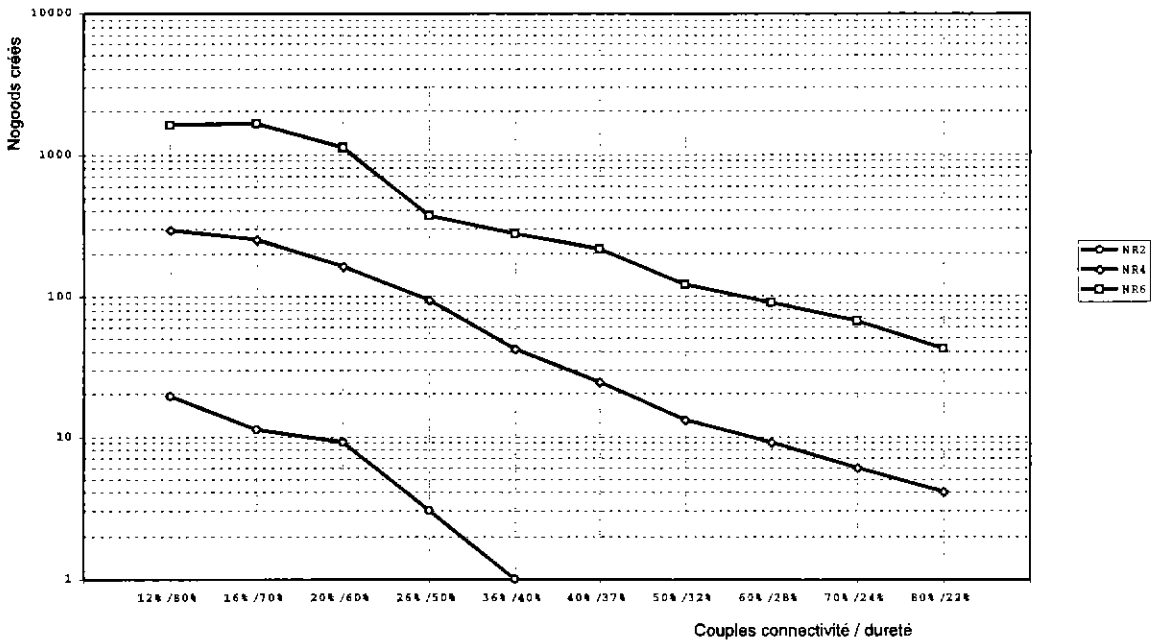

FIG. 3.9 – *Nogood Recording valué : résultats*

vient uniquement du surplus de calculs imposé par le nombre trop important de nogoods à tester. En effet, cette version induit une réduction sensible de l'espace de recherche par rapport aux autres versions. Comme il n'est pas certain que tous les nogoods parmi la multitude que produit **NR6+FC** soient nécessaires, la question d'une sélection plus pertinente reste ouverte.

Cette fois, la simplification de la recherche est suffisante pour ne pas nuire à la rapidité, et le mode de production paresseux des contraintes induites assure une réduction de l'arbre de recherche au moins équivalente à celle obtenue par un prétraitement à l'arc-cohérence. Il est alors possible d'atteindre une efficacité plus importante que celle obtenue par des moyens classiques.

3.4 Backtrack dynamique et VCSP

Les méthodes de résolution associées aux VCSP sont essentiellement de deux types : exhaustives ou locales. La rigueur des premières permet de trouver les solutions optimales et de prouver leur optimalité ; la souplesse des secondes autorise de nombreuses heuristiques assurant une convergence rapide vers de "bonnes" solutions.

Ces deux qualités sont antinomiques. Cependant, notre but est de construire un algorithme d'optimisation complet offrant une liberté maximale à la recherche. A cet égard, le Backtrack Dynamique [Gin93, GM94] s'avère très pertinent car, contrairement aux autres algorithmes complets de résolution des CSP, il peut s'affranchir partiellement de la rigidité du parcours arborescent.

Nous nous proposons d'étendre à la recherche de solutions optimales le Backtrack Dynamique [Dag97] (initialement défini dans le cadre de la satisfaction).

3.4.1 Algorithme

Un premier algorithme, calqué sur le modèle du Backtrack Dynamique présenté dans [Gin93], découle de la définition des nogoods valués. Il s'agit d'un branch and bound dont la borne inférieure repose sur des nogoods valués : à chaque valeur i de chaque variable x est associé un nogood N_{x^i} donnant un minorant de la valuation globale de l'extension d'une affectation A avec la valeur $x = i$ (ce nogood est une "mémoire" des anciennes tentatives infructueuses d'extension avec $x = i$).

La recherche maintient une affectation partielle courante A initialement vide, une borne supérieure α fournie par la dernière solution trouvée et fixée a priori à la valuation maximale, ainsi que l'ensemble des N_{x^i} initialisés au nogood nul ($n_\emptyset = (\emptyset, \bot, \emptyset)$). L'exploration utilise plusieurs fonctions :

- la procédure principale **DBT**(A) tente de développer une affectation partielle A ;
- en cas d'échec, cette dernière fait appel à la procédure de retour arrière **DBT-Retour**(A, n) avec un nogood n justifiant l'échec de A ;
- la fonction **DBT-induite**(A) a pour objet de retourner une estimation de la valuation globale de A basée uniquement sur les nogoods produits ;
- la fonction **Met-à-jour** met à jour les nogoods à l'occasion des retours. C'est la seule fonction qui modifie les nogoods.

L'algorithme est donné ci dessous :

DBT(A)

1. Si $\tilde{A} = \mathbb{V}$
   (a) Si $v(A) \prec \alpha$ alors $\alpha \leftarrow v(A)$
   (b) Soit $n =$**BT-induite**(A), **DBT-Retour**(A, n)
2. Sinon
   (a) Soit $x \in (\mathbb{V} - \tilde{A})$
   (b) S'il existe x^i telle que la valuation de **DBT-induite**$(A \cup \{x = i\}) \prec \alpha$ **DBT**$(A \cup \{x = i\})$
   (c) Sinon, soit $n = \bigvee_i$**DBT-induite**$(A \cup \{x = i\})$
   DBT-Retour(A, n)

DBT-induite(A)

renvoie un nogood de valuation maximale parmi les nogoods $\Uparrow^A (N_{x^i}), \forall x^i \in A$

DBT-Retour(A, n)

1. Si l'affectation de n est vide FIN
2. Sinon, soit $y^j \in A$
   (a) **Met-à-jour**(A, n, y^j)
   (b) $A \leftarrow A - \{y = j\}$
   (c) **DBT**(A)

Met-à-jour(A, n, y^j)

1. $\forall N_{z^k}$ dont l'affectation contient y^j, $N_{z^k} \leftarrow n_\emptyset$
2. $N_{y^j} \leftarrow \Downarrow_A (n)$

Les propriétés de mise en œuvre des nogoods assurent la validité des nogoods utilisés, mais il faut avant tout vérifier la propriété suivante :

Propriété 3.22 *A tout moment dans la procédure, l'affectation de chaque nogood N_{x^i} est incluse dans $A \cup \{x = i\}$.*

Démonstration : (3.22) *Cette propriété est vérifiée à l'initialisation. Par la suite, la production d'un nogood* N_{x^i} *dans* **Met-à-jour** *conserve la propriété :*

Si les nogoods respectent cette propriété, lorsque **DBT-induite**$(A \cup \{x = i\})$ *construit un nogood, ce-dernier concerne à l'évidence l'affectation* $A \cup \{x = i\}$. *Le nogood n justifiant le rejet de l'affectation courante A produit dans* **DBT**(A), *ligne 1.b ou 2.c, a son affectation toute entière incluse dans A. Quelle que soit la valeur* y^j *de A avec laquelle* **Met-à-jour**(A, n, y^j) *est appelée, le nogood* N_{y^j} *résultant respecte la propriété.*

*Enfin, lorsque la suppression d'une valeur de A (***DBT-Retour** *ligne 2.b) risque d'altérer la propriété, tous les nogoods concernés sont détruits au pas précédent par* **Met-à-jour**.

Validité

Nous vérifions ici que l'arrêt de la recherche (**DBT-Retour** ligne 3) n'intervient que si α est la valuation des solutions optimales du problème.

Il faut constater que le nogood n produit ligne 2.c dans **DBT** , comme tous les nogoods dont il est issu, possède une valuation supérieure ou égale à α. Il en est de même pour le nogood n produit lorsqu'une affectation complète est trouvée. **DBT-Retour** est toujours appelé avec un nogood de valuation supérieure ou égale à α. En cas d'arrêt, on dispose donc d'un nogood n d'affectation vide et de valuation supérieure ou égale à α. Ce nogood est la preuve que toute solution possède une valuation supérieure ou égale à α. Comme par ailleurs α provient de la valuation d'une solution, α est la valuation optimale du problème.

Terminaison

La terminaison est la propriété qui manque à la complétude de cet algorithme. Elle réside dans le choix de la valeur sélectionnée comme cible du backtrack (**DBT-Retour**, ligne 2).

Les nogoods produits par union ont toujours une valuation supérieure ou égale à α. Ils correspondent donc à des nogoods classiques interdisant une affectation. Pour assurer la terminaison, il suffirait d'appliquer la méthode du "Partial Order" de Mc Allester[McA93] à l'aide de ces nogoods. En deux mots, à chaque création d'un nogood, on impose des

conditions d'ordre entre les variables de l'affectation du nogood et la variable sur laquelle se porte le backtrack. Ceci a pour effet de développer progressivement un ordre partiel entre les variables. Si le choix du point de backtrack se fait de telle manière que les nouvelles conditions respectent l'ordre partiel construit jusque là, alors le processus termine (cf [McA93]).

Cependant, cette solution ne nous convient pas car elle n'exploite pas pleinement le caractère valué des nogoods. Notre objectif consiste à ne fonder l'ordre partiel que sur les nogoods réduits, afin principalement de réduire les contraintes imposées par l'ordre partiel.

Nous allons maintenant nous intéresser de plus près à ces conditions suffisantes sur le choix du point de backtrack pour assurer la terminaison en nous inspirant toujours de David Mc Allester : l'algorithme va construire et maintenir un ordre partiel sur les variables pour augmenter la liberté du backtrack (par rapport à l'ordre total usuel).

3.4.2 Terminaison

Ordre partiel

Nous définissons un ordre partiel $\leq_O$ sur les variables par la fermeture réflexive et transitive d'un ensemble O de conditions d'ordre du type $y \leq x$ entre deux variables. Plus exactement, cette fermeture est un ordre partiel si les conditions d'ordre ne présentent pas de cycles (autres que des boucles).

Nous associerons à chaque nogood $N_{x^i} = (A, v, \mathcal{C})$ des conditions d'ordre $y \leq x$ entre chaque variable y de A et x. Nous noterons $O(N_{x^i})$ l'ensemble des conditions d'ordre découlant de ce nogood.

Invariant

Construisons une valeur bornée strictement croissante liée à la progression de l'algorithme :

Définition 3.23 *La "valeur d'interdiction" associée à une variable x est donnée par la liste ordonnée des valuations des nogoods N_{x^i}.*

Soit $v1$ et $v2$ deux valeurs d'interdiction, $v1$ est supérieure ou égale à $v2$ si elles sont de longueurs égales et si tous les éléments de $v1$ sont supérieurs ou égaux à leur vis-à-vis dans $v2$.

Définition 3.24 *La valeur d'interdiction associée à un ordre total sur les variables est définie par la liste ordonnée des valeurs d'interdiction des variables.*

Soit $V1$ et $V2$ deux valeurs d'interdiction d'ordres totaux sur les variables, $V1$ est supérieure ou égale à $V2$ si tous les éléments de $V1$ sont supérieurs ou égaux à leur vis-à-vis dans $V2$.

La "valeur de terminaison" d'un problème en cours de résolution dépend d'un ordre partiel et de l'ensemble des nogoods N_{x^i} ; nous noterons $V(O,\mathcal{N})$ la valeur de terminaison du problème avec un ensemble de conditions O et un ensemble de nogoods $\mathcal{N}$. Elle se définit comme suit :

Définition 3.25 *La valeur de terminaison est l'ensemble des valeurs d'interdiction associées à chaque ordre total compatible avec l'ordre partiel courant.*

Soit $V1$ et $V2$ deux valeurs de terminaison, $V1$ est supérieure ou égale à $V2$ si chaque élément minimal de $V1$ est supérieur ou égal à au moins un élément de $V2$.

La propriété invariante que nous cherchons à vérifier est alors la suivante :

Propriété 3.26 (terminaison) *La valeur de terminaison est strictement croissante*

Les lemmes qui suivent permettront de prouver que chaque modification que l'algorithme impose aux nogoods ou à l'ordre partiel fait croître la valeur de terminaison.

Lemme 3.27 *Soit $\mathcal{N}$ un ensemble de nogoods, O et O' deux ensembles de conditions acycliques. Si $O \subset O'$ alors $V(O,\mathcal{N}) \leq V(O',\mathcal{N})$.*

Démonstration : (3.27) *Tout ordre total respectant $\leq'_O$ respecte $\leq_O$, donc tout ordre total respectant $\leq'_O$ possède un homologue de même valeur respectant $\leq_O$. Par suite, toute valeur de $V(O',\mathcal{N})$ est égale à une valeur de $V(O,\mathcal{N})$*

Lemme 3.28 *Soit O un ordre partiel, $\mathcal{N}$ un ensemble de nogoods et y une variable. Soit $\mathcal{N}'$ un ensemble de nogoods tel que, d'une part, la valeur associée à y augmente strictement et, d'autre part, seules les valeurs des variables z telles que $y \leq_O z$ soient modifiées. Alors $V(O,\mathcal{N})$ est strictement inférieure à $V(O,\mathcal{N}')$*

Démonstration : (3.28) *$V(O,\mathcal{N})$ et $V(O,\mathcal{N}')$ sont constitués des valeurs d'interdictions sur les ordres totaux compatibles avec $\leq_O$. Soit Ot l'un d'entre eux. Puisque la valeur de x augmente, alors que ne sont modifiées que les valeurs des variables nécessairement situées après x, la valeur d'interdiction de Ot calculée avec $\mathcal{N}$ est inférieure ou égale à celle calculée avec $\mathcal{N}'$. Tout élément de $V(O,\mathcal{N}')$ est donc supérieur à un élément de $V(O,\mathcal{N}')$; l'augmentation stricte de la valeur de x assure pour finir que les deux valeurs de terminaison ne sont pas égales.*

Il apparaît d'ores et déjà que, lors de chaque appel de **Met – à – jour**, l'augmentation de la valeur de terminaison dépendra des liens entre la variable visée par le backtrack et les destructions de nogoods qui en découlent. Une condition particulière doit cependant être satisfaite : l'augmentation stricte de la valeur de terminaison de la variable choisie. Pour vérifier l'augmentation stricte de la valuation du nogood N_{y^j} modifié dans **Met – à – jour**, une dernière manipulation des nogoods est nécessaire :

Définition 3.29 (Réduction partielle) *Soit A une affectation partielle et n un nogood portant sur A. Soit $C_{A\cap n}$ l'ensemble des contraintes violées par A et présentes dans la justification de n, et C_{A-n} les autres contraintes violées par A.*
Nous appelons réduction partielle d'un nogood par A et n l'augmentation (propriété 3.10) avec les contraintes de $C_{A\cap n}$, suivie de la réduction (propriété 3.9) avec les contraintes de C_{A-n} par A.

Nous noterons $\Downarrow_{(A,n)}$ cet opérateur de réduction. Il a pour effet de projeter un nogood n_2 de manière à ce que le résultat n' ait une valuation supérieure à son prédécesseur n.

Propriété 3.30 *Soit A une affectation partielle, n et n_2 deux nogoods, tous trois compatibles. Soit $n' = \Downarrow_{(A,n)} (n_2)$.*
La valuation de $\Uparrow^A (n')$ est supérieure ou égale à celle de n_2.
Si par ailleurs n_2 possède une valuation strictement supérieure à $\Uparrow^A (n)$, la valuation de n' est strictement supérieure à celle de n.

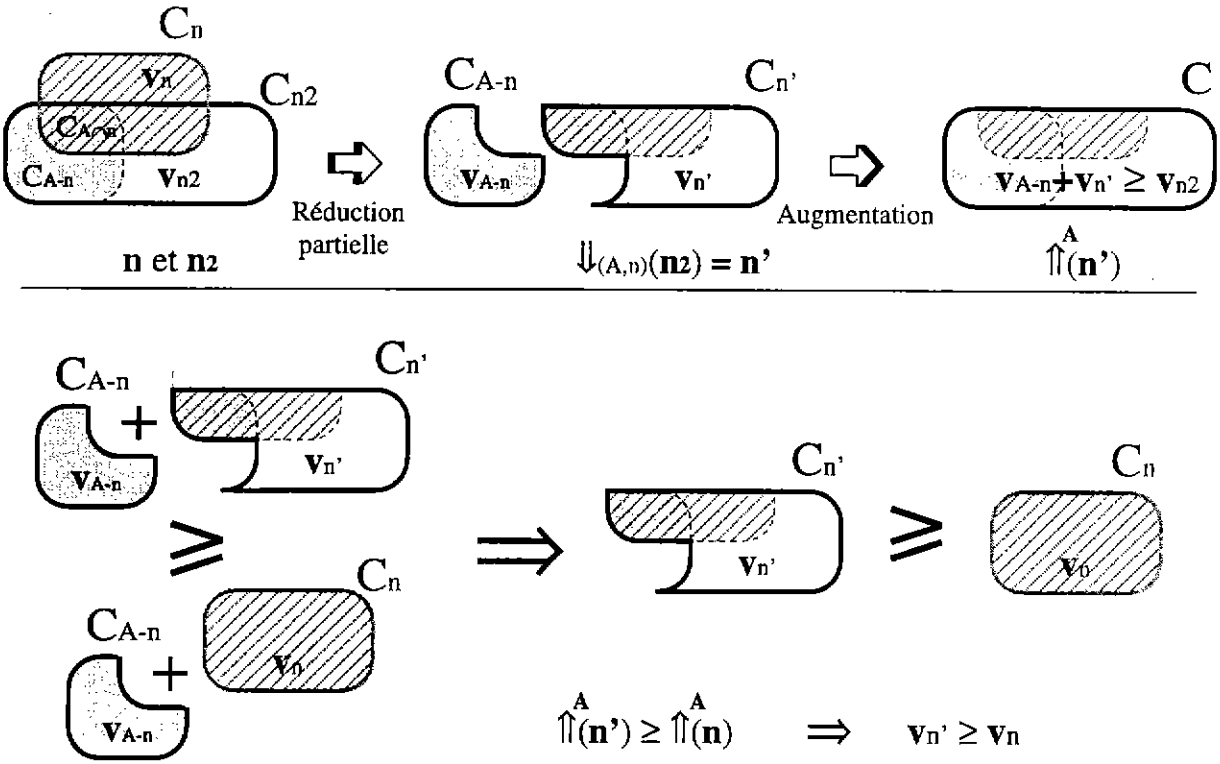

FIG. 3.10 – *Réduction partielle* $n' = \Downarrow_{A,n}(n_2)$

Démonstration : (3.30) *(figure 3.10) Soit* $\mathcal{C}_A$ *l'ensemble des contraintes violées par* A, $n = (A_n, v_n, \mathcal{C}_n)$, $n' = (A_{n'}, v_{n'}, \mathcal{C}_{n'})$ *et* $n_2 = (A_{n_2}, v_{n_2}, \mathcal{C}_{n_2})$.

Le premier point donnant la valuation de $\uparrow^A(n')$ *supérieure à celle de* n_2 *découle directement de la propriété d'équivalence 3.17.*

Pour le second, il faut d'abord noter que $\mathcal{C}_{n'} = (\mathcal{C}_{n_2} \cup \mathcal{C}_A) - (\mathcal{C}_A - \mathcal{C}_n)$. *Alors, soit* $v_{A-n} = \bigoplus_{c \in \mathcal{C}_A - \mathcal{C}_n} \phi(c)$, *le premier point indique que* $v_{n'} \oplus v_{A-n} \succeq v_2$.

Si, par hypothèse, v_2 *est strictement supérieure à la valuation de* $\uparrow^A(n)$ *alors* $v_{n'} \oplus v_{A-n} \succeq v_2 \succ v_n \oplus v_{A-n}$. *Donc* $v_{n'} \succ v_n$ *d'après la monotonie de l'agrégation.*

Considérons l'affectation $A = \{X_1^b, X_2^b, X_3^b, X_4^b\}$ *et deux nogoods de notre exemple*
$\mathbf{n} = (\emptyset, \mathbf{1}, \{\mathbf{C_3}, \mathbf{C_5}\})$ *et*
$\mathbf{n_2} = (\mathbf{A}, \mathbf{5}, \{\mathbf{C_1}, \mathbf{C_2}, \mathbf{C_3}, \mathbf{C_4}, \mathbf{C_5}\})$. $\Uparrow^A(n) = (A, 4, \{C_1, C_2, C_3, C_4, C_5\})$ *est de valuation strictement inférieure à celle de* n_2 *et en ce sens moins intéressant. Or la réduction de* n_2 *conduit au nogood nul de valuation plus faible que celle de* n. *Pour réduire* n_2 *tout en conservant le rapport de supériorité, nous ne réduisons* n_2 *qu'avec les contraintes violées par* A *et absentes de la justification de* n, *soit* C_1, C_2 *et* C_4 : $\Downarrow_{(A,n)}(n_2) = (\{X_3^b, X_4^b\}, 2, \{C_3, C_5\})$.

Algorithme final

L'algorithme utilise un ensemble de conditions d'ordre O initialement vide, ainsi que l'ensemble $\mathcal{N}$ des nogoods N_{x^i} associés à chaque valeur x^i et initialement nuls. Appelé avec l'ensemble vide, **Développe** renvoie un nogood dont la valuation est la valuation d'une solution optimale.

DBT-Retour(A, n)

1. Si l'affectation de n est vide FIN
2. Sinon, soit $y^j \in A$ telle que y ne soit inférieure à aucune autre variable de l'affectation de n selon $\leq_O$
   (a) **Met-à-jour**(A, n, y^j)
   (b) $A \leftarrow A - \{y = j\}$
   (c) **DBT**(()A)

Met-à-jour(A, n, y^j)

1. $\forall N_{z^k}$ dont l'affectation contient y^j, $N_{z^k} \leftarrow n_\emptyset$
2. $N_{y^j} \leftarrow \Downarrow_{A,N_{y^j}} (n)$
3. $O \leftarrow O \cup O(N_{y^j})$

Pour prouver que ces modifications entraînent la terminaison de cette procédure d'optimisation, nous nous contenterons de montrer qu'elle termine sur le problème de décision : existe-t-il une solution de valuation strictement inférieure à α.

La modification de **DBT-Retour** assure que, lors de l'appel de **Met-à-jour**(A, n, y^j)), aucune variable de n n'est supérieure à y selon $\leq_O$. Ceci a deux conséquences :

- toutes les nouvelles conditions induites par la création de N_{y^j} ne produisent pas de cycles. La propriété 3.27 justifie alors que la ligne 3 ne réduit pas la valeur de terminaison ;
- Cette même ligne ayant pour objet la présence de toutes les conditions d'ordres liées aux nogoods N_{z^k} présents, la seconde conséquence est que la ligne 1 n'altère que les valeurs d'interdiction de variables strictement supérieures à y.

Par ailleurs, la procédure **DBT-induite** assure que **DBT** ne construit une extension de A que si tous les nogoods N_{y^j} portant sur ses valeurs ont une augmentation de valuation

strictement inférieure à α. A moins d'une réduction de la borne α, le nogood n dans **DBT-Retour**(A, n) possède une valuation strictement supérieure à celle de tous les nogoods des valeurs de l'affectation A. Quel que soit le choix de la valeur y^j, la propriété de réduction partielle 3.30 nous assure alors que la valuation du nouveau nogood N_{y^j} (**Met-à-jour**, ligne 2) est strictement supérieure à celle de l'ancien.

La propriété 3.28 nous assure finalement que **Met-à-jour** a pour conséquence globale l'augmentation stricte de la valeur de terminaison.

Pour un problème à n variables, on est assuré que tous les n appels à **DBT** , soit une nouvelle solution est trouvée, soit il y a un appel à **Met-à-jour**. Si l'ensemble des valuations est fini[7], l'ensemble des valeurs de terminaison possibles est lui aussi fini. Alors, la procédure atteindra nécessairement une solution de valuation inférieure à *alpha* ou prouvera qu'il n'en existe pas dans un temps fini.

Il ne faut pas s'inquiéter de la reprise de la recherche lorsque la borne α est diminuée. Cette réduction peut avoir pour conséquence que la valuation de certains nogoods portant sur l'affectation courante devient supérieure ou égale à α. Ceci met en défaut notre raisonnement. Cependant, il faut noter que, dans ce cas, les premier pas de la recherche seront exclusivement consacrés à des retours arrière, jusqu'à ce que ces nogoods n'interviennent plus. En effet, **DBT-induite** interdit toute extension tant que la situation n'est pas rétablie. Il n'est donc pas nécessaire de repartir de zéro à chaque diminution de la borne supérieure.

Améliorations

Réduction de l'ordre partiel :

Remarque 3.3 *Soit x une variable dont la valeur d'interdiction est augmentée et y une variable supérieure à x selon $\leq_O$, on peut se permettre sans altérer la propriété 3.26 :*

- *de détruire tous les nogoods portant sur les valeurs de y (cf lemme 3.28), ou plutôt, si leur destruction n'est pas imposée (propriété 3.22), de les ignorer dans le calcul des valeurs d'interdiction ;*

7. On peut sans perdre de généralité limiter l'ensemble des valuations aux valuations obtenues, pour chaque sous-ensemble de contraintes, par l'agrégation des valuations de ses éléments. Il est alors fini au même titre que l'ensemble des sous-ensembles de contraintes.

- *et de remplacer dans O toutes les conditions d'ordre correspondantes par $x \leq y$. Il suffit en effet d'assurer que y dont la valeur d'interdiction est modifiée reste supérieure à x.*

Cette remarque permet d'affaiblir encore l'ordre partiel pesant sur les variables. Nous la concrétisons par la définition d'une procédure présentée dans [GM94].

La diminution $Dim(O, y)$ d'un ensemble de conditions O par une variable y par la procédure suivante :

Définition 3.31 *$Dim(O, y)$ Pour chaque variable z telle que $y \leq_O z$, enlever toutes les conditions de la forme $z' \leq z$ et ajouter $y \leq z$.*

Propriété 3.32 *Pour tout ordre total $\leq_1$ respectant $\leq_{Dim(O,y)}$ il existe un ordre total $\leq_2$ respectant $\leq_O$ tel que $\forall w \leq_2 y, w' \leq_1 w \iff w' \leq_2 w$. (cf [GM94])*

Corollaire 3.33 *Soit $\mathcal{N}$ un ensemble de nogoods N_{x^i}, et O un ordre partiel. Soit y une variable, et $\mathcal{N}'$ un ensemble de nogoods dérivé de $\mathcal{N}$ tel que la valeur de terminaison de y augmente strictement, alors que ne diminuent que celles de variables supérieures à y (selon $\leq_O$). Alors, $V(Dim(O, y), \mathcal{N}')$ est strictement supérieure à $V(O, \mathcal{N})$.*

Démonstration : (3.33) *Soit $\leq_1$ un ordre total associé à un élément minimum de $V(Dim(O, y), \mathcal{N}')$. D'après la propriété 3.32, il existe un ordre total $\leq_2$ compatible avec O et identique à $\leq_1$ pour toutes les variables inférieures à y. Or, avec $\mathcal{N}'$ ou $\mathcal{N}$, les valeurs d'interdiction de ces variables sont identiques. Comme la valeur d'interdiction de y augmente strictement avec $\mathcal{N}'$, la valeur d'interdiction associée à $\leq_1$ et $\mathcal{N}'$ est strictement supérieure à celle associée à $\leq_2$ et $\mathcal{N}$.*

Cette procédure peut donc être insérée lors de l'appel de **Met-à-jour** pour affaiblir l'ordre partiel.

Hybridation :

Remarque 3.4 **DBT-induite**(A) *n'est pas limitée à l'examen des nogoods N_{x^i} mais peut améliorer son estimation avec tout autre nogood portant sur A.*

Pour anticiper les retours arrière, on peut en particulier utiliser les nogoods issus d'un backward-checking, d'un forward-checking (**BT-induite**, **FC-induite**,...), voire même d'une agrégation des nogoods, etc.

Selon la puissance d'évaluation attribuée à la procédure **DBT-induite**, il est possible de se rapprocher d'une recherche locale fonctionnant sur des affectations complètes, ou d'une recherche complète classique stoppant les extensions au plus tôt. L'évaluation minimale proposée n'interdit que les extensions redondantes avec les explorations antérieures. Améliorée, elle peut aussi interdire les extensions n'ayant aucune chance d'améliorer la borne supérieure. Choisir d'autoriser plus de violations que nécessaire peut néanmoins s'avérer une bonne politique, surtout lorsque l'on est en mesure de tirer parti des violations rencontrées[SV94b].

Remarque 3.5 *Le choix d'une variable maximale y pour le backtrack provient de la nécessité de ne pas introduire de cycle lors de l'introduction du nogood N_{y^j}.*

Il est possible que y soit supérieure à certaines variable de l'affectation courante. Il suffit que y ne soit inférieure à aucune variable de l'affectation du nogood $N_{y^j} = \Downarrow_{A,N_{y^j}}(n)$ qui sera produit. Or la réduction et la projection ont pour effet de restreindre énormément l'affectation de n. On peut donc, par cette analyse, augmenter encore la liberté de choix du point de backtrack.

Finalement, nous disposons d'un algorithme complet utilisant un nombre polynômial de nogoods (1 par valeur). Il présente l'avantage d'offrir deux choix, celui de la valeur servant à l'extension et de celle visée par le backtrack. La liberté du second choix n'est que partielle mais permet néanmoins la mise en œuvre d'heuristiques supplémentaires.

Exemple :

Il est toujourss difficile de présenter l'arbre de recherche produit. Comme dans le cadre classique, nous nous contentons de suivre les premier pas de la recherche obtenue en affectant les variables dans l'ordre naturel (sauf interdiction due à un nogood). Les retours arrière sont effectués avec une priorité inverse mais toujours parmi les variables concernées par l'analyse de l'échec.

Nous donnons l'évolution de l'affectation courante, les raisons de son éventuel rejet, et les modifications entraînées par l'appel de **met-à-jour**.

variable 1	2	3	4	5	n	nogood ajouté	nogoods retirés	ordre partiel ajouts	ordre partiel retraits	α
a	.	.	.	.						
a	a	.	.	.						
a	a	a	.	.						
a	a	a	a	.						
a	a	a	a	a	$(\{X_1^a, X_2^a, X_3^a, X_4^a, X_5^a\}, 5, \{C_1, C_2, C_3, C_4, C_5\})$	$N_{X_1^a} = n_\emptyset$				5
.	a	a	a	a						
b	a	a	a	a	$(\{X_2^a, X_3^a, X_4^a, X_5^a\}, 4, \{C_2, C_3, C_4, C_5\})$	$N_{X_2^a} = n_\emptyset$				4
b	.	a	a	a						
b	b	a	a	a	$(\{X_3^a, X_4^a, X_5^a\}, 3, \{C_3, C_4, C_5\})$	$N_{X_3^a} = n_\emptyset$				3
b	b	.	a	a						
b	b	b	a	a	$(\{X_1^b, X_2^b, X_3^b, X_4^a, X_5^a\}, 5, \{C_1, C_2, C_3, C_4, C_5\})$	$N_{X_1^b} = n_\emptyset$				
.	b	b	a	a	$\bigvee(\Uparrow N_{X_1^a}, \Uparrow N_{X_1^b}) =$ $(\{X_2^b, X_3^b, X_4^a, X_5^a\}, 5, \{C_1, C_2, C_3, C_4, C_5\})$	$N_{X_2^b} = (\{X_3^b\}, 1, \{C_1\})$		$X_2 > X_3$		
.	.	b	a	a	$\bigvee(\Uparrow N_{X_2^a}, \Uparrow N_{X_2^b}) =$ $(\{X_3^b, X_4^a, X_5^a\}, 5, \{C_1, C_2, C_3, C_4, C_5\})$	$N_{X_3^b} = (\{X_3^b\}, 2, \{C_1, C_2\})$				
.	.	.	a	a	$\bigvee(\Uparrow N_{X_3^a}, \Uparrow N_{X_3^b}) =$ $(\{X_4^a, X_5^a\}, 3, \{C_1, C_2, C_3, C_4, C_5\})$	$N_{X_4^a} = (\{X_4^a, X_5^a\}, 2, \{C_1, C_2, C_3, C_4\})$		$X_4 > X_5$		
.	.	.	.	a						
a	.	.	.	a						
a	a	.	.	a	$\bigvee(\Uparrow N_{X_3^a}, \Uparrow N_{X_3^b}) =$ $(\{X_1^a, X_2^a, X_5^a\}, 3, \{C_1, C_2, C_5\})$	$N_{X_1^a} = (\{X_1^a, X_2^a, X_5^a\}, 3, \{C_1, C_2, C_5\})$		$X_1 > X_2$ $X_1 > X_5$	$+X_1 > X_3$ $-X_2 > X_3$	
.	a	.	.	a						
b	a	.	.	a	$\bigvee(\Uparrow N_{X_3^a}, \Uparrow N_{X_3^b}) =$ $(\{X_1^b, X_2^a, X_5^a\}, 3, \{C_1, C_2, C_5\})$	$N_{X_1^b} = (\{X_1^b, X_2^a, X_5^a\}, 3, \{C_1, C_2, C_5\})$				
.	a	.	.	a	$\bigvee(\Uparrow N_{X_1^a}, \Uparrow N_{X_1^b}) =$ $(\{X_2^a, X_5^a\}, 3, \{C_1, C_2, C_5\})$	$N_{X_2^a} = (\{X_2^a, X_5^a\}, 3, \{C_1, C_2, C_5\})$	$N_{X_1^a}$ $N_{X_1^b}$	$X_2 > X_5$		
.	.	.	.	a	etc.					

TAB. 3.3 – *Backtrack Dynamique valué : exemple*

3.4.3 Expérimentations

Version testée

Toutes les remarques concernant le Backtrack Dynamique valué sont implémentées dans une version "**DBT-FC**":

- l'ordre partiel est assoupli au maximum lors du backtrack sur une variable (cf remarque 3.3);
- afin d'atteindre des temps de calcul raisonnables, le nogood produit par **EFC-induite** est toujours utilisé (remarque 3.4);
- le choix du point de backtrack est élargi au maximum (remarque 3.5).

D'autre part, comme nous ne disposons pas d'heuristiques de choix du retour-arrière, la dernière valeur affectée est toujours choisie comme point de backtrack. Ceci nous permet du moins d'effectuer le même parcours arborescent que l'algorithme classique du Forward-checking afin de juger plus sûrement la pénalité des calculs supplémentaires imposés par le Backtrack Dynamique.

Résultats

Le tableau 3.4 compare **DBT-FC** au Forward-checking en termes de temps CPU et de nœuds développés et donne le nombre moyen d'opportunités lors de chaque backtrack (n) pour notre algorithme.

20 variables, 5 valeurs; Optimum 2-3										
	Couples connectivité/dureté									
DBT-FC	12/80	16/70	20/60	26/50	36/40	40/37	50/32	60/28	70/24	80/22
temps	0,86	1,71	2,35	2,89	2,99	2,90	2,85	2,81	2,69	2,58
noeuds	0,27	0,53	0,68	0,69	0,91	0,96	0,99	1,00	1,00	1,00
n	3,7	2,6	2,5	2,0	1,7	1,7	1,6	1,6	1,6	1,6

TAB. 3.4 – *Backtrack Dynamique valué : résultats*

Les résultats font tout d'abord apparaître que le temps de calcul supplémentaire imposé par la complexité de l'algorithme reste raisonnable, surtout pour les problèmes à

forte dureté. En ce qui concerne l'espace de recherche exploré, il faut noter que les mémorisations destinées à l'origine à garantir la terminaison permettent aussi de réduire l'exploration. En ce sens, le mode de mémorisation peut constituer une alternative pertinente à celui choisi pour le Nogood Recording.

La liberté du backtrack est moins importante lorsque la connectivité prime sur la dureté. Les nogoods produits possèdent alors nécessairement une affectation plus large et induisent un ordre partiel plus sévère. En ce qui concerne le nombre de choix, sa faiblesse doit être relativisée : les problèmes ne possèdent que 20 variables et bien peu sont affectées lorsqu'un forward-checking est employé. Par ailleurs, la liberté de choix intervient principalement en début de recherche. Le durcissement de l'ordre partiel en fin de recherche traduit simplement la nécessité d'un parcours exhaustif complet pour prouver l'optimalité du résultat. Cette phase étant la plus longue, elle influe beaucoup sur le résultat.

En résumé, cette technique apparaît raisonnable à condition que l'on dispose d'une heuristique de backtrack adéquate et que l'on ne s'attache qu'à des problèmes dont la preuve de l'optimalité des solutions est aisée. Lorsque ce second point n'est pas acquis, il peut être intéressant de figer en cours de recherche un ordre total et d'abandonner le comportement dynamique et ses lourdeurs. Une autre politique peut consister à considérer les mémorisations comme une méthode sûre permettant aux recherches locales d'éviter le piège des optimums locaux.

3.5 Résultats généraux

Nous reprenons sur des planches communes (figure 3.11) les résultats obtenus avec les divers algorithmes proposés :

- **FC**, le Forward-checking valué ;
- **DAC**, le prétraitement avec l'algorithme Directed Arc-consistancy Counts suivi du Forward-checking ;
- **NR-x**, le Nogood Recording valué avec une stratégie de mémorisation ne retenant que les nogoods d'arité inférieure ou égale à x ;
- **DBT**, le Dynamic Backtracking valué ;

- **AC+FC**, le prétraitement avec l'arc-cohérence valuée suivi du Forward-checking ;
- **AC+FC-**, le prétraitement avec l'arc-cohérence valuée, avec la production d'une unique contrainte par valeur, suivi du Forward-checking ;
- **AC+AC**, le renforcement permanent de l'arc-cohérence. Les expérimentations de cette version se limitent aux instances les plus faciles car les temps d'exécution sont envion 100 fois plus long que ceux des autres algorithmes.

Conclusions

Le modèle des contraintes induites qui s'appuie sur l'adjonction d'une justification aux contraintes est relativement simple. Sa mise en œuvre pratique est cependant plus délicate. Les manipulations des contraintes sont multiples et offrent beaucoup de degrés de liberté... trop semble-t-il. En effet, les traitements sont parfois contradictoires, sans que l'on sache jamais choisir à coup sûr les plus intéressants. La question du meilleur compromis entre les trois critères de qualité des contrainte induite — affectation réduite, valuation élevée, justification réduite — se pose en permanence sans obtenir de réponse certaine et définitive.

Il apparaît cependant que la gestion des justifications est au centre des problèmes. En effet, les algorithmes de résolution des CSP tirent leur efficacité de la décomposition du critère global en contraintes portant sur de petits sous-problèmes de quelques variables. Il est même impératif d'éviter les contraintes de trop forte arité. Pour une contrainte induite, l'importance de l'arité est simplement synonyme d'inutilité (cf l'évaluation du Nogood Recording). Or, non seulement une réduction de l'arité passe inévitablement par une sélection fine des justifications, mais, de plus, les agrégations nécessaires pour reconstituer des valuations intéressantes sont extrêmement dépendantes des justifications.

Nous avons essayé de proposer des outils optimaux dans la mesure où ils permettent d'éviter le recours à un compromis dans la gestion des justifications. Il s'agit tout d'abord des opérations de réduction et d'augmentation qui permettent de diminuer la taille des justifications et par conséquent des affectations sans altérer les valuations. Ces opérations sont une réponse suffisamment simple et efficace au problème de la réduction de contraintes induites pour être systématiquement mises en œuvre lors de l'utilisation des contraintes induites.

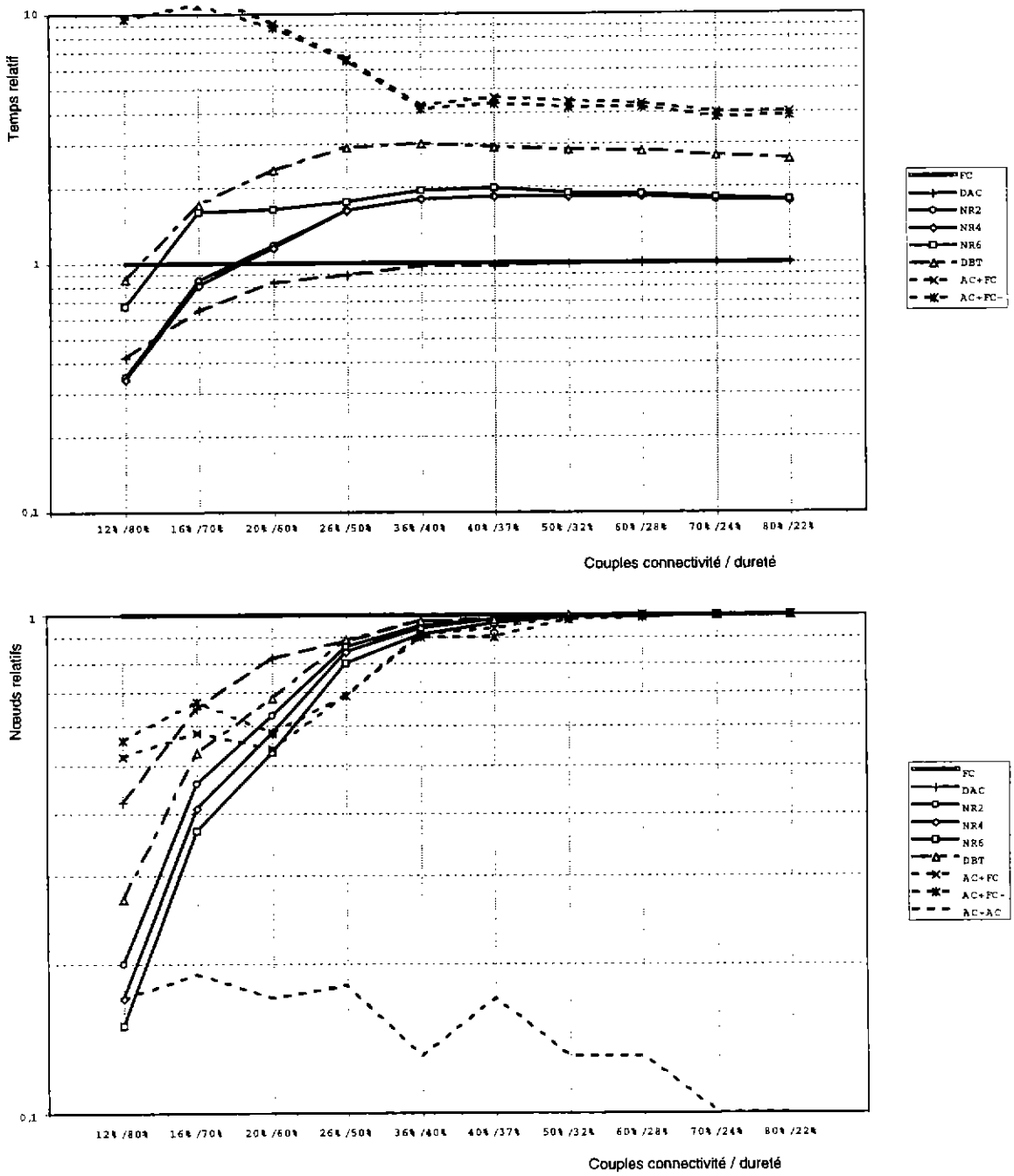

FIG. 3.11 – *Résultats généraux*

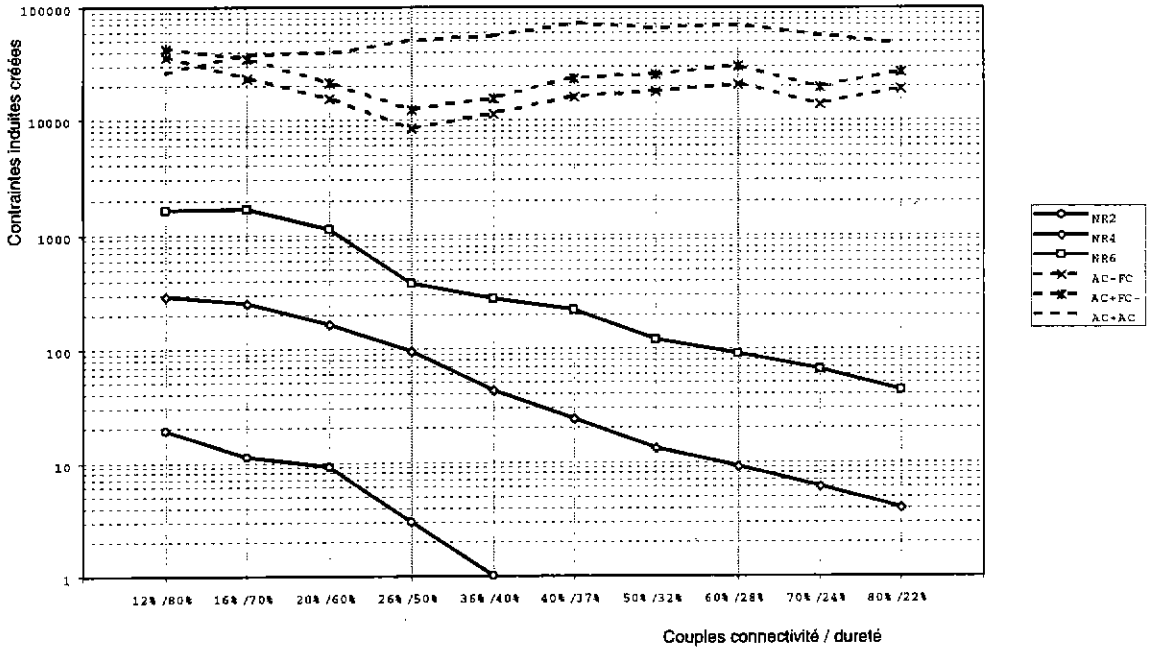

FIG. 3.12 – *Résultats généraux*

Nous avons aussi défini un opérateur d'agrégation permettant de tirer d'un ensemble de contraintes induites une valuation maximale. Ce-dernier est cependant d'une utilisation pratique trop complexe pour améliorer l'efficacité des recherches. La principale leçon qu'il permet de tirer concerne la disjonction des justifications : si l'on cherche à agréger efficacement un ensemble de contraintes induites, il est impératif de maîtriser *a priori* les intersections de leurs justifications. La voie à suivre pour obtenir une efficacité supérieure à celle de DAC par une utilisation massive de valuations induites semble être la définition de recouvrements judicieux de l'ensemble des contraintes. Et ce afin d'agréger librement les valuations justifiées par chaque sous-ensemble.

Les contraintes induites utilisées avec parcimonie et à bon escient permettent cependant d'apporter des propriétés nouvelles, sources d'efficacité. Le Backtrack Dynamique est un exemple original des possibilités liées aux contraintes induites. L'application à laquelle sont destinés ces développements est une illustration concrète de l'intérêt que peut représenter l'utilisation des contraintes induites.

Chapitre 4

Ordonnancement de systèmes satellitaires : définition du problème et solutions actuelles

4.1 Présentation du problème

Les satellites d'observation de la Terre qui nous intéressent ont pour mission la collecte d'informations (image radar, photographie optique, etc.) et la transmission de ces données vers des stations réceptrices au sol. L'objectif de tels systèmes est de répondre à un maximum de demandes dans les meilleurs délais. Leur rôle d'observation leur impose généralement de survoler tout le globe terrestre à basse altitude ce qui, comme nous le verrons, est la source des problèmes d'optimisation de l'utilisation des satellites d'observation. Outre les contraintes matérielles liées à l'observation, des exigences provenant des clients ou des opérateurs peuvent être formulées.

4.1.1 L'observation de la Terre par satellite

La requête initiale d'un client correspond à une demande d'observation d'une zone de la Terre, ou "**demande d'image**". Le travail d'observation demandé est traduit en tâches élémentaires de "**prise de vue**". Cette dernière se définit par l'activation d'instruments d'observation et d'enregistrement selon un mode et à des dates donnés. Une demande

d'image sur une zone suffisamment petite pourra être satisfaite par une seule prise de vue, sinon, plusieurs prises de vue sur des zones de taille réduite sont nécessaires. Les opérateurs du système ont à charge de traduire les requêtes des clients en demandes d'image sur des mailles (zone réduite accessible en une prise de vue) recouvrant l'ensemble des surfaces à photographier. Les créneaux de visibilité d'une maille sont autant de prises de vue permettant la réalisation de sa photographie.

Aux prises de vue qui collectent l'information correspond la tâche de "**télémesure**" consistant à établir le contact avec une station sol afin de lui transmettre les images réalisées et enregistrées à bord des satellites.

La principale caractéristique des systèmes satellitaires d'observation provient de l'orbite des satellites. Ils ne peuvent pas se déplacer à leur guise au-dessus du sol mais sont assujettis à une trajectoire, figée lors de la mise à poste, dont ils ne peuvent s'écarter que de façon exceptionnelle[1]. Contrairement aux satellites de communication géostationnaires[2] évoluant à 36 000 km d'altitude, les orbites des satellites d'observation sont "**défilantes**" à des altitudes de l'ordre de 500 km à 1500 km[3], et ce pour plusieurs raisons : la première provient de la nécessité d'observer tout le globe terrestre, la seconde de la précision limitée des instruments de prise de vue. De ce fait, ils ne peuvent exercer leurs activités principales que lors de "**créneaux de visibilité**", durées durant lesquelles le satellite survole les cibles au sol (stations sol, zone à imagier, etc.).

Un mot sur le fonctionnement des instruments de prise de vue : ils offrent une certaine souplesse dans la mesure où la relative liberté de leur direction de visée[4] peut compenser la rigidité de la trajectoire. Il faut cependant noter qu'ils ne fonctionnent généralement pas à la manière d'un appareil photo habituel capturant toute la scène d'un coup, mais en mode défilant. L'instrument ne photographie en fait qu'une fine bande sous la trajectoire du satellite. Ce dernier étant défilant, la bande photographiée balaye le sol, et l'ensemble des bandes imagées reconstitue la prise de vue complète. Il en résulte des dates et durées de prises de vue étroitement liées aux créneaux de visibilité.

1. Les manœuvres de changement d'orbite sont très coûteuses en ergols ou gaz propulsifs. Elles ne sont donc utilisées que ponctuellement, pour des déviations infimes afin généralement de compenser "l'usure" de la trajectoire et de maintenir une orbite parfaite.

2. Un satellite est dit géostationnaire s'il est toujours à la verticale du même point au sol.

3. La trajectoire au sol et l'altitude d'un satellite sont directement liées. Pour simplifier, plus le satellite est proche du sol, plus il se déplace vite; à 36 000 km, il tourne dans le plan de l'équateur aussi vite que la Terre.

4. Instruments radar essentiellement.

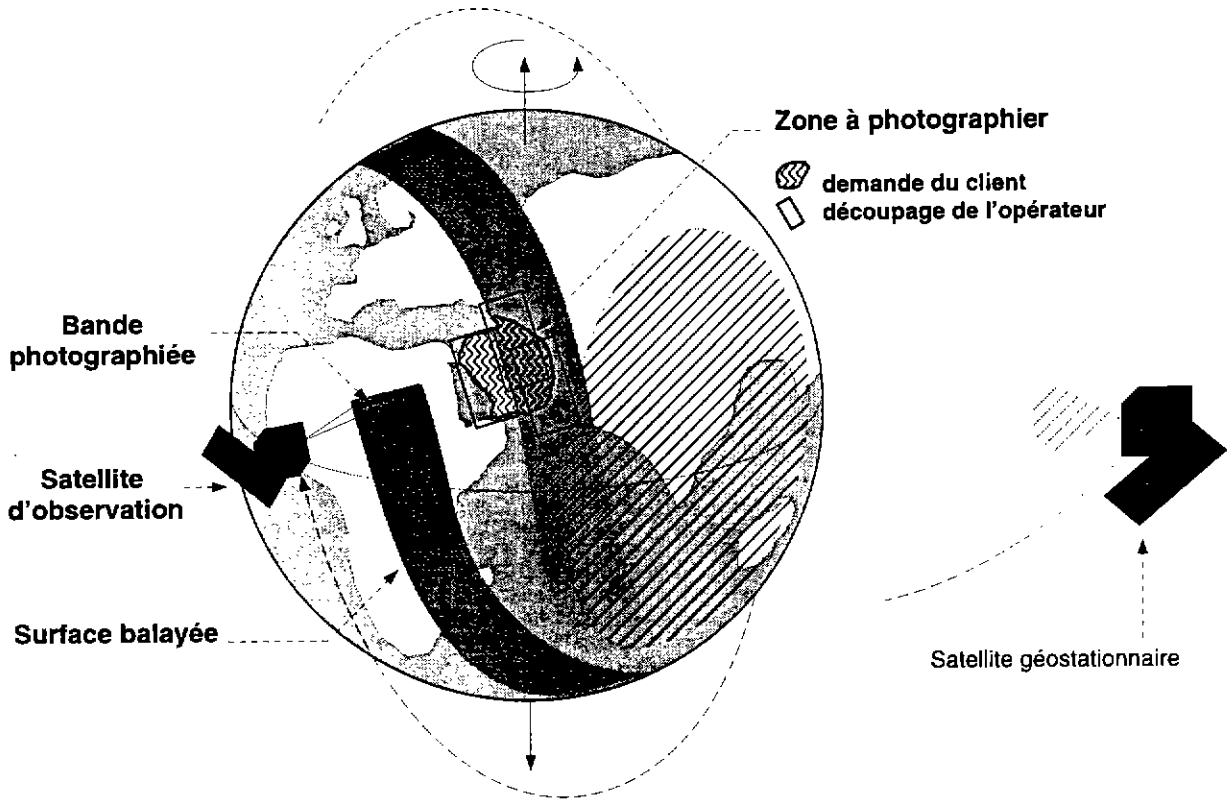

FIG. 4.1 – *Satellites défilants.*

Les instruments de prise de vue sont adaptés à l'observation de divers types de rayonnements (radar, visible, infrarouge ou autre). Les nuages étant opaques à certains de ces rayonnements, il est souvent nécessaire de prendre en compte des facteurs climatiques dans la réussite des opérations de prise de vue.

4.1.2 Contraintes

Les principales contraintes sont matérielles et découlent des ressources limitées de la plateforme spatiale. Les équipements au sol sont en général suffisants pour ne pas contraindre le travail des satellites. Les ressources du satellite peuvent être regroupées en trois catégories : les instruments de prise de vue, les limitations en débit et les limitations en capacité.

Tout d'abord, les instruments de télémesure et de prise de vue ne peuvent fonctionner que lorsque le satellite est en visibilité avec la cible au sol. Un autre impératif matériel interdit à un instrument photographique d'effectuer deux prises de vue simultanément et impose de respecter certains délais d'orientation et d'activation entre deux images . Ces ressources sont utilisées par les prises de vue, puis restituées à la fin de la tâche. Ceci se traduit par des contraintes d'exclusion entre prises de vue que l'on voudrait effectuer lors d'un même passage, sur des zones géographiques superposées ou trop proches.

Ensuite, la puissance électrique des plateformes et la vitesse d'enregistrement sont limitées. Ces ressources sont elles aussi utilisée pendant les prises de vue puis restituées. La différence avec le premier type de ressource provient de la nature des exclusions. Les premières sont binaires et liées à la géographie des images. Les secondes dépendent des débits et de la date des prises de vue. Par ailleurs, le dépassement des limitations peut parfois nécessiter la conjonction de plus de deux prises de vue.

L'espace mémoire de stockage des images en attente de télémesure, ainsi que l'énergie disponible dans les batteries de la plateforme constituent un troisième type de ressources. La mémoire est consommée lors de la réalisation d'une prise de vue et renouvelée lors de chaque télémesure. L'énergie est consommée par l'activation de tous les instruments et renouvelée lors des passages en exposition solaire. La limitation en mémoire est généralement plus contraignante que celle en énergie. Cependant, certains instruments, d'imagerie radar en particulier, ont une forte consommation de cette seconde ressource. Ces ressources

sont fortement cumulatives[5], donc concernent simultanément des ensembles importants de prises de vue, et elles sont renouvelables selon des modes plus ou moins complexes.

Les autres contraintes pesant sur l'observation relèvent essentiellement des souhaits exprimés par les clients : date au plus tôt, au plus tard, date de livraison, mode d'observation (image radar, infrarouge, ...), qualité, etc. Elles sont directement intégrées dans la définition des prises de vue satisfaisant la demande d'image.

Plus rarement, certaines exigences de synchronisation entre diverses photographies sont exprimées ; dans tous les cas, il s'agit de souhaits peu contraignants. En effet, le temps de réponse très élevé d'un système d'observation[6] et les aléas climatiques rendent improbable la satisfaction d'exigences temporelles pointues.

4.1.3 Formalisation des objectifs

Les requêtes des clients ont conduit à la définition d'un certain nombre de mailles à photographier et d'ensembles de prises de vue associés. L'opérateur doit décider des prises de vue qui seront effectivement réalisées par le système satellitaire et de leurs dates de réalisation afin d'en déduire la programmation du satellite. En d'autre termes :

Il faut sélectionner un ensemble de prises de vue matériellement réalisables et satisfaisant au mieux les demandes d'image.

Le problème de construction d'un plan réalisable est un problème combinatoire : pour chaque maille, il faut sélectionner une de ses prises de vue. Les contraintes matérielles lui imposent une structure forte. Les instruments et les limitations en débit interdisent des sous-ensembles de prises de vue nécessairement proches sous la trajectoire du satellite. A ces contraintes "locales" viennent s'ajouter des contraintes "globales" portant sur l'ensemble des prises de vue et limitant leur nombre total par l'intermédiaire des capacités restreintes des batteries et de l'enregistreur.

Le problème de sélection d'un plan satisfaisant devient alors un problème d'optimisation combinatoire fortement structuré. L'importance des contraintes portant sur la réalisabilité des plans constitue le principal obstacle à l'optimisation : l'application d'heuristiques basées sur une connaissance *a priori* du problème se heurte invariablement à des

5. Il faut un grand nombre des tâche consommantes pour saturer la ressource.

6. Le temps de réponse est le délai entre le dépôt de la requête et le retour au sol des photographies correspondantes.

interdictions ou à des incompatibilités matérielles.

4.2 L'ordonnancement de systèmes satellitaires : solutions existantes

4.2.1 Aide à la décision

La majorité des moyens opérationnels de planification des missions d'observation s'appuient sur les compétences des opérateurs humains. Ceux-ci sont soutenus dans leur tâche par des outils d'aide à la décision qui ont pour but de collecter et de synthétiser les informations concernant le plan en cours de production [DJ96, PCR96, Phi96]. Schématiquement, l'opérateur insère une à une les prises de vue qu'il sélectionne. A tout moment, les outils l'informent sur le plan en cours, les ressources disponibles, les tâches en attente et les conséquences immédiates des diverses actions à la disposition de l'opérateur.

Lorsqu'un fonctionnement automatique est possible, il est obtenu par l'intégration sous forme de règles des stratégies heuristiques des opérateurs. Le système sélectionne alors de manière automatique les prises de vue à insérer en priorité, et, éventuellement, celles à rejeter en cas de conflit. Dans tous les cas, l'optimisation dépend uniquement de l'expertise humaine.

Il s'agit essentiellement d'outils d'aide à la décision laissant le soin de l'optimisation aux opérateurs. Les modes d'ordonnancement automatique s'appuient uniquement sur la mise en œuvre d'heuristiques définies par ces mêmes opérateurs.

4.2.2 SPOT 1

V. Gabrel illustre sur SPOT 1 une méthodologie pertinente pour la planification des systèmes satellitaires d'observation[Gab94]. L'analyse du travail de planification qu'elle propose permet une répartition des tâches entre machine et opérateur humain adaptée à leur compétences respectives. Pour des questions de décisions stratégiques et tactiques multicritères, il ressort que l'intervention humaine est indispensable. L'utilisation d'outils automatiques dans une analyse à long et moyen terme est néanmoins nécessaire pour dégrossir le travail d'optimisation et présenter aux opérateurs des informations synthétiques

sur la faisabilité et l'opportunité de la réalisation des différentes images.

Les outils d'analyse proposés sont suffisamment puissants pour permettre, au besoin, un ordonnancement entièrement automatique avec une bonne maîtrise du niveau d'optimisation. Ils sont issus de la recherche opérationnelle et exploitent les particularités de structure des problèmes liés aux satellites d'observation. Ils ont en effet en commun de photographier en balayant le sol avec leurs instruments au cours de leur déplacement. De cette liaison forte entre le temps et la position géographique découle une organisation des contraintes entre prises de vue structurée par la date de survol. Nous verrons plus en détail par la suite la nature de ces contraintes et les outils de recherche opérationnelle qui y sont associés.

4.2.3 OSCAR

Les travaux menés au CERT sur SPOT s'intéressent essentiellement à l'aspect automatique du problème d'ordonnancement.

C. Badie et G. Verfaille[Bad88, BV89, VS94] ont développé un outil permettant la construction progressive d'un plan de production par l'allocation de ressources aux images progressivement insérées. Les techniques de la recherche opérationnelle et de l'intelligence artificielle sont alliées. La première assure la validité d'une insertion vis-à-vis des contraintes matérielles et revient, si besoin est, sur les choix antérieurs. La seconde permet de guider au moyen de règles la sélection des images, le choix de la prise de vue et la stratégie d'insertion (retour sur les choix antérieurs et sélection de ces-derniers).

Cet outil présente l'avantage de pouvoir insérer en temps réel les demandes d'images et offre un analyse utile des éventuels conflits produits par l'insertion. Il ne permet cependant pas d'optimisation donc n'autorise aucun contrôle sur la qualité du résultat final. En outre, les ressources cumulatives telles que la mémoire ne sont pas prises en compte.

Plus récente, une étude menée en collaboration entre le CNES et le CERT s'est attachée à traiter l'ordonnancement de SPOT 5 comme un problème d'optimisation combinatoire[ABB+95, BVA+96]. L'objet était de comparer les méthodes locales et exactes, et parmi ces-dernières, les modélisations sous forme de CSP ou de problèmes

linéaires en nombres entiers. Les algorithmes habituels résolvant les CSP valués se sont avérés incompétents et deux méthodes se sont illustrées :

4.2.4 Recherche tabou

La première est issue des recherche locales et adapte le principe des recherches tabou [GL93]. Les prises de vue sont considérées dans l'ordre chronologique et un algorithme glouton produit rapidement une solution en sélectionnant systématiquement les prises de vue compatibles avec les sélections antérieures. Le moteur de recherche locale consiste à choisir une prise de vue du plan réalisable et à la rejeter. L'algorithme glouton produit une nouvelle solution en reprenant la construction du plan à partir de la prise de vue sélectionnée. La sélection des prises de vue insérées par l'algorithme glouton et rejetées par le moteur de recherche locale est dirigée par une liste tabou mémorisant les sélections et rejets effectués dans les derniers pas de la recherche. L'objectif est d'éviter de répéter trop souvent les mêmes choix.

Cette méthode présente l'intérêt de s'affranchir assez aisément des difficultés posées par les contraintes cumulatives et permet par sa souplesse la mise en œuvre d'heuristiques nombreuses. Il faut cependant un grand nombre d'itérations et l'optimalité des solutions ne peut pas être prouvée.

4.2.5 Pseudo-programmation dynamique

A l'opposé, la seconde méthode à l'origine des Poupées Russes[VLS96] est extrêmement rigide, optimale et très rapide, mais est mise en échec par les contraintes de forte arité. Elle s'inspire directement des méthodes de programmation dynamique et se place dans le cadre des VCSP. Les images sont ordonnées dans l'ordre chronologique et la recherche résout successivement (par un Forward-checking valué) les sous-problèmes constitués par les n dernières images, n allant de 0 au nombre total d'images. Ces résultats intermédiaires sont utilisés pour accélérer les recherches ultérieures en fournissant une évaluation pertinente des pénalités nécessairement entraînées par la poursuite du plan courant avec les k dernières variables.

Cet algorithme est particulièrement efficace sur les problèmes d'ordonnancement de SPOT 5. Cependant, il est très sensible à la structure des problèmes qu'il résout et s'avère

incapable de terminer sa recherche lorsqu'interviennent des contraintes d'arité importante.

4.2.6 Positionnement de l'étude

Il apparaît tout d'abord que la modélisation sous forme de CSP valués reste parfaitement adaptée au problème de l'ordonnancement de systèmes satellitaires. La difficulté réside dans la définition d'outils adaptés à ce problème. Sur ce point, l'algorithme des Poupées Russes démontre d'une part que la résolution optimale du problème d'ordonnancement sur un horizon limité à environ une journée est accessible, et d'autre part, qu'il est possible d'exploiter la structure du problème au moyen de valuations induites.

Notre premier souci consistera donc à adapter nos algorithmes à base de contraintes induites, afin de résoudre le problème de l'ordonnancement à court terme.

Par ailleurs, la relative faiblesse des moyens d'optimisation impose de prendre en compte l'intervention d'un opérateur humain dans le processus d'ordonnancement. A ce titre, le travail de V. Gabrel nous semble le plus pertinent puisqu'il limite l'utilisation des moyens algorithmiques à la résolution optimale des sous-problèmes qui lui sont accessibles, avec pour objectif final d'aider le travail d'optimisation d'un opérateur humain. Notre objectif est cependant d'élargir le champ des possibilités des outils d'aide à l'ordonnancement et donc, à terme, de produire des ordonnancements satisfaisants de manière entièrement automatique. Pour l'heure, conscient des limites de nos capacités d'optimisation, nous garderons toujours à l'esprit le fait que l'ordonnancement doit être autant que possible orientable et modifiable de manière externe. Comme nous l'avons vu, les contraintes induites permettent d'apporter en partie la souplesse recherchée.

Le principe que nous utilisons (les contraintes induites) est donc volontairement plus général que celui des Poupées Russes. Il s'adapte *a priori* bien à notre application tout en apportant des propriétés potentiellement intéressantes (explication du résultat, fonctionnement dynamique, souplesse et variété des modes de recherche). Nous jouerons plus particulièrement sur ces possibilités pour approcher la résolution exacte des problèmes à court terme où des contraintes globales telles que la limitation en ressource mémoire sont dimensionnantes.

Chapitre 5

L'ordonnancement à court terme

5.1 Ordonnancement journalier du satellite SPOT 5

5.1.1 Problème posé et objectifs

Le futur satellite SPOT 5 qui nous intéresse est héliosynchrone[1]. Trois instruments optiques identiques sont disponibles. Ils peuvent modifier l'angle de leur visée vers la droite ou la gauche de la trace au sol du satellite, indépendamment les uns des autres. Ils se différencient par leur pointage fixe, sous le satellite, vers l'avant ou vers l'arrière. Une image peut être réalisée en "mono" par chacun des trois instruments. Pour une réalisation "stéréo" elle nécessite l'utilisation combinée des deux instruments avant et arrière.

Trois hypothèses simplificatrices, mais réalistes dans le cadre de notre application sont utilisées :

- les équipements au sol ne sont pas contraignants. Ils sont toujours en mesure de suivre la demande du satellite ;
- la capacité en énergie électrique n'est pas contraignante ;
- les télémesures n'entrent pas en conflit avec les prises de vue et sont toujours utilisées pour transmettre une fois par jour les images effectuées.

La gestion des ressources sol et énergétiques sera donc ignorée. Quant aux télémesures,

1. A l'endroit où tombe l'ombre du satellite, il est toujours midi ou minuit !

elles ne seront pas planifiées, mais directement intégrées dans la gestion de la contrainte mémoire : peu importe comment se fera une télémesure, il suffit de connaître la consommation de mémoire potentielle qu'elle autorise.

Enfin, le critère d'optimisation nous est donné par le biais d'une valeur entière dérivant du gain commercial attendu et attribué à chaque image. L'objectif est bien sûr de fournir un ordonnancement qui maximise la somme des valeurs des images satisfaites.

5.1.2 Modélisation du problème journalier

La modélisation du problème sous la forme d'un CSP valué est tout à fait naturelle :

La structure de valuation est additive, avec l'intégration de la valuation infinie : $(E, \preceq, \bot, \top, \oplus) = (\mathbb{N}, \leq, 0, \infty, +)$.

A chaque maille est associée une variable x_k. Ses valeurs x_k^i correspondent à chaque prise de vue permettant la réalisation de l'image de la maille. Il faut aussi compter une valeur supplémentaire x_k^*, dite "**valeur rejet**", pour signifier que l'on choisit de ne pas réaliser l'image.

Les contraintes sont de quatre types ; les premières correspondent au critère d'optimisation, et les autres traduisent les différentes catégories de contraintes matérielles :

- des contraintes unaires portent sur les valeurs rejet de chaque variable. Leur valuation est fonction de l'importance accordée à l'image ;
- des contraintes binaires portent sur des paires d'images et interdisent les paires de prises de vue en conflit sur un instrument photographique. Leur valuation est infinie;
- des contraintes de faible arité interdisent des n-uplets de prises de vue provoquant des dépassements de débit. La faible arité (2 ou 3) vient du nombre d'instruments qui limite à 3 le nombre maximum de prises de vue simultanées. Leur valuation est infinie ;
- une contrainte cumulative, associée à chaque intervalle entre deux télémesures, qui porte sur la majorité des variables du créneau et interdit les n-uplets de prises de vue nécessitant un espace mémoire supérieur à la mémoire disponible. Leur valuation est infinie. Bien sûr, comme la génération *a priori* de tous les n-uplets

interdits est irréaliste, cette contrainte est traitée en intention sous la forme : $\sum_{x_k^i \text{ sélectionnées}} \text{consommation}(x_k^i) \leq \text{capacité_mémoire}$.

L'ensemble des contraintes matérielles est généré à l'aide du modèle de fonctionnement du satellite. Les contraintes d'exclusion sur les instruments se déduisent de l'examen, pour chaque prise de vue, de son enchaînabilité avec les prises de vue qui la suivent. Les contraintes de débit interviennent ensuite par l'examen des prises de vue simultanées satisfaisant les contraintes d'exclusion. Enfin, la contrainte mémoire impose simplement que la somme des consommations des prises de vue ne doit pas dépasser une capacité M donnée.

5.1.3 Jeux d'essai

Les jeux de données utilisés pour mettre en application les développements algorithmiques des chapitres précédents sont fournis par le Centre National d'Etudes Spatiales.

Les problèmes correspondent à la simulation réaliste des requêtes de clients sollicitant les services du satellite. Ces dernières sont divisées en demandes d'images puis en prises de vue dont toutes les caractéristiques sont données (date, consommations, instruments utilisés, etc.). Un modèle simplifié de fonctionnement du satellite permet de vérifier si une séquence de prises de vue respecte les contraintes matérielles. Il permet de construire les contraintes instrumentales ou de débit qui lient les prises de vue.

L'ensemble des jeux d'essai consiste en 498 problèmes[lem] :

- 362 problèmes (numérotés de 1 à 362), correspondant à des simulations limitées à une orbite, pour lesquels la limitation de l'enregistreur peut être ignorée.
- du plus gros de ces 362 problèmes (le numéro 11), 13 problèmes (401 à 413) sont dérivés en réduisant le nombre de demandes d'images. 9 autres (501 à 509) sont construits à partir du même problème numéro 11 en réduisant le nombre de conflits entre les photographies.
- 101 problèmes (1001 à 1101) sur plusieurs orbites où la prise en compte de la contrainte mémoire est impérative.

- du plus important de ces problèmes (le numéro 1021), 6 problèmes (1401 à 1406) et 7 problèmes (1501 à 1507) sont obtenus respectivement en réduisant le nombre de demandes d'images et le nombre des conflits.

Les VCSP ainsi produits mettent en échec l'ensemble des méthodes de résolution complètes classiques. Même si l'on ne tient pas compte de la contrainte mémoire, particulièrement difficile à gérer dans un schéma de recherche par essai-erreur du fait de son arité[2], les problèmes générés restent difficilement solubles par les algorithmes classiques (à l'exception des récentes Poupées Russes et d'algorithmes de programmation linéaire en nombres entiers dédiés au problème) dans des limites de temps raisonnables.

Dans un premier temps, nous nous attacherons à résoudre le problème sans contrainte mémoire. Nous étudierons dans un second temps la prise en compte de cette contrainte particulière.

5.2 Contraintes locales

Les contraintes locales sont essentiellement constituées de contraintes d'exclusion sur les instruments photographiques. Ces dernières présentent des propriétés très spécifiques qui orienteront les méthodes de résolution.

Exclusion et plus long chemin

Pour mettre en évidence la structure particulière que les contraintes d'exclusion imposent au problème, il faut adopter l'hypothèse que chaque image ne possède qu'une opportunité de réalisation par créneau de planification. Cette hypothèse est réaliste car le délai de revisite moyen[3] des satellite d'observation est généralement plus long que les

2. L'archétype du problème avec contraintes de forte arité est celui des "pigeons" qui consiste à faire entrer $n+1$ pigeons dans n boites contenant un pigeon. Les algorithmes dédiés aux CSP explorent toutes les combinaisons possibles d'affectation des n premiers pigeons dans les n boites pour constater à chaque fois l'impossibilité de placer le dernier pigeon : toutes les feuilles de l'arbre de recherche sont explorées.

3. Le délais de revisite, comme son nom l'indique, est le temps séparant deux passages du satellite en visibilité d'une cible au sol

plages de temps sur lesquelles porte l'ordonnancement. Pour chaque demande d'image, le choix est binaire : l'effectuer par l'unique prise de vue possible ou la rejeter.

Alors, il est possible d'ordonner les images selon leur date de début de prise de vue croissante. Il en découle une propriété bien mise en évidence dans [Gab94] :

Propriété 5.1 *L'enchaînabilité, définie comme la contraposée de l'incompatibilité matérielle entre prises de vue successives (selon l'ordre des dates de début), est une relation binaire transitive.*

Soient trois prises de vue x_i, x_j et x_k, avec $i < j < k$. Si x_j peut être effectuée après x_i, et x_k après x_j, alors x_k est réalisable derrière x_i. Evidemment, ceci ne peut concerner que les compatibilités ou incompatibilités liées à l'enchaînement temporel des prises de vue, donc aux dates et durées de réquisition du matériel photographique. Par ailleurs, un instrument ne tolérant qu'une image à la fois, les contraintes qu'il impose peuvent se mettre sous la forme de contraintes binaires interdisant des paires de prises de vue. Ceci nous assure que si les prises de vue respectent deux à deux la relation d'enchaînabilité, elles sont globalement compatibles.

La conséquence directe est la possibilité d'utiliser des algorithmes de programmation dynamique et, plus précisément, de plus court chemin (cf figure 5.1) : il s'agit de trouver un enchaînement optimal des prises de vue dans un graphe transitif d'arcs autorisés. La résolution construit de manière récursive les séquences optimales ayant pour origine une prise de vue donnée à l'aide de la propriété suivante :

Propriété 5.2 *L'enchaînement optimal partant d'une prise de vue est constitué de cette prise de vue suivie du meilleur des chemins optimaux partant de ses successeurs directs dans le graphe d'enchaînabilité.*

Démonstration : (5.2) *Notons $S(x_n)$ une séquence de prises de vue partant de la prise de vue de l'image x_n et poursuivie par les prises de vue d'un ensemble d'images $x_k, k \geq n$, et $\hat{S}(x_n)$ une séquence $S(x_n)$ optimale. Une séquence optimale $\hat{S}(x_{n-1})$ est constituée de x_{n-1} suivie de la meilleure séquence $S(x_k), k \geq n$ compatible avec x_{n-1}. La propriété 5.1 assure que la compatibilité est équivalente à l'enchaînabilité de x_{n-1} et x_k. On peut donc se limiter à l'examen des séquences $\hat{S}(x_k)$ où x_k est un successeur de x_{n-1}. Enfin, pour limiter l'examen aux successeurs directs, il suffit de constater que si x_k précède $x_{k'}$, alors x_k suivie de $\hat{S}(x_{k'})$ est une séquence meilleure que $\hat{S}(x_{k'})$.*

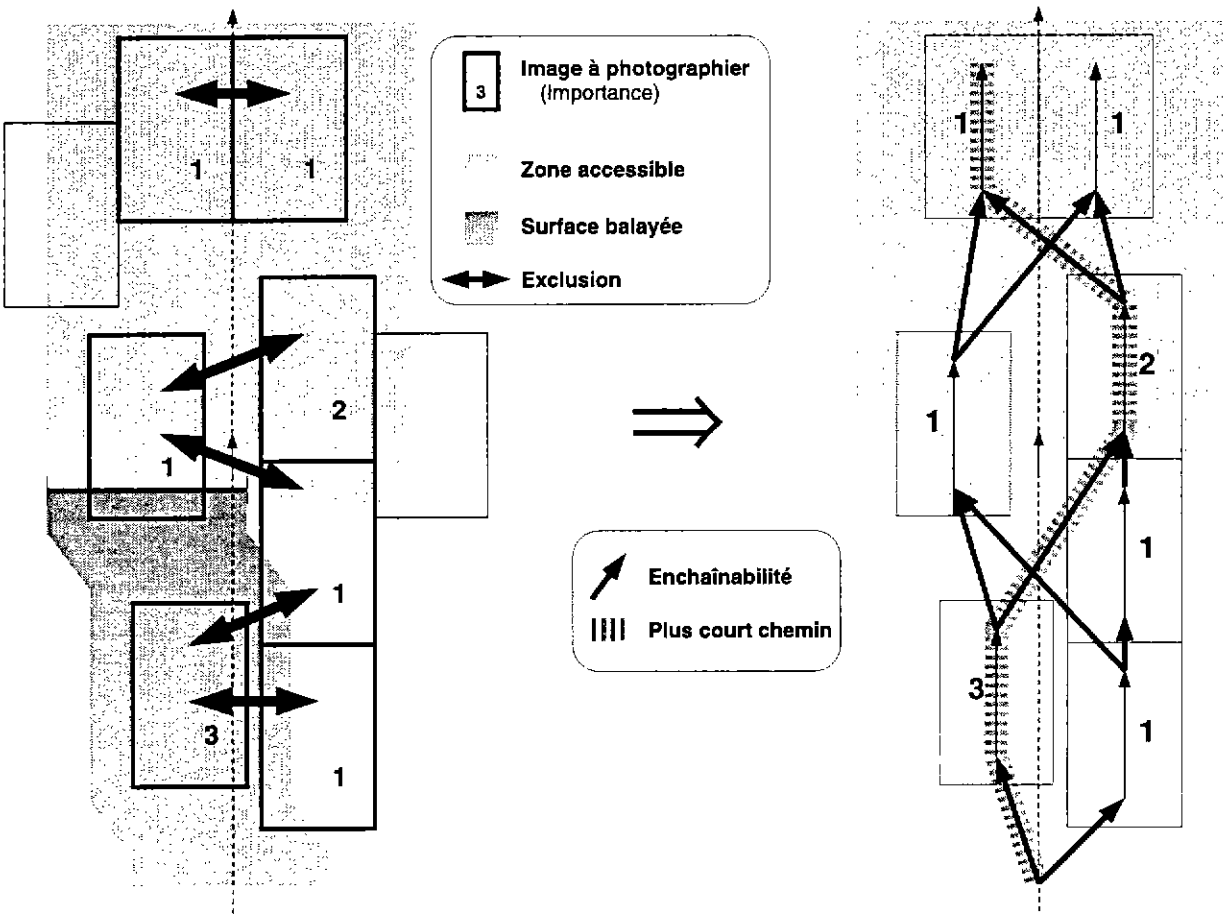

FIG. 5.1 – *Plus long chemin*

Notre application est cependant trop complexe puisqu'elle ne fait aucune hypothèse sur le nombre d'opportunités. Au contraire, nous nous intéressons à un satellite multi-instruments qui offre donc plusieurs opportunités de prise de vue à chaque passage. La difficulté vient de ce que l'on risque de dupliquer la réalisation des images. On est obligé d'introduire des contraintes interdisant les couples de prises de vue se rapportant à une même image. Or, ces incompatibilités brouillent la transitivité de la relation d'enchaînabilité. x^1 peut être suivie de y et y de x^2, sans que l'enchaînement des trois ne soit autorisé. Cette difficulté peut être levée grâce à un traitement simultané des prises de vue d'une même image, de manière à ne pas risquer de poursuivre le choix de l'une d'entre elles par un chemin passant par une autre prise de vue. Cela revient à supposer que l'on est en mesure d'exhiber un ordre total des prises de vue qui respecte à la fois les relations temporelles entre prises de vue et le regroupement des prises de vue d'une même image. Un tel ordre signifie que les zones au sol sont toujours survolées dans le même ordre et que les instruments de prise de vue ont des débatements et des activations indépendants[4] (dans le cas multi-instruments).

Multi-instruments et plus long chemin

Le satellite qui nous intéresse est optique et héliosynchrone. Cela implique en particulier que les prise de vue sont toujours effectuées dans le même sens de balayage (par exemple lors des passages du sud vers le nord). Un ordre des images satisfaisant est donc obtenu en les classant par latitude croissante. On est alors certain que les prises de vue effectuées lors du passage d'un instrument respectent l'ordre de leurs images.

Pour l'exemple, considérons seulement deux instruments photographiques. Si l'on considère que le satellite ne survole pas deux fois le même point au sol, cela revient à dire que l'on dispose de deux passages pour effectuer les images. Une image x_k possède donc deux opportunités, x_k^1 et x_k^2. Pour chaque passage, une relation transitive d'enchaînabilité des prises de vue correspondantes est définie. Alors, pour prendre en compte simultanément les deux passages, nous considérons des chemins doubles effectués par les deux instruments. Nous notons $S(x_m, x_n)$ un chemin partant de la prise de vue de l'image x_m sur le premier instrument et de la prise de vue de l'image x_n sur le second. $S(x_m, x_n)$ est donc constitué des prises de vue x_m^1 et x_n^2 ainsi que de deux ensembles de prises de

4. On peut alors considérer un survol avec plusieurs instruments comme plusieurs survols indépendants avec chaque instrument.

vue $x^1_k, k \geq m$ et $x^2_{k'}, k' \geq m$ tels que les contraintes d'enchaînabilité soient respectées. Nous nous intéressons bien sûr aux chemins de valeur optimale $\hat{S}(x_m, x_n)$partant de deux prises de vue x^1_m et x^2_n .

La relation récurrente utilisée est la suivante :

Propriété 5.3

- *Si* $m < n$, *alors* $\hat{S}(x_m, x_n) = x^1_m + \max\{\hat{S}(x'_m, x_n)$ *tel que* x'^1_m *soit un successeur direct de* $x^1_m\}$;
- *Si* $m > n$, *alors* $\hat{S}(x_m, x_n) = x^2_n + \max\{\hat{S}(x_m, x'_n)$ *tel que* x'^2_n *soit un successeur direct de* $x^2_n\}$;
- *Si* $m = n$, *alors* $\hat{S}(x_m, x_m) = x^1_m + x^2_m + \max\{\hat{S}(x'_m, x''_m)$ *tel que* x'^1_m *et* x''^2_m *soient respectivement des successeurs directs de* x^1_m *et* $x^2_m\}$.

Démonstration : (5.3) *La condition* $m < n$ *assure que* x^2_m *n'est présente dans aucun des chemins* $\hat{S}(x'_m, x_n)$. *Dans la mesure où, d'autre part, les deux instruments sont indépendants, la propriété 5.2 prouve le premier point. La démonstration du second point est symétrique de la première.*

Le dernier point relève toujours du même principe : le chemin optimum partant de x^1_m *et* x^2_m *est nécessairement constitué de ces prises de vue suivies du meilleur des chemins compatibles avec les deux prises de vue.*

Ces relations suffisent à la production des chemins optimaux partant de toutes les paires de prises de vue. Cependant, il n'est pas nécessaire de développer la totalité des chemins car les relations récurrentes ne font intervenir que les successeurs directs. La figure 5.2 illustre les calculs nécessaires si l'on suppose que les successeurs directs sont parmi les 4 images suivantes. On notera que l'efficacité de la procédure est liée à ce que l'on peut considérer comme une "largeur de bande" du problème. Plus les conflits restent locaux, plus il est facile d'étendre un chemin optimal avec une nouvelle image. Cette méthode est néanmoins toujours polynômiale.

Il faut aussi remarquer que les relations récurrentes prennent en compte aussi bien les cas où la duplication d'une image est inutile que ceux où elle est nécessaire (image

stéréo). Tout dépend la valeur que l'on attribue aux chemins. Pour une image mono, un chemin passant par les deux prises de vue de l'image doit prendre la même valeur que si le chemin ne passait que par une seule prise de vue. Pour une image stéréo, seuls les chemins passant par les deux prises de vue seront accrédités de la valeur attribuée à l'image.

Il ressort, dans tous les cas, que les contraintes instrumentales conduisent à une structure forte du problème qu'il est possible et intéressant d'exploiter car les méthodes de résolution associées sont extrêmement performantes. La généralité du problème posé, alliée à la variété des contraintes, nous interdit cependant d'exploiter ces méthodes en l'état : il est nécessaire de les généraliser au cas où les propriétés de l'enchaînabilité ne sont pas systématiquement respectées.

Généralisation de la programmation dynamique

Dans notre application, l'inconvénient majeur provient de la dépendance entre les divers instruments de prise de vue, dépendance résultant de la limitation globale en débit énergétique de la plateforme. Ces contraintes supplémentaires sont cette fois impossibles à intégrer à un processus de recherche de plus long chemin. Elles ne remettent cependant pas en question la structure générale du problème ni les principes généraux adaptés à ce type de structure.

L'idée est simple : si l'extension des séquences optimales avec une nouvelle image ne

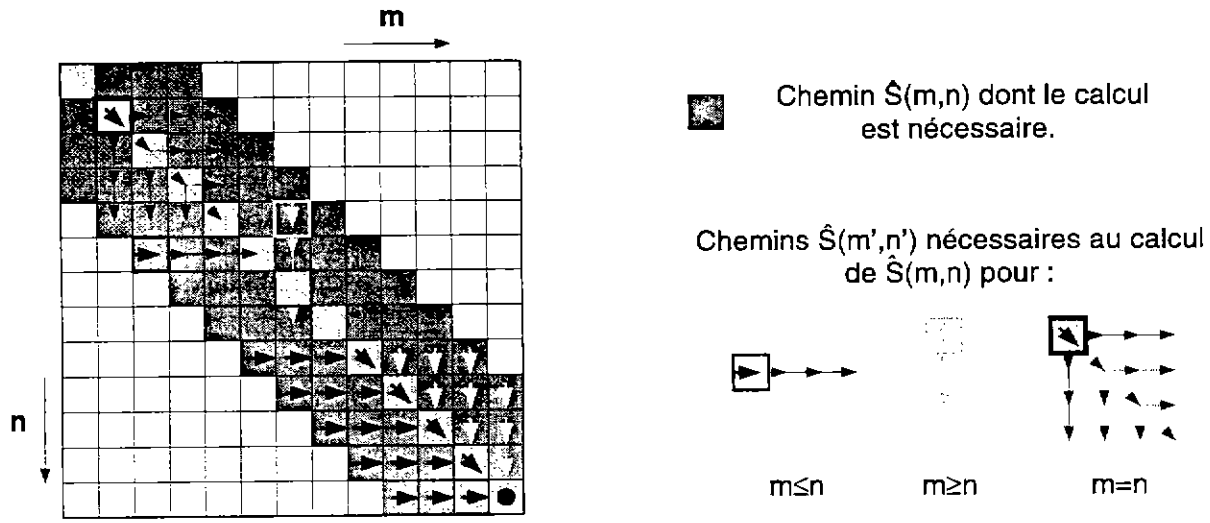

FIG. 5.2 – *Plus long chemin double*

dépend pas que de la compatibilité de la prise de vue considérée avec ses successeurs temporels directs, du moins ne dépend-elle que des compatibilités entre un petit noyau de prises de vue immédiatement connectées par des contraintes. Le principe de mémorisation des solutions partielles doit conserver sa pertinence.

L'algorithme des "**Poupées Russes**" offre une réponse et permet, en quelque sorte, de généraliser la programmation dynamique dans le cadre des VCSP. A la différence de la programmation dynamique, la mémorisation de la solution optimale pour les n dernières variables ne permet pas de construire directement la solution optimale, mais donne seulement un minorant de la valuation globale des chemins que l'on compte poursuivre par ces n variables. Cet algorithme est extrêmement simple, donc rapide, et bien adapté au problème qui nous intéresse [ABB+95].

Pour notre part, nous préférons cependant adapter les algorithmes sur le modèle du Nogood-recording qui relève d'un principe plus général et plus souple que les Poupées Russes. Les nogoods ont pour effet de mémoriser en cours de recherche des minorants concernant n'importe quel sous-ensemble de contraintes, et de les soumettre éventuellement à condition (affectation du nogood). Les Poupées Russes ne produisent implicitement que des nogoods de la forme $(\emptyset, v, \mathcal{C})$, où $\mathcal{C}$ est l'ensemble des contraintes portant sur les n dernières variables. Ceci nous permettra de travailler plus librement, dans le cadre général et clairement posé des contraintes induites, quitte à particulariser et à simplifier les algorithmes par la suite.

5.2.1 VCSP particulier

Indépendamment de la structure du graphe de contraintes, le VCSP additif associé au problème présente des particularités importantes qui devront être prises en compte.

Les contraintes dures, tout d'abord, ne méritent pas d'être intégrées aux justifications. En effet, aucune redondance n'est à craindre sur une contrainte de poids infini ($\infty + \infty = \infty$). Par ailleurs, les contraintes de poids infini correspondent à des impératifs matériels immuables. Le second intérêt des justifications, la possibilité d'un fonctionnement dynamique et interactif, ne concerne jamais ces contraintes. Nous pourrons donc limiter les justifications aux seules contraintes interdisant les valeurs de rejet.

Il faut d'autre part insister sur deux points :

- les contraintes souples, dont dépend l'optimisation, sont unaires et ne portent que sur les valeurs de rejet ;
- les contraintes dures ne concernent que les valeurs associées à une prise de vue.

Que devient dans ces conditions un forward-checking évaluant l'extension avec une variable ?

- Soit une prise de vue est encore accessible, et le forward-checking ne permet aucune augmentation de la valuation globale et renvoie une valuation nulle ;
- soit toutes les prises de vue sont incompatibles avec les affectation antérieures, et le choix de la valeur de rejet donne la plus faible valuation que l'on peut espérer.

Dans ce second cas, la nature particulière du VCSP implique que le choix de la valeur de rejet est une condition nécessaire. De plus, la seule conséquence de ce choix est la violation de la contrainte unaire associée à la valeur de rejet.

Plutôt que de considérer le forward-checking comme une évaluation prévisionnelle, nous pouvons le considérer comme un rejet forcé des images dont toutes les prises de vue sont interdites. L'intérêt de ce point de vue est qu'il remplace la contrainte induite que produirait le forward-checking habituel par des violations effectives. Comme nous l'avons vu, l'agrégation de ces dernières avec d'autres contraintes induites est grandement facilitée. En d'autres termes, cette manœuvre permet d'intégrer l'efficacité du forward-checking dans un cadre de contraintes induites, tout en évitant les coûts de gestion des contraintes induites qu'il produit.

5.2.2 Propositions

Nous adaptons donc l'algorithme du Nogood-recording valué présenté précédemment au vu des remarques précédentes : les contraintes matérielles sont ignorées dans les justifications, et l'évaluation utilise un forward-checking qui affecte autoritairement à la valeur rejet les images dont toutes les prises de vue sont interdites.

D'autre part, la forme particulière de notre problème permet de se ramener simplement à une recherche similaire aux Poupées Russes. Il suffit pour cela d'utiliser un ordre statique

et de commencer par affecter toutes les images à la valeur de rejet. La première feuille atteinte est le rejet de toutes les prises de vue et la recherche va progressivement essayer d'introduire les prises de vue en remontant vers la racine. L'affectation courante est alors composée d'une série de valeurs de rejet $A = \{x_0^*, ..., x_k^*\}$ suivie d'affectations quelconques. Lorsque le nogood recording remonte par un retour arrière jusqu'à la variable x_k, il fournit un nogood $n = (A, v, \mathcal{C})$ où v correspond à la valuation optimale des extensions de A. La réduction de ce nogood correspond à la soustraction de l'ensemble des contraintes $c_i^*, 0 \leq i \leq k$. Dans la mesure où les valeurs de rejet correspondantes n'apparaissent dans aucune autre contrainte et sont indépendantes du reste du problème, la réduction naturelle du nogood $\Downarrow(n)$ est égale à $(\emptyset, v - v(A), \mathcal{C}')$, où $\mathcal{C}'$ est un ensemble de contraintes concernant exclusivement les violations liées aux variables $x_p, p > k$, et $v - v(A)$ est la valuation optimale du sous-problème restreint aux variables $x_p, p > k$. On peut encore rapprocher le Nogood Recording des Poupées Russes en n'utilisant, lors de l'évaluation d'une affectation A, que le nogood portant sur le sous-problème $\mathbb{V} - \bar{A}$.

Etant donné l'efficacité du principe des Poupées Russes, nous ne nous priverons pas d'utiliser cette variante facile du Nogood-recording. Par comparaison, le Nogood-recording habituel et les poupées Russes sont les versions opportuniste et systématique d'un même algorithme.

Quelques chiffres comparatifs

Nous donnons les résultats obtenus avec plusieurs versions :

- "**FC**" Le Forward-checking étendu (un prétraitement avec **DAC** est inutile car il ne produit aucune valuation).
- "**NR-x**", le Nogood Recording naturel ne mémorisant que les nogoods d'affectation de taille inférieure ou égale à x.
- "**NR-dbt**", et le Nogood Recording copiant la stratégie de mémorisation du Backtrack Dynamique : les nogoods sont conservés tant que leur affectation est incluse dans l'affectation courante à une valeur près.
- "**NR-dbt-x**", le cumul des deux stratégies.
- "**NR***", le Nogood Recording lancé avec l'affectation de toutes les variables à la valeur de rejet. Cette version est extrêmement proche des Poupées Russes.

Toutes les remarques faites dans la section précédente, concernant le forward-checking en particulier, sont implémentées pour le nogood recording. On se référera à la section 5.3.3, page 135 pour une description précise de l'algorithme utilisé.

L'ordre des variables est statique et correspond à l'ordre croissant des dates de première opportunité des images. La seule heuristique utilisée consiste à affecter en priorité une variable à la valeur qui participait à la dernière solution trouvée. L'objet est d'obtenir un comportement proche de la programmation dynamique qui produit un nouveau chemin optimal à partir des chemins optimaux antérieurs.

Les résultats font apparaître le nombre de variables des problèmes n ainsi que le meilleur résultat obtenu en 30 minutes et le temps nécessaire à son obtention en secondes. Lorsque l'optimalité du résultat est prouvée, il est précédé du signe "*".

Du fait de la forme de chaîne du graphe de contraintes, il est très facile de produire des nogoods de faible arité. Sur certains gros problèmes, des difficulté liées à l'espace mémoire utilisé par les nogoods peuvent subvenir si la sélection est trop faible. Lorsque l'exécution est interrompue à cause d'un dépassement mémoire (elle est limitée à $350Mbit$!) les résultats sont donnés en **gras**. Les versions "NR-dbt-1" et "NR-dbt-2" nécessitant un espace mémoire important ($> 80Mbit$) sont testées sur un ordinateur différent et environ deux fois plus lent que celui utilisé pour les autres tests.

Nous reprenons pour comparaison les jeux d'essai utilisés dans [BVA+96]. Il s'agit d'une sélection de problèmes d'ordonnancement suffisamment petits pour que la contrainte mémoire n'intervienne pas (problèmes limités à une orbite).

Problèmes sans contrainte mémoire : exemples													
Pb	n	FC		NR-0		NR-dbt		NR-dbt-0		NR-dbt-1		NR-dbt-2	
54	67	37	406	*37	1	*37	1	*37	1	*37	1	*37	2
29	82	*8059	1431	*8059	1	*8059	1	*8059	1	*8059	2	*8059	3
42	190	159050	651	155050	15	155050	13	155050	10	155050	23	**155050**	**10s**
28	230	278105	5	277105	225	276105	1300	276105	585	276105	1645	277105	116
5	309	275	1	275	1	275	1	275	1	275	1	**275**	**1s**
11	364	41497	25	38484	1129	40484	1002	*38478	1498	38484	1329	**41493**	**242s**
404	100	119	81	*114	2	*114	4	*114	1	*114	7	*114	15
408	200	8234	1	6234	170	6233	1796	*6228	388	6234	135	**6234**	**161s**
412	300	35400	1	32387	417	34387	75	*32381	712	**32387**	**392s**	**35392**	**123s**
503	143	11119	125	*11113	7	*11113	7	*11113	2	*11113	6	*11113	7
505	240	22260	84	21259	30	*21253	1698	*21253	390	*21253	1008	**21259**	**93s**
507	311	29405	96	27395	1448	28401	1230	27395	1388	28406	1446	**29405**	**21s**
509	348	39457	11153	39457	208	39457	261	36452	1456	39457	692	**39460**	**123s**

TAB. 5.1 – *Résultats sans contrainte mémoire*

A l'évidence, les nogoods d'affectation vide assurent l'essentiel de l'efficacité du nogood recording. Il n'est pas utile de risquer des débordements mémoire avec une sélection plus large. Il ne faut cependant pas conclure à l'inutilité absolue des nogoods d'arité élevée : ils permettent une importante réduction de l'espace de recherche, mais à un prix trop élevé. En effet, pour une même valuation de solution, "NR-dbt-2" explore en moyenne deux fois moins d'affectations partielles que "NR-dbt-0".

La stratégie de mémorisation du backtrack dynamique apparaît comme une option plausible qui se marie bien avec une sélection habituelle selon l'arité des contraintes induites.

Dans tous les cas, le Forward-checking est largement surclassé. Le Nogood Recording valué permet une résolution optimale de problèmes dont la taille est particulièrement importante pour ce genre de techniques.

Les résultats des algorithmes complets présentés dans l'article d'où nous tirons la sélection des jeux d'essai [BVA+96] sont reportés ci-dessous. Ils concernent les évaluations du Forward-checking "**DFBB**", des Poupées Russes"**RDS**" et de la programmation linéaire en nombres entiers "**PL**". Il est important de préciser que les algorithmes utilisés dans cet article sont hautement spécialisés et utilisent des méthodes de décomposition et de transformation des VCSP à résoudre ainsi que des heuristiques sur l'ordre des variables. On pourra évaluer l'écart de performances par lacomparaison des performances des deux Forward-checking "FC" et "DFBB" identiques sur le principe.

La simplicité des Poupées Russes en fait un algorithme d'une grande rapidité que le lourd Nogood Recording égale difficilement sur les plus gros problèmes. Les résultats sont néanmoins suffisants pour que les mémorisations sous forme de contraintes induites constituent une alternative plausible. En particulier, elles autorisent la mise en œuvre de nombreuses stratégies de recherche, et donc permettent d'échapper à la rigidité du mode de recherche de Poupées Russes. Etant donné l'impossibilité des Poupées Russes de fournir des solutions, même sous-optimales, aux problèmes avec contrainte mémoire (du fait de leur rigidité), la relative souplesse des contraintes induites, alliée à leurs performances, est bienvenue.

Problèmes sans contrainte mémoire : exemples											
Pb	n	NR-dbt-0		NR*-dbt-0		RDS		BFBB		DFBB	
54	67	37	1	*37	1	*37	3	*37	1	*37	32
29	82	*8059	1	*8059	1	*8059	1	*8059	1	*8059	12
42	190	155050	10	176056	1798	*155050	14	*155050	74	159050	1201
28	230	276105	585	292105	1176	270105	415	270105	916	273105	612
5	309	275	1	283	1799	*262	1702	262	1800	264	1213
11	364	*38478	1498	*38478	383	*38478	419	38480	1800	39502	646
404	100	*114	1	*114	1	*114	2	*114	7	115	600
408	200	*6228	388	*6228	32	*6228	184	*6228	292	6234	603
412	300	*32381	712	*32381	157	*32381	255	*32381	1667	33415	611
503	143	*11113	2	*11113	3	*11113	38	*11113	3	12115	611
505	240	*21253	390	*21253	30	*21253	108	*21253	76	22255	603
507	311	27395	1388	*27390	111	*27390	303	*27390	1270	29426	620
509	348	36452	1456	*36446	270	*36446	382	36448	1800	36467	638

TAB. 5.2 – *Résultats comparatifs sans contrainte mémoire*

Problèmes sans contrainte mémoire : statistiques					
Nombre de Variables	Nombre de problèmes	% résolution < 30min			
		FC	NR-dbt-0	NR*-dbt-0	RDS
- 100	267	79,6%	99,2%	100,0%	99,6%
100 - 200	61	0,0%	54,1%	78,7%	90,2%
200 - 300	38	0,0%	47,4%	63,2%	55,3%
300 - 400	13	0,0%	30,8%	46,2%	53,8%
400 -	4	0,0%	0,0%	0,0%	0,0%
Total	383	55,1%	83,6%	90,1%	90,6%

TAB. 5.3 – *Résultats statistiques sans contrainte mémoire*

5.3 Contrainte globale

5.3.1 Contrainte mémoire et optimisation multicritère

Contrairement aux contraintes locales de débit, la contrainte mémoire repose simultanément sur une grande partie des prises de vue, provoquant des liaisons entre des images très éloignées dans le graphe d'enchaînabilité. De fait, elle compromet fortement le succès des principes liés à la mémorisation de solutions partielles. Par ailleurs, le caractère cumulatif de la contrainte mémoire met en échec les algorithmes de satisfaction de contraintes en général. En effet, ceux-ci tirent leur puissance de la détection *a priori* des incohérences globales grâce à la localité des violations de contraintes. Lorsqu'une contrainte est cumulative, il faut affecter un grand nombre de variables, donc s'enfoncer profondément dans l'arbre de recherche, avant de pouvoir tirer des conclusions sur la satisfaction de la contrainte. Prenons un exemple. Supposons que nous avons n images à ordonnancer, avec pour chaque image le choix entre une prise de vue et le rejet. La seule contrainte est la capacité d'enregistrement limitée à $n-1$ prises de vue. Un algorithme de recherche considérant la contrainte comme des n-uplets interdits est amené à examiner l'ensemble de toutes les feuilles. Les méthodes de prévision sont impuissantes à anticiper la violation de la contrainte mémoire puisqu'il existe toujours un moyen de la satisfaire (le rejet). L'analyse des échecs est inopérante car chaque échec est dû à l'affectation entière.

Il ressort de l'étude du problème sans contrainte mémoire que l'on est parfaitement capable d'optimiser un critère additif soumis à l'ensemble des contraintes matérielles, mémoire mise à part. Or, la contrainte mémoire est un critère additif que l'on souhaite non pas minimiser, mais tenir en dessous d'un certain seuil. D'ailleurs, le problème posé ci-dessus fait clairement apparaître qu'on ne s'épargnera des explorations inutiles qu'à la condition d'ajouter les prévisions de consommations faites sur chaque variable séparément. Cependant, compter les consommations de mémoire ne suffit pas car il est toujours possible de ne pas consommer. Il est impératif d'associer les consommations à la valeur des solutions correspondantes. L'exemple proposé est trivial pour nous car nous sommes en mesure de tenir le raisonnement : "*ne pas consommer plus que la capacité est possible, mais au prix d'un rejet*", et plus généralement : "*ne consommer que m est possible, mais au prix de $n-m$ rejets*".

Ces réflexions nous conduisent naturellement à définir une structure de valuation bicritère permettant aux nogoods de donner une information combinée du type "toute

extension complète requérant un supplément de consommation maximal de m induit une pénalité minimale de p".

Précisons ce que nous entendons par valuation bicritère. Le premier critère est une structure additive $(\mathbb{N}, \leq, 0, \infty, +)$ naturelle associée aux pénalités. Le second critère est la consommation mémoire modélisée par une structure additive $([0, M] \cup \{\infty\}, \leq, 0, \infty, +)$ restreinte à l'intervalle $[0, M]$, si M est la capacité mémoire maximale[5]. Nous travaillons dans un espace cartésien à deux dimensions noté $\mathbb{N}_2$ dont les points sont des couples pénalité-consommation que nous appellerons "**coûts**". Cet espace est muni d'un opérateur d'addition et d'un ordre partiel, directement dérivés des structures de valuation de chaque critère. Soit $\varsigma_1 = (p_1, m_1)$ et $\varsigma_2 = (p_2, m_2)$ deux coûts :

- $\varsigma_1 + \varsigma_2 = (p_1 + p_2, m_1 + m_2)$
- $\varsigma_1 \leq \varsigma_2 \iff p_1 \leq p_2$ et $m_1 \leq m_2$

Le coût d'une affectation est donné par le couple formé, d'une part, de la somme des pénalités associées aux images rejetées par l'affectation, et, d'autre part, de la somme des consommations des prises de vue sélectionnées. Si l'on associe à chaque prise de vue une contrainte unaire correspondant à sa consommation, le coût d'une affectation correspond naturellement à la somme des coûts des contraintes violées.

Cette modélisation sous la forme d'une paire de nombres est cependant insuffisante. Elle ne permet pas d'effectuer les prévisions et les mémorisations que l'on souhaite. En effet, l'exploration d'une variable conduit généralement à une conclusion de la forme "soit j'augmente la consommation, soit j'augmente la pénalité", selon la valeur que l'on choisit. Or, le seul moyen d'agréger ces deux valuations potentielles en une seule valuation certaine (à des fins de mémorisation ou de prévision) est de considérer leur minimum, à l'instar de la propriété d'union. La seule valuation inférieure à la fois à une pénalité pure et à une consommation pure est la valuation nulle ! Si, d'un côté, il est possible d'éviter toute pénalité et, de l'autre, de ne rien consommer, alors, à moins de mémoriser toutes les possibilités, rien ne peut être déduit. Ceci nous oblige à considérer des valuations plus complexes qui soient en mesure de modéliser ce type de disjonction plus finement que par un unique couple pénalité-consommation.

La valuation d'un nogood ne sera pas un point dans un espace à deux dimensions en deçà duquel aucune solution n'est à espérer, mais une surface délimitée par une courbe à

5. Il faut comprendre que si $x \geq M$, alors $x = \infty$.

l'intérieur de laquelle aucune solution n'est à espérer. La section qui suit présente formellement les adaptations de la structure de valuation qui permettront d'intégrer la contrainte mémoire au critère d'optimisation.

5.3.2 Structure de valuation bicritère

Surface valuation

Définition 5.4 (valuations) *Une valuation s est une surface de $\mathbb{N}_2$ engendrée par une ensemble de coûts $c_i \in \mathbb{N}_2$. Elle se définit comme $s = \{\varsigma \in \mathbb{N}_2$ tels que $\exists i / \varsigma \geq \varsigma_i\}$.*

Nous appelons S l'ensemble des valuations. Nous appelons valuation unitaire une valuation engendrée par un unique coût et notons v l'application de $\mathbb{N}_2$ dans S qui à chaque coût ς associe la valuation unitaire $s = v(\varsigma)$ correspondante.

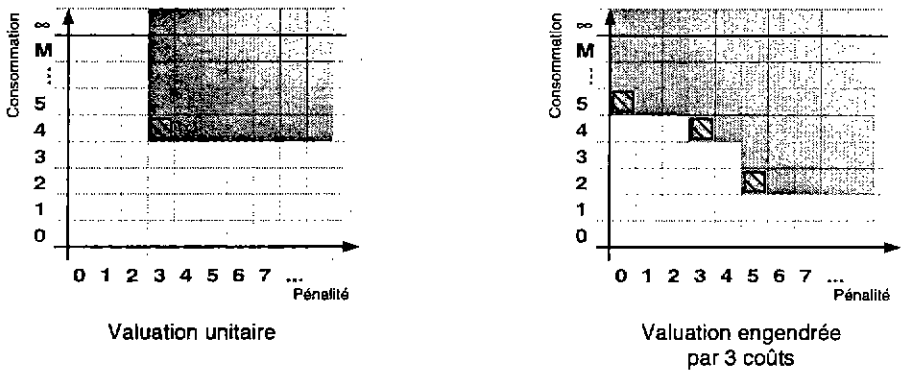

FIG. 5.3 – *Valuations*

La signification de ces valuations est la suivante : si une contrainte de valuation s est violée par une affectation A, alors le coût de toutes ses extensions appartient à la surface définie par s

Nous définissons maintenant les opérateurs associés à ces valuations.

Définition 5.5 (ordre partiel) *Soit s_1 et s_2 deux valuations, $s_1 \preceq s_2 \Longleftrightarrow s_1 \supset s_2$*

Une valuation est plus faible qu'une autre si l'ensemble des coûts autorisés par la seconde est un sous-ensemble de ceux autorisés par la première. Cet ordre partiel hérite donc de toutes les bonnes propriétés de l'inclusion[6].

6. Ceci prouve en passant qu'il s'agit d'un ordre partiel.

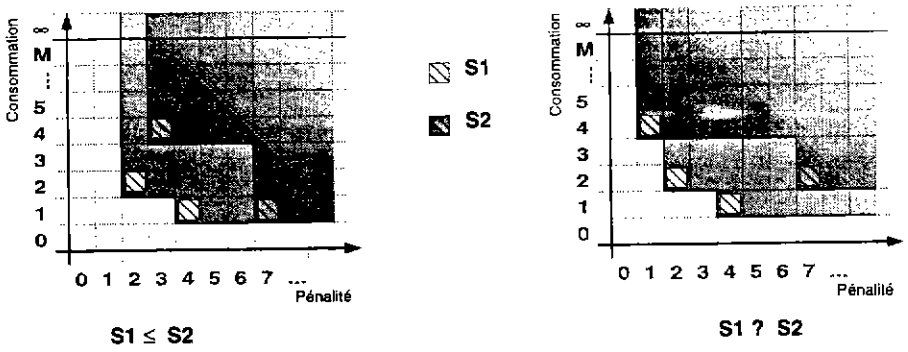

FIG. 5.4 – *Ordre partiel*

Définition 5.6 (agrégation) *Soit s_1 et s_2 deux valuations, $s_1 \oplus s_2 = \{\varsigma \in \mathbb{N}_2$ tels que $\exists \varsigma_1 \in s_1, \varsigma_2 \in s_2 / \varsigma \geq \varsigma_1 + \varsigma_2\}$.*

Remarque 5.1 *L'agrégation d'une valuation s avec une valuation unitaire déduite d'un coût ς correspond à une simple translation de s selon le vecteur c.*

Pour des raisons pratiques de temps de calcul, nous essaierons de nous limiter dans notre application à des agrégations de ce type.

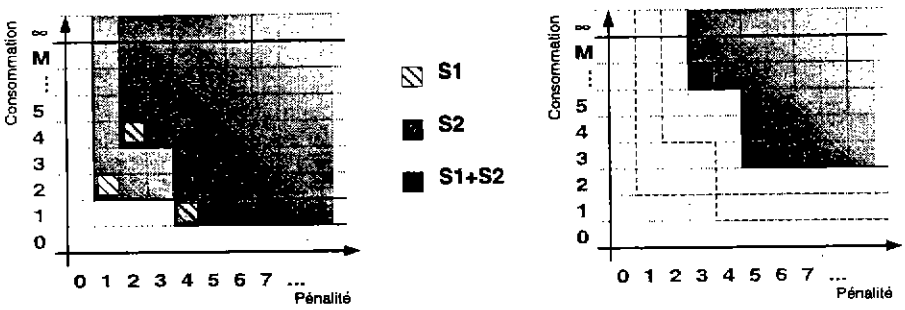

FIG. 5.5 – *Agrégation de deux valuations*

Propriété 5.7 (isomorphisme) *La fonction v est un isomorphisme monotone de l'ensemble des coûts vers l'ensemble des valuations unitaires.*

Démonstration : (5.7) *v est une bijection par construction.*

Soit ς_1 *et* ς_2 *deux coûts,*

$$\begin{aligned} v(\varsigma_1) \oplus v(\varsigma_2) =& \{\varsigma \text{ tels que } \exists \varsigma_1' \in v(\varsigma_1), \varsigma_2' \in v(\varsigma_2) / \varsigma \geq \varsigma_1' + \varsigma_2'\} \\ \Rightarrow v(\varsigma_1) \oplus v(\varsigma_2) =& \{\varsigma \text{ tels que } \exists \varsigma_1' \geq \varsigma_1, \varsigma_2' \geq \varsigma_2 / \varsigma \geq \varsigma_1' + \varsigma_2'\} \\ \Rightarrow v(\varsigma_1) \oplus v(\varsigma_2) =& \{\varsigma \text{ tels que } \varsigma \geq \varsigma_1 + \varsigma_2\} \\ \Rightarrow v(\varsigma_1) \oplus v(\varsigma_2) =& v(\varsigma_1 + \varsigma_2) \end{aligned}$$

Soit ς_1, ς_2 *deux coûts tels que* $\varsigma_1 \leq \varsigma_2$. $\{\varsigma \geq \varsigma_1\} \supset \{\varsigma \geq \varsigma_2\}$ *donc* $v(\varsigma_1) \leq v(\varsigma_2)$.

Cette propriété nous permet d'affirmer que les deux représentations des valuations bicritères sont parfaitement équivalentes. Plus précisément, les valuations telles que nous venons de les définir intègrent la notion plus naturelle des coûts. Cette définition permet cependant de modéliser des valuations plus complexes que les simples paires pénalité-consommation sous la forme de surfaces engendrées par plusieurs coûts. Un telle surface doit être considérée comme une disjonction entre plusieurs coûts possibles. Elle se construit à partir de la définition du minimum de deux valuations. Intuitivement, si l'on a le choix entre deux valuations, soit si les coûts possibles appartiennent à l'un ou l'autre des deux ensembles, alors les coûts possibles appartiennent certainement à l'union des deux ensembles.

Nous définissons donc formellement le minimum ci-dessous. Il est à noter que cette définition du minimum s'adapte à une structure de valuation ne possédant qu'un ordre partiel et répond aux réserves formulées à l'occasion des développements que nous avons marqués d'un "†".

Définition 5.8 (minimum) *Le minimum d'un ensemble de valuations est la valuation engendrée par l'union des ensembles de coûts engendrant les valuations. En d'autres termes,* $\min_i v_i = \bigcup_i v_i$.

Définition 5.9 (maximum) *Le maximum d'un ensemble de valuations est l'intersection des ensembles de coûts définissant les valuations :* $\max_i v_i = \bigcap_i v_i$.

La relation d'ordre étant définie par l'inclusion, il était naturel que le minimum soit l'union et le maximum l'intersection : notre structure correspond à une structure ensembliste.

Nous avons maintenant en main tous les éléments pour définir la structure de valuation que nous allons utiliser : $\mathbb{S} = (S, \preceq, v(0,0), v(\infty,\infty), \oplus)$ où $\preceq$ et $\oplus$ ont été définis plus haut. Il ne reste qu'à s'assurer de la monotonie de $\oplus$ par rapport à $\prec$.

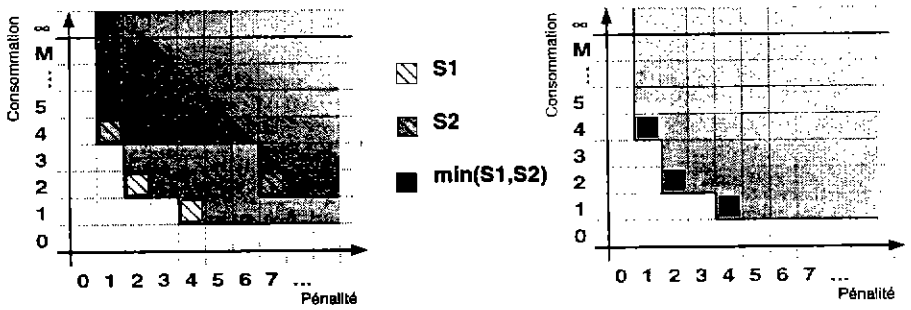

FIG. 5.6 – *Minimum*

Propriété 5.10 *Soit s, s_1 et s_2 trois éléments de S, si $s_1 \preceq s_2$ alors $s_1 \oplus s \preceq s_2 \oplus s$*[7].

Démonstration : (5.10) $s_2 \oplus s = \{\varsigma$ *tels que* $\exists \varsigma_2 \in s_2, \varsigma' \in s / \varsigma_2 + \varsigma' \leq \varsigma\}$. *Or* $\varsigma_2 \in s_2 \Rightarrow \varsigma_2 \in s_1$. *Donc tout* ς *de* $s_2 \oplus s$ *appartient aussi à* $s_1 \oplus s$.

Formulation des contraintes

Si l'utilisation de la mémoire est un critère, sa consommation doit être perçue comme une contrainte, au même titre que les contraintes unaires portant sur les rejets. Nous sommes donc amenés à poser sur chaque prise de vue une contrainte unaire dont la valuation correspond à l'espace mémoire consommé par la prise de vue.

Notre problème se compose donc d'un ensemble de demandes d'images . Une demande d'image x_k d'une importance p_k peut être réalisée par un ensemble de prises de vue x_k^i, chacune nécessitant un espace mémoire de taille m_k^i, ou être rejetée (x_k^*). Une contrainte unaire c_k^i de valuation $\phi(c_k^i) = v(0, m_k^i)$ est posée sur chaque prise de vue ; une contrainte unaire c_k^* de valuation $\phi(c_k^*) = v(p_k, 0)$ est posée sur le rejet.

La multiplication des contraintes pondérées est regrettable pour des questions de gestion des justifications. Il est cependant possible de considérer une unique contrainte dont la valuation dépend du n-uplet interdit. Nous pouvons nous permettre cette simplification dans la mesure où nous n'utilisons pas les développements concernant l'opérateur d'agrégation optimal. Les remarques ‡ ont précisé, lorsqu'une ambiguïté se présentait, les dispositions à prendre : d'une manière générale, lorsque la valuation n'est pas imposée par une violation, il faut toujours considérer la valuation la plus défavorable (le minimum ou

7. Il est implicitement affirmé que les membres sont comparables.

le maximum selon les cas). Cette solution présente le l'inconvénient de noyer quelque peu les justifications.

5.3.3 Propositions

Nous nous proposons donc d'utiliser un Nogood Recording avec la structure de valuation que nous venons de définir. Le choix de l'algorithme provient de la nature des contraintes associées aux instruments. La structure de valuation bicritère doit quant à elle permettre de prendre efficacement en compte la contrainte mémoire.

Certaines modifications de l'algorithme du Nogood Recording valué correspondant aux remarques formulées lors de sa définition doivent être apportées. En effet, avec un ordre partiel, plusieurs possibilités d'optimisation se présentent.

On peut choisir de n'accepter que les solutions qui améliorent strictement la borne supérieure[8] ou bien tolérer aussi celles qui ne sont pas comparables à la borne supérieure. La première possibilité n'est pas acceptable pour notre application. En effet, la consommation mémoire n'est pas réellement un critère à optimiser. Nous ne la considérons comme un critère que pour récupérer les capacités d'analyse et de prévision liées aux nogoods. Mais il ne faut en aucun cas entendre par là que l'on souhaite des solutions consommant aussi peu de mémoire que possible. Pour être certain d'obtenir toutes les solutions optimales indépendamment de leur consommation, il est nécessaire de tolérer l'obtention de solutions incomparables.

Ceci pose aussi le problème du réajustement de la borne maximale lorsqu'une nouvelle solution acceptable est découverte. Intuitivement, il ne faut accepter aucune solution de valuation supérieure ou égale à la valuation de l'une des anciennes solutions. On ne tolérera donc que les solutions de valuation inférieure au minimum des valuations des solutions précédentes. La mise à jour doit donc se faire en affectant la borne supérieure au minimum de celle-ci et de la nouvelle solution. La décroissance stricte de la borne supérieure reste garante de la terminaison du processus.

Cependant, selon l'effet recherché, il peut être intéressant d'être plus restrictif. On peut par exemple ne tolérer que les solutions dont la pénalité est strictement inférieure aux pénalités des solutions précédentes. α est de la forme $v(p_\alpha, 0)$, et si $v(p, m)$ est une

8. Cette option a été implicitement choisie lors de la définition du Nogood Recording valué.

nouvelle solution, $\alpha \leftarrow v(min(p_\alpha, p), 0)$.

La première solution propose une recherche plus exhaustive et plus complète car elle recherche la pénalité optimale pour chaque niveau de consommation[9]. De la seconde solution découle une recherche plus réduite tout à fait équivalente à celle que l'on obtiendrait avec une contrainte mémoire naturelle et qui n'offre comme résultat qu'une solution de pénalité optimale. Cependant, rien ne permet d'affirmer que la seconde possibilité est plus intéressante. Comme nous l'avons vu, la mémorisation des solutions est capitale pour l'efficacité de la recherche : le temps perdu à mémoriser des bouts de solutions est très largement compensé par l'intérêt de leur réutilisation. La première solution est à ce titre préférable puisque l'exhaustivité de l'optimisation lui impose de produire des nogoods dont l'information est plus complète. On peut rapprocher cette possibilité des techniques produisant de manière artificielle plus de contraintes induites que nécessaire [SV94b].

Algorithme

SNR(A)

- **SNR-fc**(A)
- Si A est complète alors $\alpha \leftarrow \min(\alpha, v(A))$
- Sinon
  - Soit $x \notin \tilde{A}$, $N = \emptyset$
  - Pour chaque valeur i de x,
    - $n \leftarrow$**SNR-induite**$(A \cup \{x = i\})$
    - Si $\neg(\phi(n) \succeq \alpha)$ alors $n \leftarrow$**SNR**$((A \cup \{x = i\}))$
    - (Si $x^i \notin n$ alors renvoie n :backjumping)
    - $N \leftarrow N \cup \{n\}$
  - $n_{union} \leftarrow \bigvee_{n \in N} n$
  - Mémorise $\Downarrow (n_{union})$ et renvoie n_{union}

SNR-fc(A)

- S'il existe une variable $x \notin \tilde{A}$ telle que ϕ (**SNR-induite**$(A \cup x^i) \succeq \alpha) \forall x^i \neq x^*$ alors
  - $A \leftarrow A \cup \{x^*\}$
  - **SNR-fc**(A)

9. Ce qui est équivalent à la consommation minimale pour chaque niveau de pénalité.

SNR-induite(A)

- Si $\phi(\mathbf{BT-induite}(A)) \succeq \alpha$
  Renvoie **BT-induite**(A)
- S'il existe un nogood n violé par A tel que $\phi(\Uparrow^A (n)) \succeq \alpha$
  Renvoie $\Uparrow^A (n)$
- Sinon
  Renvoie $\max_{A \text{ viole } n} \Uparrow^A (n)$

Nous préférons détailler la fonction **SNR-induite** en plusieurs test successifs bien qu'inclus afin de prendre en compte les remarques sur l'efficacité des nogoods et les risques de noyade.

Quelques chiffres

Nous reprenons ici les meilleurs algorithmes "NR-dbt-0" et "NR*-dbt-0" sur les jeux d'essai sans contrainte mémoire. L'objectif est de fournir la courbe complète des couples pénalité-consommation optimaux. Lorsque le résultat est obtenu en moins de 30 minutes, nous utilisons comme précédemment le signe "*". Dans tous les cas, nous donnons le temps (en secondes) nécessaire à l'obtention de la meilleure pénalité trouvée en 30 minutes de recherche.

Etant donné le ralentissement induit par la structure de valuation plus complexe, nous testons aussi deux algorithmes "NR-dbt-00" et "NR*-dbt-00", obtenus par simplification des deux premiers :

- le Forward-checking est abandonné, afin de limiter les tests devenus coûteux et d'éviter des calculs sur les redondances de justification.
- un tri préalable est effectué sur les nogoods utilisés pour estimer les affectations partielles. Plus précisément, l'examen de l'extension avec une variable x ne prend en compte que les nogoods produits à l'occasion des échecs antérieurs des extensions avec x.

Lorsque l'ordre d'affectation des variables est statique, ces modifications nous dispensent de gérer les justifications. En effet, un nogood construit lors de l'examen d'une variable x ne concerne (après réduction) que des contraintes liant les successeurs de x. Lors d'un

nouvel examen de x l'affectation courante ne risque pas de violer des contraintes du nogood : la valuation locale de l'affectation peut être ajoutée à la valuation du nogood. Nous sommes alors très proches des Poupées Russes.

Problèmes sur une orbite : exemples									
Pb	n	NR-dbt-0		NR*-dbt-0		NR-dbt-00		NR*-dbt-00	
11	364	41490	1378	40558	1706	40484	1721	*38478	383
404	100	116	197	*114	317	*114	8	*114	4
408	200	6234	730	6274	1621	6234	1328	*6228	950
412	300	34387	281	32442	1796	34387	184	*32381	1141
503	143	11117	517	11143	1758	*11113	517	12115	8
505	240	22260	128	22310	1489	21259	121	*21253	272
507	311	29405	1046	30489	1638	29405	274	*27390	433
509	348	39460	889	38520	1738	39457	1306	*36446	1633

TAB. 5.4 – *Résultats sur une orbite*

Problèmes sur une orbite : statistiques		
Nombre de Variables	Nombre de problèmes	% résolution < 30min NR*-dbt-00
- 100	267	99,6%
100 - 200	61	63,9%
200 - 300	38	55,3%
300 - 400	13	53,8%
400 -	4	0,0%
Total	383	86,9%

TAB. 5.5 – *Résultats statistiques sur une orbite*

A l'évidence, une recherche exhaustive ne convient pas du tout au Nogood Recording naturel. Néanmoins, pour ces problèmes où la contrainte mémoire n'intervient pas, il suffirait d'appliquer la stratégie d'optimisation de la pénalité seulement pour retrouver les résultats obtenus avant l'introduction d'un critère multiple.

Un second point évident est l'importance capitale des simplifications que nous avons apportées. Elles doivent néanmoins être relativisées dans la mesure où, d'une part, elles imposent un ordre statique des variables et, d'autre part, leur nécessité est liée à l'optimisation de l'implémentation.

Enfin, la version "NR*-dbt-00" se détache nettement dans la mesure où elle est capable de fournir la valuation bicritère optimale pour des problèmes comportant plus de 300 variables. La modélisation des valuations que nous proposons permet, avec une utilisation pertinente des mémorisations, de prendre efficacement en compte une contrainte de forte arité telle que la mémoire. Mais nous atteignons là les limites de notre algorithme alors que les problèmes avec contrainte mémoire sont encore plus importants.

5.3.4 Coopération

Les deux versions du Nogood Recording, opportuniste et exhaustif, voient leur complémentarité mise en relief par l'introduction de la contrainte mémoire et des valuations complexes.

Les versions exhaustives et rigides parviennent à faire progresser régulièrement leur optimisation, mais elles doivent pour cela accepter une certaine rigidité dans leur recherche. Cela implique qu'elles ne fournissent un résultat intéressant qu'au terme de leur recherche (lors de la résolution du dernier sous-problème). Il est à craindre que ce défaut devienne rédhibitoire pour les gros problèmes, si ces versions ne terminent pas dans un temps raisonnable.

A l'inverse, les versions opportunistes peuvent profiter de leur liberté pour atteindre rapidement des solutions intéressantes, grâce notamment à des heuristiques du type "meilleur rapport pénalité/consommation d'abord". Leur manque de rigueur dans la construction des nogoods les rend cependant très rapidement incapables de progresser et encore moins de prouver l'optimalité des solutions.

Il nous a donc semblé intéressant de profiter d'une des propriétés des nogoods, la réutilisabilité, afin de faire profiter une recherche opportuniste des acquis d'une recherche systématique. La version finale de notre algorithme s'inspire de [MV96] et effectue deux recherches simultanées coopérant par l'échange des nogoods produits. Dans notre cas, l'échange ne se fait que d'une recherche vers l'autre. Il faut en effet considérer que la recherche exhaustive produit tous les nogoods qu'il est intéressant de produire au fur et à mesure de ses besoins. A l'inverse, une recherche opportuniste est rarement en mesure de produire le détail des solutions optimales sur les sous-problèmes réduits aux dernières variables.

Choix des versions

Nous utilisons donc la version "NR*" avec une stratégie de recherche produisant tous les couples pénalité-consommation optimaux pour les sous-problèmes imbriqués. Au fur et à mesure de sa production, cette version envoie ses nogoods à un second algorithme sur le modèle de "NR". L'objectif de ce-dernier est de trouver au plus tôt les bonnes solutions. Sa stratégie de recherche consiste donc à améliorer uniquement la pénalité de ses solutions intermédiaires.

Au vu des résultats précédents, l'algorithme systématique sera plus précisément la version "NR*-dbt-00", utilisé avec un ordre chronologique d'affectation des variables.

Le choix de la version opportuniste est plus difficile. En effet, cette dernière doit profiter des possibilités qu'offre le choix d'un ordre quelconque sur les variables mais reçoit des nogoods construits selon un odre d'affectation éventuellement différent. Auquel cas, les simplifications proposées précédemment perdent de leur pertinence :

- il est impératif de gérer les justifications mais ;
- il est certainement utile de considérer les nogoods indépendamment de la variable sur laquelle ils ont été posés. Ils ne reflètent pas nécessairement une prévision sur le sous-problème constitué par les variables qui suivent celle sur laquelle ils sont posés ;
- enfin, lorsque l'on n'utilise plus un ordre d'affectation chronologique, le Forward-checking redevient intéressant puisqu'il peut être amené à détecter des conflits entre variables éloignées dans l'ordre d'affectation.

Nous testerons donc les deux versions, "NR-dbt-0" et "NR-dbt-00", selon que les ordres sur les variables sont communs ou non aux deux algorithmes coopérants. Dans tous les cas, la stratégie utilisée consistera à améliorer uniquement la pénalité des solutions.

Les ordres sur les variables que nous utiliserons sont de deux types : chronologique et meilleur rapport pénalité/consommation d'abord.

Résultats

Notre algorithme est appelé "**NR+NR***", nous en testons plusieurs versions selon l'utilisation d'heuristiques :

- "**NR+NR***" correspond à la coopération de "NR-dbt-00" et "NR*-dbt-00" avec un ordre chronologique ;
- "**NR^h+NR***" : "NR-dbt-0" utilisé avec l'heuristique de meilleur rapport pénalité/consommation d'abord et "NR*-dbt-00" avec un ordre chronologique ;
- "**NR^h+NR*^{h}**" "NR-dbt-00" et "NR*-dbt-00" tous deux avec l'heuristique meilleur rapport pénalité/consommation d'abord.

Les performances de ces algorithmes sont comparées aux résultats obtenus dans [BVA+96] avec les algorithmes de Forward-checking "**DFBB**" et de recherche tabou "**TS**". Nous donnons aussi les performances de notre Forward-checking "**FC^h**" amélioré par l'heuristique de choix des variables selon le rapport pénalité/consommation. Comme d'habitude, la meilleure pénalité obtenue ainsi que le temps nécessaire sont donnés.

Problèmes avec contrainte mémoire : exemples													
Pb	n	FC^h		NR+NR*		NR^h+NR*		NR^h+NR*^{h}		DFBB		TS	
1021	1057	481637	1	497593	78	481596	65	481596	8	468416	1988	459399	2197
1401	488	481103	35	487095	1080	473097	58	473097	149	470109	648	461109	846
1403	665	481256	117	497252	57	481251	682	481250	1116	468416	1867	459463	1324
1405	855	481452	1	497456	312	481467	642	481467	824	470647	1342	461630	1574
1502	209	29042	1	*28042	1	*28042	1	*28042	1	28045	601	28045	454

TAB. 5.6 – *Résultats sur plusieurs orbites*

Conclusions

Le principe de mémorisation est primordial dans la résolution des problèmes d'ordonnancement soulevés par notre application.

Nous avons montré comment il est possible de résoudre les problèmes sans contrainte mémoire à l'aide de l'algorithme du Nogood Recording valué. Ce dernier permet d'obtenir des résultats optimaux sur des problèmes dépassant les 300 variables et 10 000 contraintes.

Par comparaison aux performances obtenues avec d'autres méthodes de résolution, notre implémentation pourtant rudimentaire s'avère généralement plus rapide. Le Nogood Recording constitue une alternative intéressante des Poupées Russes, tant pour son efficacité que pour sa souplesse.

L'introduction de la contrainte mémoire constitue toujours un point difficile qui interdit la résolution optimale des problèmes d'ordonnancement. Les valuations bicritères que nous avons définies permettent cependant d'affirmer que la difficulté provient plus de la taille des problèmes où intervient la contrainte mémoire que des restrictions liées à la mémoire. En effet, nos développements ont permis de gérer efficacement cette contrainte avec comme résultat la résolution optimale des problèmes posés sur une orbite, résolution qui fournit pour chaque niveau de consommation l'ordonnancement de meilleure pénalité. Par contre, les problèmes plus larges regroupant plusieurs orbites ne sont pas accessibles.

Nos tentatives pour obtenir, faute d'optimalité, de "bonnes solutions"par une résolution coopérative sont décevantes. La réduction de la recherche, apportée par les mémorisations, compense difficilement les lourdeurs de l'implémentation. Les résultats obtenus avec des méthodes de recherche locale restent les plus intéressants. Il semble que, pour cette application, les recherches optimales ne valent que si elles sont en mesure de terminer.

Il ne faut cependant pas être trop négatif car la résolution optimale bicritère sur une orbite est un résultat important. Il est en effet possible de décomposer les problèmes d'ordonnancement en sous-problèmes indépendants du point de vue des instruments et reliés uniquement par la contrainte mémoire. Chacun de ces sous-problèmes est de taille inférieure aux problèmes sur une orbite et doit pouvoir être résolu de manière optimale. La reconstitution d'une solution optimale globale à partir des courbes optimales de chaque sous-problème est alors aisée : elle est obtenue par la somme des valuations bicritères optimales.

Chapitre 6

Conclusion, générale

Notre objectif était d'étendre aux CSP valués dont l'opérateur d'agrégation n'est pas idempotent les principes de mémorisation utilisés dans le cadre classique. Ces-derniers s'appuient sur la notion de contrainte induite qu'il nous fallait donc adapter au cadre valué. C'est maintenant chose faite.

Avec la définition des contraintes induites, composées d'une affectation, d'une valuation d'interdiction, et d'une justification sous forme d'un ensemble de contraintes, il est possible d'étendre tous les algorithmes qui exploitent des contraintes induites : k-cohérence, Nogood Recording, etc.

L'agrégation des contraintes induites est le point difficile. Il impose un compromis entre la qualité du résultat de l'agrégation et la complexité des calculs. En particulier, seules les versions simplifiées s'avèrent réalistes et, parmi celles-ci, seules les méthodes qui ne produisent pas de manière intensive les contraintes induites sont en mesure d'offrir une alternative au Forward-checking qui soit compétitive sur les VCSP aléatoires.

L'efficacité des contraintes induites dans le cadre des CSP valués, en terme de restriction de l'espace exploré, n'est cependant pas remise en cause. La réduction de l'espace de recherche est en général significative. Aussi peut-on toujours espérer, par des simplifications judicieuses, par une optimisation des calculs ou par une adéquation particulière à une application, retrouver des performances intéressantes en terme de temps de calcul.

En l'occurrence, le problème d'ordonnancement des prises de vue d'un système satellitaire d'observation se prête bien à l'utilisation de contraintes induites. Ces dernières permettent de résoudre des problèmes de taille très importante qui mettent en échec les

méthodes classiques de résolution. Sur cette application pratique comme pour les VCSP aléatoires, la simplification de l'utilisation des contraintes induites est souvent payante[1].

Les performances restent cependant insuffisantes pour répondre totalement au besoin des opérateurs de tels systèmes. En particulier, les contraintes fortement cumulatives interdisent la résolution directe des plus gros problèmes. Une issue réaliste est toutefois proposée grâce à l'utilisation d'une structure de valuation bicritère qui, combinée aux mémorisations, permet de diviser ces problèmes en sous-problèmes de taille accessible.

Nous n'avons pas exploré les possibilités offertes par les contraintes induites dans une perspective de réutilisation d'une résolution à l'autre, lorsque l'on traite des séries de problèmes proches. Le mémorisation est certainement d'une grande utilité dans ces conditions.

Nous n'avons pas non plus cherché à remédier au problème majeur des contraintes induites que pose la noyade des justification lors des agrégations. Une amélioration plus calculatoire du modèle très simple que nous avons utilisé peut s'avérer intéressante.

Dans le cas général, il nous semble que les contraintes induites ont montré leurs limites dans le cadre des CSP valués : théoriquement efficaces mais trop lourdes en pratique. Il est probablement plus pertinent de chercher à traiter les justifications par une gestion *a priori* des intersections plutôt que d'effectuer des calculs permanents.

1. La simplification extrême est constituée par les Poupées Russes.

Annexe A

Lexique

Variables	
x, y, z	: une variable ;
$\mathbb{V}$	: l'ensemble de toutes les variables ;
x^i	: la valeur i de la variable x ;
A	: une affectation partielle ;
$\tilde{A}$	: l'ensemble des variables affectées dans A.

Contraintes	
c	: une contrainte ;
$\mathcal{C}$	: un ensemble de contraintes ;
$\mathbb{C}$	: l'ensemble de toutes les contraintes ;
$\tilde{c}$	: l'ensemble des variables liées par la contrainte c ;
$\phi(c)$	: la valuation de c.

Contraintes induites	
n, ζ	: un nogood ou une contrainte induite ;
$\mathcal{I}$	: un ensemble de contraintes induites ;
$\mathbb{I}$	: l'ensemble de toutes les contraintes induites ;
$\phi(\zeta)$	: la valuation de ζ ;
$\bigvee_i(\zeta_i)$	: l'union des contraintes induites ζ_i (propriétés 2.8 et 3.5) ;
$\uparrow^{\mathcal{C}}(\zeta)$	: augmentation de ζ avec les contraintes de $\mathcal{C}$;
$\Uparrow^{A}(\zeta)$	: augmentation de ζ avec les contraintes violées par A ;
$\downarrow_{\mathcal{C}}(\zeta)$	: réduction de ζ avec les contraintes de $\mathcal{C}$;
$\Downarrow(\zeta)$	: réduction de ζ avec les contraintes violées par son affectation.

Valuations	
E	: l'ensemble des valuations ;
$\preceq$	: la relation d'ordre sur E ;
$\oplus$	: l'opérateur d'agrégation de E^2 vers E ;
$\ominus$	: l'opérateur de soustraction de E^2 vers E ;
$v(A)$	: la valuation locale de A (cf section 2.2.1) ;
$\odot$	: l'opérateur d'agrégation des contraintes induites de $\mathbb{I}^n$ vers $\mathcal{E}$ (cf section 3.1.3) ;
$v_{\odot}(A)$	: la valuation locale induite de A (cf section 3.1.2).

Procédures	
xx-Evalue	: procédure d'évaluation de la cohérence globale de l'algorithme xx ;
XX-Evalue	: procédure d'évaluation de la valuation globale de l'algorithme XX ;
XX-induite	: procédure d'évaluation de la valuation globale par production d'une contrainte induite (algorithme XX).

Application	
x	: une demande d'image ;
x^i	: la $i^{ème}$ prise de vue permettant de réaliser l'image x ;
x^*	: le rejet de la prise de vue x ;
c_x^*	: la contrainte représentant la pénalité associée au rejet de x ;
c_x^i	: la contrainte représentant la consommation de la prise de vue x^i ;
p	: une pénalité $\in \mathbb{N}$;
m	: une consommation $\in [0, M] \cup \{\infty\}$;
M	: la capacité mémoire ;
ς	: un coût, soit une paire (p, m) ;
$\mathbb{N}_2$	: l'ensemble des coûts ;
s	: une valuation bicritère (soit un ensemble de coûts) ;
S	: l'ensemble des valuations ;
$v(c)$	: la valuation unitaire engendrée par le coût ς.

Exemple

5 variables $X_1 ... X_5$ de domaine $\{a, b\}$ ou $\{a, b, c\}$;

5 contraintes binaires $C_1 ... C_5$ de la forme $X_i > X_j$;

une structure de valuation additive et une valuation 1 pour chaque contrainte.

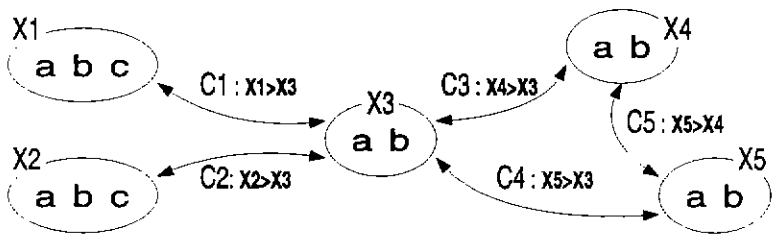

FIG. A.1 – *Exemple.*

Annexe B

Espérance de solutions

Soit un CSP binaire défini par :

- n variables ;
- d valeurs par variable ;
- C contraintes (soit une connectivité du graphe de contrainte *con* telle que $C = con \times \frac{n.(n-1)}{2}$) ;
- une dureté *dur* des contraintes, rapport entre le nombre de n-uplets autorisés sur le nombre de n-uplets possibles (d^2 en l'occurrence).

La probabilité qu'une contrainte soit satisfaite par une affectation complète est $1 - dur$. Si l'on suppose les violations de contraintes indépendantes, la probabilité qu'une affectation satisfasse toutes les contraintes est $(1 - dur)^C$.

Sachant qu'il y a d^n affectations complètes possibles, une estimation de l'espérance du nombre de solutions est $E = d^n.(1 - dur)^C$.

Cette petite équation permet de prédire avec une bonne précision la limite entre les problèmes aléatoire avec et sans solution. Elle se situe lorsque $E = 1$.

Plaçons-nous dans le cadre valué et intéressons-nous au nombre x de contraintes dont on autorise la violation. La probabilité pour qu'une affectation complète satisfasse toutes les contraintes sauf x (sous lhypothèse dindépendance) est le produit du nombre d'ensembles de x contraintes parmi C (C_n^x) par la probabilité que x contraintes données soient violées (dur^x) et les autres satisfaites ($(1 - dur)^{(C-x)}$): $C_n^x.dur^x.(1 - dur)^{(C-x)}$.

D'où l'estimation de lespérance du nombre de solutions violant x contraintes :
$E(x) = d^n \times C_n^x.dur^x.(1-dur)^{(C-x)}$.

Plus précisément, pour les jeux d'essai qui nous intéressent :
$E(2) = d^n \times \frac{C.(C-1)}{2}.dur^x.(1-dur)^{(C-x)}$.

Après avoir fixé le nombre de variables et la taille de leur domaine, nous utilisons cette équation pour déterminer la dureté en fonction de la connectivité ou vice-versa.

Annexe C

Algorithmes détaillés

Cette annexe regroupe les principaux algorithmes dédiés aux CSP classiques tels qu'ils sont décrits dans [VS95], ainsi que le Forward-checking pour CSP valués.

C.1 CSP Classiques

Nous utilisons dans ce chapitre quelques fonctions pour manipuler les CSP :

- **VARIABLES**(p) retourne l'ensemble des variables d'un problème p ;
- **DOMAINE**(x) retourne l'ensemble des valeurs possibles de la variable x ;
- **CONTRAINTES**(x) retourne l'ensemble des contraintes qui pèsent sur la variable x.

C.1.1 Le Backtrack

Les fonctions de plus haut niveau sont décrites en figure C.1. Celles de plus bas niveau répondent aux spécifications suivantes :

- **CHOIX-VARIABLE**($\mathcal{V}$) retourne une variable de l'ensemble de variables $\mathcal{V}$;
- **CHOIX-VALEUR**(d) retourne une valeur de l'ensemble de valeurs d ;

- **COHERENCE**$(\mathcal{C}, A)$ retourne $FAUX$ si une contrainte de $\mathcal{C}$ est insatisfaite par l'affectation partielle A et retourne $VRAI$ sinon.

C.1.2 Le Forward-checking

Outre les procédures de haut niveau que nous détaillons en figure C.2, il nous faut quelques fonctions supplémentaires définies comme suit :

- **CONTRAINTES**(x, y) retourne l'ensemble des contraintes qui pèsent sur les deux variables x et y à la fois ;
- **VOISINES**$(x, \mathcal{V})$ retourne l'ensemble des variables de $\mathcal{V}$ reliées par une contrainte à la variable x ;
- **ENLEVE**(i, x) supprime la valeur i du domaine de la variable x ;
- **EMPILE**$(\mathcal{V})$ empile en mémoire les domaines courants des variables de l'ensemble de variables $\mathcal{V}$;
- **DEPILE**$(\mathcal{V})$ rétablit les domaines des variables de $\mathcal{V}$ en les dépilant de la mémoire.

Le Forward-checking s'obtient à partir du Backtrack en remplaçant la fonction **BT-VALEUR** par **FC-VALEUR**.

C.1.3 Conflict Directed Backjumping

Nous utilisons les fonctions supplémentaires suivantes :

- **VARIABLES**$(\mathcal{C})$ retourne l'ensemble des variables liées par les contraintes de l'ensemble $\mathcal{C}$;
- **ABSENTE**$(x, \mathcal{V})$ retourne $VRAI$ si la variable x n'appartient pas à l'ensemble $\mathcal{V}$ et $FAUX$ sinon ;
- **ANALYSE-COHERENCE**$(\mathcal{C}, A)$ retourne l'ensemble vide si aucune contrainte de $\mathcal{C}$ n'est violée par A et retourne le singleton constitué d'une contrainte instatisfaite dans le cas contraire.

Les fonctions de haut niveau sont données par la figure C.3.

Backtrack(p)

renvoie **BT-VARIABLES**($\emptyset$, **VARIABLES**(p))

BT-VARIABLES($A, \mathcal{V}$)

: A est l'affectation partielle d'un ensemble de variables

: $\mathcal{V}$ est l'ensemble des variables non affectées

si $\mathcal{V} = \emptyset$
alors renvoie *succes*
sinon, soit $x =$ **CHOIX-VARIABLES**($\mathcal{V}$)
renvoie **BT-VARIABLE**($A, \mathcal{V}, x,$ **DOMAINE**(x))

BT-VARIABLE($A, \mathcal{V}, x, d$)

si $d = \emptyset$
renvoie *échec*
sinon, soit $i =$ **CHOIX-VALEUR**(d)
si **BT-VALEUR**($A, \mathcal{V}, x, i$) = *succès*
alors renvoie *succès*
sinon renvoie **BT-VARIABLE**($A, \mathcal{V}, x, d - \{i\}$)

BT-VALEUR($A, \mathcal{V}, x, i$)

si **COHERENCE**(**CONTRAINTES**(x), $A \cup \{x = i\}$)
alors renvoie **BT-VARIABLES**($A \cup \{x = i\}, \mathcal{V} - \{x\}$)
sinon renvoie *échec*

FIG. C.1 – *Backtrack*

FC-VALEUR$(A, \mathcal{V}, x, i)$

EMPILE$(\mathcal{V})$
si **COHERENCE**(**CONTRAINTES**$(x), A \cup \{x = i\}$)
et si **FC-COHERENCE-VARIABLES**$(A \cup \{x = i\}, x,$ **VOISINES**$(x, \mathcal{V}))$
alors, *poursuite* $\leftarrow$ **BT-VARIABLES**$(A \cup \{x = i\}, \mathcal{V} - \{x\})$
sinon *poursuite* $\leftarrow$ *échec*
DEPILE$(\mathcal{V})$
renvoie *poursuite*

FC-COHERENCE-VARIABLES$(A, x, \mathcal{V})$

: A est l'affectation partielle d'un ensemble de variables

: $\mathcal{V}$ est l'ensemble des variables non affectées

si $\mathcal{V} = \emptyset$
alors renvoie $VRAI$
sinon, soit $y =$ **CHOIX-VARIABLES**$(\mathcal{V})$
si **FC-COHERENCE-VARIABLE**$(A, x, y,$ **DOMAINE**$(y))$
alors renvoie **FC-COHERENCE-VARIABLES**$(A, x, \mathcal{V} - \{y\})$
sinon renvoie $FAUX$

FC-COHERENCE-VARIABLE(A, x, y, d)

si $d = \emptyset$
alors renvoie $FAUX$
sinon, soit $j =$ **CHOIX-VALEUR**(d)
soit *autres* = **FC-VARIABLE**$(A, x, y, d - \{j\})$
si **COHERENCE**(**CONTRAINTES**$(x, y), A \cup \{y = j\}$)
alors
ENLEVE(j, y)
renvoie **FC-VARIABLE**$(A, x, y, d - \{j\})$
sinon
FC-VARIABLE$(A, x, y, d - \{j\})$
renvoie $VRAI$

FIG. C.2 – *Forward-checking*

Conflict-directed-backjumping(p)

renvoie **CBJ-VARIABLES**($\emptyset$, **VARIABLES**(p))

CBJ-VARIABLES($A, \mathcal{V}$)

: A est l'affectation partielle d'un ensemble de variables

: $\mathcal{V}$ est l'ensemble des variables non affectées

si $\mathcal{V} = \emptyset$
alors renvoie $\emptyset$
sinon, soit $x =$ **CHOIX-VARIABLES**($\mathcal{V}$)
renvoie **CBJ-VARIABLE**($A, \mathcal{V}, \emptyset, x,$ **DOMAINE**(x))

CBJ-VARIABLE($A, \mathcal{V}, \mathcal{C}x, d$)

si $d = \emptyset$
alors renvoie $\mathcal{C} \cup$ **DOMAINE**(x)
sinon, soit $i =$ **CHOIX-VALEUR**(d)
soit $\mathcal{C}' =$ **CBJ-VALEUR**($A, \mathcal{V}, x, i$)
si $\mathcal{C}' = \emptyset$
alors renvoie $\emptyset$
sinon
si **ABSENT**($x,$ **VARIABLES**($\mathcal{C}'$))
alors renvoie $\mathcal{C}'$
sinon renvoie **CBJ-VARIABLE**($A, \mathcal{V}, \mathcal{C} \cup \mathcal{C}', x, d - \{i\}$)

CBJ-VALEUR($A, \mathcal{V}, x, i$)

soit $\mathcal{C} =$ **ANALYSE-COHERENCE**(**CONTRAINTES**(x), $A \cup \{x = i\}$)
si $\mathcal{C} = \emptyset$
alors renvoie **CBJ-VARIABLES**($A \cup \{x = i\}, \mathcal{V} - \{x\}$)
sinon renvoie $\emptyset$

FIG. C.3 – *Conflict Directed Backjumping*

C.1.4 Le Nogood Recording

La notion de contrainte induite intervient dans cet algorithme et par suite celle de justification. Cet ensemble de contraintes initiales du problème est explicitement défini lors de la création d'une nouvelle contrainte. Pour une contrainte initiale, il correspond implicitement au singleton constitué de cette même contrainte.

Nous utilisons les fonctions suivantes :

- **CREE-NOGOOD**$(A, \mathcal{C})$ ajoute une contrainte définie comme suit :
  - le n-uplet interdit correspond à la projection de l'affectation A sur les variables de **VARIABLES**$(\mathcal{C})$;
  - $\mathcal{C}$ constitue la justification.
- La fonction **ANALYSE-COHERENCE**(d)oit être modifiée pour retourner, en cas de violation, non pas une contrainte violée, mais sa justification.

L'algorithme est donné figure C.4.

C.2 CSP Valués

C.2.1 Le Branch and Bound en profondeur d'abord

Les fonctions de plus haut niveau sont décrites en figure C.5. Outre les fonctions de bas niveau définies dans la section précédente, il nous faut définir la fonction permettant d'extraire la valuation d'une affectation :

- **VALUATION**$(\mathcal{C}, A)$ retourne l'agrégation (opérateur "$\oplus$") des valuations des contraintes de $\mathcal{C}$ qui sont violées par A. La valuation minimale correspondant à la satisfaction de toutes les contraintes est "$\bot$".

C.2.2 Le Forward-checking valué

Le Forward-checking valué s'obtient à partir du Branch and Bound en remplaçant la fonction **BB-VALEUR** par**FCV-VALEUR** (figure C.6). Il nécessite l'utilisation de

Nogood-recording(p)

renvoie **NR-VARIABLES**($\emptyset$, **VARIABLES**(p))

NR-VARIABLES($A, \mathcal{V}$)

: A est l'affectation partielle d'un ensemble de variables

: $\mathcal{V}$ est l'ensemble des variables non affectées

si $\mathcal{V} = \emptyset$
alors renvoie $\emptyset$
sinon, soit $x =$ **CHOIX-VARIABLES**($\mathcal{V}$)
soit $\mathcal{C} =$ **NR-VARIABLE**($A, \mathcal{V}, \emptyset, x,$ **DOMAINE**(x))
CREE-NOGOOD($A, \mathcal{C}$)
renvoie $\mathcal{C}$

NR-VARIABLE($A, \mathcal{V}, \mathcal{C}x, d$)

si $d = \emptyset$
alors renvoie $\mathcal{C} \cup$ **DOMAINE**(x)
sinon, soit $i =$ **CHOIX-VALEUR**(d)
soit $\mathcal{C}' =$ **NR-VALEUR**($A, \mathcal{V}, x, i$)
si $\mathcal{C}' = \emptyset$
alors renvoie $\emptyset$
sinon
si **ABSENT**(x, **VARIABLES**($\mathcal{C}'$))
alors renvoie $\mathcal{C}'$
sinon renvoie **NR-VARIABLE**($A, \mathcal{V}, \mathcal{C} \cup \mathcal{C}', x, d - \{i\}$)

NR-VALEUR($A, \mathcal{V}, x, i$)

soit $\mathcal{C} =$ **ANALYSE-COHERENCE**(**CONTRAINTES**(x), $A \cup \{x = i\}$)
si $\mathcal{C} = \emptyset$
alors renvoie **NR-VARIABLES**($A \cup \{x = i\}, \mathcal{V} - \{x\}$)
sinon renvoie $\emptyset$

FIG. C.4 – *Nogood Recording*

BB(p, α)

: on cherche une affectation de valuation minimale

: strictement inférieure à α

soit $\beta =$ **BB-VARIABLES**$(\emptyset,$ **VARIABLES**$(p), \alpha, \bot)$
si $\beta \succeq \alpha$
alors renvoie *échec*
sinon renvoie β

BB-VARIABLES$(A, \mathcal{V}, \alpha, \beta)$

: A est l'affectation partielle d'un ensemble de variables

: $\mathcal{V}$ est l'ensemble des variables non affectées

: α est la borne max

: β est la borne min correspondant au nœud courant

si $\mathcal{V} = \emptyset$
alors renvoie β
sinon, soit $x =$ **CHOIX-VARIABLES**$(\mathcal{V})$
renvoie **BB-VARIABLE**$(A, \mathcal{V}, x,$ **DOMAINE**$(x), \alpha, \beta)$

BB-VARIABLE$(A, \mathcal{V}, x, d, \alpha, \beta)$

si $d = \emptyset$
alors renvoie α
sinon, soit $i =$ **CHOIX-VALEUR**(d)
soit $\alpha' =$ **BB-VALEUR**$(A, \mathcal{V}, x, i, \alpha, \beta)$
renvoie **BB-VARIABLE**$(A, \mathcal{V}, x, d - \{i\}, \alpha', \beta)$

BB-VALEUR$(A, \mathcal{V}, x, i, \alpha, \beta)$

soit $\beta' = \beta \oplus$ **VALUATION**(**CONTRAINTES**$(x), A \cup \{x = i\})$
si $\beta' \succeq \alpha$
alors renvoie α
sinon renvoie **BB-VARIABLES**$(A \cup \{x = i\}, \mathcal{V} - \{x\}, \alpha, \beta')$

FIG. C.5 – *Branch and Bound en profondeur d'abord*

fonctionnalités de bas niveau permettant l'attribution d'une valuation prévue à chaque valeur des variables et la sauvegarde de ces données :

- **POSE-VAL**(i, x, v) associe à la valeur i de la variable x la valuation v ;
- **PREND-VAL**(i, x) renvoie la valuation associée à la valeur i de la variable x. Par défaut, cette valuation vaut $\bot$;
- **EMPILE**$(\mathcal{V})$ empile en mémoire les valuations courantes associées aux variables de l'ensemble de variables $\mathcal{V}$;
- **DEPILE**$(\mathcal{V})$ rétablit les valuations des valeurs des variables de $\mathcal{V}$ en les dépilant de la mémoire.

FCV-VALEUR$(A, \mathcal{V}, x, i, \alpha, \beta)$

EMPILE$(\mathcal{V})$
soit $\beta' = \beta \oplus$ **VALUATION**(**CONTRAINTES**$(x), A \cup \{x = i\}, \alpha, \beta)$
soit $\beta'' = \beta' \oplus$ **FCV-VARIABLES**$(A \cup \{x = i\}, x,$ **VOISINES**$(x, \mathcal{V}), \alpha, \beta', \bot)$
si $\beta'' \succeq \alpha$
alors $\gamma \leftarrow \alpha$
sinon $\gamma \leftarrow$ **BB-VARIABLES**$(A \cup \{x = i\}, \mathcal{V} - \{x\}, \alpha, \beta'')$
DEPILE$(\mathcal{V})$
renvoie γ

FCV-VARIABLES$(A, x, \mathcal{V}, \alpha, \beta, \delta)$

: A est l'affectation partielle d'un ensemble de variables

: $\mathcal{V}$ est l'ensemble des variables non affectées

: δ est le surcoût minimum prévu

si $\mathcal{V} = \emptyset$
alors renvoie δ
sinon, soit $y =$ **CHOIX-VARIABLES**$(\mathcal{V})$
soit $\delta' = \delta \oplus$ **FCV-VARIABLE**$(A, x, y,$ **DOMAINE**$(y))$
si $\beta \oplus \delta' \succeq \alpha$
alors renvoie δ'
sinon renvoie**FCV-VARIABLES**$(A, x, \mathcal{V} - \{y\}, \alpha, \beta, \delta')$

FCV-VARIABLE(A, x, y, d)

si $d = \emptyset$
alors renvoie $\bot$
sinon, soit $j =$ **CHOIX-VALEUR**(d)
soit $\delta_{y^j} =$ **PREND-VAL**$(j, y) \oplus$
VALUATION(**CONTRAINTES**$(x, y), A \cup \{y = j\})$
POSE-VAL(j, y, δ_{y^j})
renvoie $\min(\delta_{y^j},$ **FCV-VARIABLE**$(A, x, y, d - \{j\})$

FIG. C.6 – *Forward-checking valué*

Bibliographie

[ABB+95] Agnèse (J.C.), Bataille (N.), Bensana (E.), Blumstein (D.) et Verfaillie (G.). – Exact and Approximate Methods for the Daily Management of an Earth Observation Satellite. *In : Proc. of the 5th ESA Workshop on Artificial Intelligence and Knowledge Based Systems for Space.* – Noordwijk, The Netherlands, 1995.

[AS94] Alliot (Jean-Marc) et Schiex (Thomas). – *Intelligence Artificielle et Informatique Théorique.* – Cépaduès-Editions, 1994, 2nd édition. ISBN : 2-85428-324-4.

[Bad88] Badie (C.). – *Contribution à l'étude des problèmes d'affectation de ressources et d'ordonnancement : application au domaine spatial.* – Toulouse, France, Thèse de doctorat, ENSAE, 1988.

[BFM+96] Bistarelli (S.), Fargier (H.), Montanari (U.), Rossi (F.), Schiex (T.) et Verfaillie (G.). – Semiring-based CSPs and Valued CSPs: Basic Properties and Comparison. *In : Over-Constrained Systems (LNCS 1106, Selected papers from the Workshop on Over-Constrained Systems at CP'95, reprints and background papers)*, éd. par Jampel (M.), Freuder (E.) et Maher (M.), pp. 111–150. – Springer, 1996.

[BJ95] Bouquet (F.) et Jégou (P.). – Solving Over-constrained CSP Using Weighted OBDD. *In : Proc. of the CP95 Workshop on "Over-Constrained Systems".* – Cassis, France, 1995.

[BR96] Bessière (C.) et Régin (J.C.). – MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?). *In : Proc. of CP-96*, pp. 61–75. – Cambridge, MA, USA, 1996.

[Bry86] Bryant (R.). – Graph-based algorithms for boolean function manipulation. *IEEE Trans. on Computers*, vol. C-35, n° 8, 1986, pp. 677–691.

[McA93] McAllester (D.). – Partial order backtracking. – 1993. "www.ai.mit.edu/people/dam/dynamic.ps".

[MJPL92] Minton (S.), Johnston (M.), Philips (A.) et Laird (P.). – Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, vol. 58, 1992, pp. 160–205.

[Mor93] Morris (P.). – The breakout method for escaping from local minima. *In : Proc. of AAAI-93.*

[MV96] Martinez (D.) et Verfaillie (G.). – Echange de Nogoods pour la résolution coopérative de problèmes de satisfaction de contraintes. *In : Actes de la Seconde Conférence Nationale sur la Résolution pratique de Problèmes NP-complets (CNPC96)*, pp. 261–274. – Nancy, France, 1996.

[PCR96] Parrod (Y.), Corbière (D.) et Rebejac (C.). – TIMELINE, a generic tool for mission planning. *In : 4th International Symposium on Space Mission Operation and Ground Data Systems.*

[Phi96] Phillips (S.). – The capabilities of the graphical observation scheduling system as used by the ASTRO-2 spacelab mission. *In : 4th International Symposium on Space Mission Operation and Ground Data Systems.*

[Pro93] Prosser (P.). – Hybrid Algorithms for the Constraint Satisfaction Problems. *Computational Intelligence*, vol. 9, n° 3, 1993, pp. 268–299.

[SF94] Sabin (D.) et Freuder (E.). – Contradicting Conventional Wisdom in Constraint Satisfaction. *In : Proc. of ECAI-94*, pp. 125–129. – Amsterdam, The Netherlands, 1994.

[SFV95] Schiex (T.), Fargier (H.) et Verfaillie (G.). – Valued Constraint Satisfaction Problems: Hard and Easy Problems. *In : Proc. of IJCAI-95*, pp. 631–637. – Montréal, Canada, 1995.

[SK93] Selman (B.) et Kautz (H.). – Domain Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. *In : Proc. of IJCAI-93*, pp. 290–295. – Chambery, France, 1993.

[Smi94] Smith (B.). – Phase Transition and the Mushy Region in Constraint Satisfaction Problems. *In : Proc. of ECAI-94*, pp. 100–104. – Amsterdam, The Netherlands, 1994.

[SV93] Schiex (T.) et Verfaillie (G.). – Nogood Recording for Static and Dynamic CSP. *In : Proc. of the 5th IEEE International Conference on Tools with Artificial Intelligence (TAI93)*, pp. 48–55. – Boston, MA, 1993.

[SV94a] Schiex (T.) et Verfaillie (G.). – Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools*, vol. 3, n° 2, 1994, pp. 187–207.

[SV94b] Schiex (T.) et Verfaillie (G.). – Stubbornness: a Possible Enhancement for Backjumping and Nogood Recording. *In : Proc. of ECAI-94*, pp. 165–169. – Amsterdam, The Netherlands, 1994.

[VLS96] Verfaillie (G.), Lemaître (M.) et Schiex (T.). – Russian Doll Search for Solving Constraint Optimization Problems. *In : Proc. of AAAI-96*, pp. 181–187. – Portland, OR, 1996.

[VS94] Verfaillie (G.) et Schiex (T.). – Solution Reuse in Dynamic Constraint Satisfaction Problems. *In : Proc. of AAAI-94*, pp. 307–312. – Seattle, WA, 1994.

[VS95] Verfaillie (G.) et Schiex (T.). – Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes: bilan de quelques approches. *Revue d'Intelligence Artificielle*, vol. 9, n° 3, 1995, pp. 269–309.

[Wal96] Wallace (R.). – Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problem. *In : Proc. of AAAI-96*, pp. 188–195. – Portland, OR, 1996.

[WF95] Wallace (R.) et Freuder (E.). – Heuristic methods for over-constrained constraint satifaction problems. *In : Proc. of the CP'95 Workshop on Over-Constrained Systems.* – Cassis, France, 1995.

Liste des figures

2.1 Exemple. 7

2.2 Recherche arborescente du Backtrack. 9

2.3 Forward-checking. 11

2.4 Conflict-directed backjumping. 16

2.5 Nogood-recording. 21

2.6 Le Branch and Bound 33

2.7 Le Forward-checking valué. 36

2.8 Contraintes testées par les algorithmes de prévision. 38

2.9 DAC. 40

2.10 DAC + Forward-checking. 41

2.11 Evaluation des valuations globales par les Poupées Russes 41

2.12 Poupées Russes. 43

3.1 Equivalence $\Uparrow^{A'}(\varsigma)$, $\Uparrow^{A'}(\Downarrow(\varsigma))$ 53

3.2 Arc-cohérence valuée. 72

3.3 Espérance du nombre de solutions 74

3.4 Arc-cohérence valuée : résultats 76

3.5 Arc-cohérence valuée : résultats 77

3.6 Nogood-recording valué sans réduction. 82

3.7 Nogood-recording valué. 82

3.8 Nogood Recording valué : résultats 84

3.9 Nogood Recording valué : résultats 85

3.10 Réduction partielle $n' = \Downarrow_{A,n} (n_2)$ 92

3.11 Résultats généraux 101

3.12 Résultats généraux 102

4.1 Satellites défilants. 105

5.1 Plus long chemin 118

5.2 Plus long chemin double 121

5.3 Valuations 130

5.4 Ordre partiel 131

5.5 Agrégation de deux valuations 131

5.6 Minimum 133

A.1 Exemple. iii

C.1 Backtrack ix

C.2 Forward-checking x

C.3 Conflict Directed Backjumping xi

C.4 Nogood Recording xiii

C.5 Branch and Bound en profondeur d'abord xiv

C.6 Forward-checking valué xvi

Liste des tableaux

2.1 Backtrack dynamique 26

3.1 Arc-cohérence valuée 70

3.2 Arc-cohérence valuée, ordre inverse 71

3.3 Backtrack Dynamique valué : exemple 97

3.4 Backtrack Dynamique valué : résultats 98

5.1 Résultats sans contrainte mémoire 125

5.2 Résultats comparatifs sans contrainte mémoire 127

5.3 Résultats statistiques sans contrainte mémoire 127

5.4 Résultats sur une orbite 137

5.5 Résultats statistiques sur une orbite 137

5.6 Résultats sur plusieurs orbites 140

Dans cette étude, nous nous intéressons à la résolution des "**Problèmes de Satisfaction de Contraintes valués**" (CSP valués). Cette modélisation extrêmement simple, intuitive et riche s'adapte facilement à une grande variété de problèmes d'optimisation. De nombreux travaux ont par ailleurs fourni toute une bibliothèque d'algorithmes de résolution.

Dans le cadre classique de la satisfaction des CSP, l'efficacité de ces méthodes n'est plus à démontrer et leur domaine d'application est assez bien connu. L'introduction d'une valuation des contraintes dans le modèle CSP, qui donne naissance aux CSP valués, est cependant trop récente pour que les possibilités et les limites d'efficacité soient clairement identifiées. L'ordonnancement des satellites d'observation de la Terre fait partie des CSP valués à la frontière des compétences actuelles en matière d'optimisation. Cette application est pour nous un prétexte au développement d'outils de résolution nouveaux étendant le champ des CSP valués accessibles.

Notre travail est centré sur la notion de "contrainte induite", largement utilisée dans le cadre classique, et dont l'extension difficile aux CSP valués limite actuellement l'amélioration des algorithmes de recherche complets. Afin de dépasser ces difficultés, directement liées à la non idempotence de l'agrégation des valuations des contraintes, nous proposons un formalisme minimal pour les contraintes induites. Nous montrons ensuite comment il peut être mis en œuvre pour étendre l'essentiel des algorithmes développés dans le cadre classique. Enfin, l'utilité de ces nouveaux outils est illustré sur les problèmes d'ordonnancement du satellite SPOT 5.

Mots-clefs : CSP, VCSP, Optimisation, Ordonnancement, Satellites

---

In this study, we are interested in "**Valued Constraint Satisfaction Problems**" (Valued CSPs) solving. This model which is extremely genuine, intuitive and rich matches easily most of optimization problems. Many works have produced a full library of solving algorithms.

In the CSP framework, the efficiency of these methods is admitted and their limits well known. The adjunction of a valuation to the constraints in the CSP model that brought Valued CSPs is quite recent and the possibilities and the solving limits are not well identified. Earth observation satellite scheduling is a problem that is on the border of the optimization abilities at the present time. For us, this application is a pretense to extend the field of accessible Valued CSPs.

Our work focuses on the concept of "induced constraint" that is widely used in the classical CSP framework but is hard to extend to the Valued CSP framework. So that the improvement of the complete solving tools is limited for the present. In order to overcome these difficulties, directly linked to the non idempotency of the aggregation of the constraints valuations, we propose a minimal formalism for induced constraints. Then, we show how to extend the main algorithms developped in the classical CSP framework. At last, the usefullness of these new tools is illustrated on the satellite SPOT 5 scheduling problems.

Keywords: CSP, VCSP, Optimization, Scheduling, Satellites

ÉCOLE NATIONALE SUPÉRIEURE DE L'AÉRONAUTIQUE ET DE L'ESPACE

10, avenue Edouard-Belin - B.P. 4032 - 31055 TOULOUSE CEDEX

Tél. (33) 62 17 80 80 - Fax (33) 62 17 83 30