Application de la Programmation en Logique avec Contraintes au Problème d'Emploi du temps

Xavier Cousin

N° d'Ordre : 971

THESE

Présentée

DEVANT L'UNIVERSITE DE RENNES I

U.F.R. ..... I F S I C .....

pour obtenir

le grade de DOCTEUR de l'UNIVERSITE de RENNES 1
mention INFORMATIQUE

PAR

Xavier COUSIN

Sujet de la thèse

Application de Programmation en Logique avec Contraintes au Problème d'Emploi du Temps

Soutenue le 02 juin 1993 devant la commission d'Examen

MM.	BANATRE J.P.	Président
	GONDRAN M.	Rapporteurs
	DELAHAYE J.P.	
	SEGUIN J.	Examinateurs
	DINCBAS M.	
	HERMAN D.	
	LEPAPE J.P.	Invité

Sujet :

Application de la Programmation en logique avec Contraintes au Problème d'Emploi du Temps

par

Cousin Xavier

Thèse présentée par Xavier COUSIN

Sujet : Application de la Programmation en Logique avec Contraintes au Problème d'Emploi du Temps.

Le but de cette thèse est de proposer une aide à la modélisation et à la résolution des problèmes d'emploi du temps. Pour faire cette aide, nous utilisons la programmation en logique avec contraintes.

La première partie de cette thèse fait état des travaux rencontrés dans la littérature pour bien comprendre le lien à faire entre programmation en logique avec contraintes et problème d'emploi du temps. Elle présente d'une part la problématique du problème d'emploi du temps par les différentes résolutions rencontrées et d'autre part les différentes techniques de programmation en logique avec contraintes avec les divers langages existants.

La deuxième partie présente ma réflexion sur la manière de faire le lien entre programmation en logique avec contraintes et problème d'emploi du temps. Afin d'avoir une résolution plus générale, nous définissons un nouveau modèle de coloration : la coloration multi-dimensionnelle. Puis nous donnons une aide à la résolution du problème d'emploi du temps en proposant un modèle basé sur cette coloration. Celui-ci se traduit facilement en programmation en logique avec contraintes. Enfin, nous donnons une aide à la modélisation en détectant les contraintes à relâcher en cas de blocage. Ainsi nous pouvons faire un relâchement sensé permettant une interaction intelligente entre une résolution d'un problème et sa modélisation.

Thesis presented by Xavier COUSIN

Subject : Application of Constraint Logic Programming to Timetable Problem.

The goal of this thesis is to propose an aid to the modelisation and the resolution of timetable problems with the constraint logic programming.

The first part of this thesis presents the works encountered in literature to have a good understanding of the link between constraint logic programming and timetable problem. On the one part, we present the problematic of timetable problem by the different encountered resolutions, on the other part the different techniques of constraint logic programming with the varied existing languages.

The second part presents my thought on the manner to make the link between constraint logic programming and timetable problem. In order to have a most general resolution, we define a new coloration model : the multi-dimensional coloration. Next, we give an aid to the timetable problem resolution by suggesting a model based on this coloration. We translate it easily into constraint logic programming. Finally, we give an aid to the modelisation by detecting the constraints to relaxe in case of conflict. Then, we can make a sensible relaxation allowing an intelligent interaction between a resolution of one problem and its modelisation.

Remerciements

Je remercie le CNET et particulièrement les membres de l'équipe PLA pour m'avoir proposé ce sujet et pour avoir suivi mes travaux.

Je remercie mon directeur de thèse, Jean SEGUIN, pour m'avoir autorisé à soutenir ainsi que les membres éminents du jury pour y être présents.

Plus particulièrement, je remercie Mouloud Kharoune pour m'avoir encouragé et pour s'être impliqué intégralement dans ma thèse.

Enfin, je remercie ceux et celles qui, tout au long du déroulement de ma thèse, m'ont supporté et encouragé.

PLAN

Introduction générale

Cette thèse a été faite au sein des laboratoires du CNET à Lannion dans l'équipe PLA. Elle a été financée en partie par le CNET et en partie par le Conseil Régional de Bretagne.

Le but de cette thèse est de proposer une aide à la modélisation et à la résolution des problèmes d'emploi du temps.

Le problème d'emploi du temps met en jeu plusieurs entités : les objets à placer, les éléments d'affectations et les contraintes liées aux objets. La résolution d'un problème d'emploi du temps consiste à associer un élément d'affectation à chaque objet de telle manière que les contraintes en place soient toutes satisfaites.

La réalisation à la main d'un emploi du temps n'est pas facile. La combinaison de divers facteurs : gros volume de données, forte combinatoire, prise en compte de nombreuses et différentes contraintes font de la recherche d'une solution un pénible travail monopolisant un ou deux hommes pendant plusieurs jours. Chaque fois qu'une impossibilité apparaît, il faut tout remettre en question. La difficulté est de pouvoir prendre en compte cette masse d'informations sans rien oublier en cours d'élaboration. De plus, certains emplois du temps sont remis en cause régulièrement par suite d'événements intempestifs.

Ces difficultés ont induit l'idée d'informatiser la construction des emplois du temps. Cependant, les résolutions existantes se trouvent confronter à plusieurs problèmes.

Dans un premier cas, aucune solution n'est trouvée, donc n'existe. Il faut relâcher les contraintes les plus superflues dans l'espoir de dégager des solutions au nouveau problème.

Dans un deuxième cas, des solutions peuvent exister mais leur recherche d'une façon automatique demande trop de temps : on devra se limiter à un ensemble incomplet d'affectations.

Pour ces deux cas, le procédé consiste alors à partir d'un ensemble d'affectations (ne respectant pas toutes les contraintes en place) pour obtenir une solution satisfaisante la plus approchée de celle désirée.

Ces ensembles d'affectations sont alors solutions d'un problème proche de celui initial. Nous les nommerons solutions approchées.

Ainsi, la résolution du problème d'emploi du temps se trouve confronter à deux difficultés :

- chercher une modélisation d'un problème parmi plusieurs possibles. Cette modélisation doit convenir à un ensemble de personnes,

- rechercher une ou plusieurs solutions au problème ainsi modélisé.

Pour une bonne résolution, il est important que la recherche de solutions satisfaisantes interagisse sur la recherche d'une bonne modélisation du problème.

C'est pourquoi, dans cette thèse, mon apport tourne autour de trois points :

- nous proposons un nouveau modèle de coloration : la coloration multi-dimensionnelle afin d'avoir une résolution plus générale,

- nous donnons une aide à la résolution du problème d'emploi du temps en proposant un modèle de résolution qui se traduit facilement en programmation en logique avec contraintes ; ce modèle s'appuie sur la nouvelle coloration,

- puis nous donnons une aide pour mieux modéliser un problème d'emploi du temps en tenant compte de la résolution de modèles précédants. L'interaction entre résolution et modélisation doit passer par un relâchement de contraintes tel qu'il recentre la résolution sur un modèle plus satisfaisant. Les mécanismes de relâchement actuels ne convenant pas pour ces problèmes appliqués et pour ce type de langages, nous proposons un modèle de relâchement intelligent.

La première partie de cette thèse fait état des travaux rencontrés dans la littérature pour bien comprendre le lien entre programmation en logique avec contraintes et problème d'emploi du temps. Elle présente d'une part la problématique du problème d'emploi du temps par les différentes résolutions rencontrées et d'autre part les différentes techniques de programmation en logique avec contraintes avec les divers langages existants.

La deuxième partie présente ma réflexion sur la manière de faire ce lien entre programmation en logique avec contraintes et problème d'emploi du temps. En première étape, un modèle de résolution est introduit s'appuyant sur une nouvelle coloration que je définis. La deuxième étape présente ma réflexion sur le relâchement de contraintes permettant de faire une interaction intelligente entre une résolution d'un problème et sa modélisation.

Pour présenter les problèmes d'emploi du temps, nous avons choisi un aspect théorique, illustré sur un même exemple. Dans ce cadre théorique, nous introduisons des notions mathématiques de représentation et de construction d'un problème général d'emplois du temps. Nous présentons ensuite quelques algorithmes de résolution (donc de contruction d'un emploi du temps) et leurs applications.

Il faut noter que les travaux rencontrés n'ont pas d'interaction entre la recherche de solutions et la recherche d'une bonne modélisation.

De plus, nous nous sommes aperçu que les problèmes où il fallait faire différents types d'affectation à un même objet étaient mal traités. La difficulté provient du fait que

les affectations de divers types interagissent entre elles.

Le deuxième volet de l'état de l'art présente la programmation en logique avec contraintes. Nous évoluons de la notion de satisfaction de contraintes vers celle de programmation en logique avec contraintes. Nous présentons quelques techniques d'évaluation et de satisfaction de contraintes pour la programmation en logique avec contraintes puis des méthodes de "relaxation" de contraintes pour la recherche de solutions approchées avec cette programmation. Nous ne retenons pas ces méthodes, car elle recherche les solutions à tâtons donc d'une façon trop aveugle pour le problème d'emploi du temps.

Enfin, nous présentons un langage, CLEF V1, qui fait partie du monde de la programmation en logique avec contraintes et qui a été développé au CNET par l'équipe PLA. Ce langage m'a servi d'outil de test et de validation pour mes idées.

La deuxième partie de ce rapport concerne exclusivement ma réflexion sur l'application de la programmation en logique avec contraintes au problème d'emploi du temps.

Le modèle de résolution proposé s'appuie sur un nouveau modèle de coloration. L'intérêt de ce modèle de coloration est qu'il généralise les résolutions de problèmes d'emploi du temps et permet de prendre en compte différents types d'affectations et leurs interactions afin de détecter au plus tôt les mauvaises combinaisons d'affectations.

De plus, on prouve qu'un cas particulier de problème peut transformer sa résolution avec la coloration multi-dimensionnelle en une résolution avec une multi-coloration uni-dimensionnelle.

Un modèle de résolution est alors proposé faisant le lien entre problème de satisfaction de contraintes et problème d'emploi du temps. Un premier intérêt est qu'il s'appuie sur la coloration multi-dimensionnelle et qu'il se transpose facilement en programmation en logique avec contraintes. D'autre part, l'introduction de deux nouvelles contraintes permet de prendre en compte une grande variété de propriétés de problème d'emploi du temps couramment rencontrées. Par exemple, la propriété de compacité qui jusqu'à présent n'était pas entiérement exprimée l'est par ces contraintes.

Ainsi pour illustrer ces notions, nous présentons un exemple de construction d'un problème d'emploi du temps d'un lycée et sa résolution. Cet exemple a l'avantage de dégager des propriétés spécifiques à cette classe de problèmes.

Les résultats trouvés par les deux langages CLEF V1 et CHIP nous permettent de conclure favorablement.

Cependant, ce modèle de résolution n'intervient que si le problème est bien modélisé. Si le problème est "surcontraint", il n'y aura pas de solution. Il faudra alors relâcher des contraintes pour trouver une meilleure modélisation.

Dans le cas particulier du problème d'emploi du temps, nous connaissons à priori les contraintes à mettre en oeuvre, mais nous ne savons pas toujours quelles sont les contraintes à relâcher. Souvent le relâchement de contraintes ne suit pas une loi rigoureuse, il s'agit plus d'un relâchement satisfaisant un ensemble de personnes, donc suivant une loi plus subjective.

Malheureusement, lorsque ces problèmes sont appliqués par la programmation en logique avec contraintes et qu'il en découle un échec ou que la résolution a été brusquement interrompue, il n'existe aucun moyen pour détecter les contraintes en cause, pour les relâcher et évaluer la "qualité" des solutions ainsi obtenues. La "qualité" est ici utilisée dans le sens du degré de satisfaction voulue en fonction d'un critère objectif (lié à une fonction mathématique) ou subjectif (lié à un intervenant humain).

Rappelons qu'un cas d'échec signifie que toutes les possibilités ont été testées et qu'aucune d'entre elles ne valide l'ensemble des contraintes. Autrement exprimé, elles violent un certain sous-ensemble de contraintes (pas forcément les mêmes, pour chaque possibilité testée). Il faut détecter les sous-ensembles de contraintes pénalisantes, en choisir un qui correspond aux contraintes les moins intéressantes, puis les relâcher afin d'avoir une "meilleure" solution.

Le deuxième volet de ma proposition va dans ce sens et propose une aide pour détecter les contraintes à relâcher et donc une aide pour mieux modéliser un problème à partir de résolutions précédentes.

Le premier point de ce volet concerne un modèle d'hypergraphe de contraintes insatisfaites. Si nous ne considérons qu'un ensemble de possibilités et qu'un ensemble de contraintes, ce modèle permet de trouver un choix d'ensembles de contraintes à relâcher. Quelques théorèmes prouvent qu'il est nécessaire de relâcher un de ces choix et que le relâchement est suffisant pour que le nouveau problème sans ces contraintes ait une solution.

Le deuxième point décrit un modèle d'application de cet hypergraphe de contraintes insatisfaites dans des langages de programmation en logique avec contraintes conformes à certaines hypothèses : domaines discrets finis, hiérarchie de contraintes, ... Nous montrons comment détecter les contraintes pénalisantes et les relâcher dans des langages de tels types. Quelques recommendations sont données pour trouver des "meilleures" solutions. L'intérêt de ce modèle est que la partie ajoutée au langage est minime.

A partir de CLEF V1, le troisième point montre alors comment implanter ce modèle d'application et de recherche de "meilleures" solutions. Ceci permet de mettre en évidence les divers ajouts faits à tout langage de programmation en logique avec contraintes intégrant des domaines finis et discrets pour obtenir ces mécanismes.

A partir du langage CLEF V1 amélioré, un exemple illustratif permet de se rendre compte de l'utilisation de ces mécanismes.

Avec le modèle proposé de résolution basée sur la coloration multi-dimensionnelle et avec le modèle de relâchement intégrée au langage choisi de programmation en logique avec contraintes, nous pouvons conclure que les problèmes d'emploi du temps sont bien résolus.

Un concepteur d'emplois du temps peut bien modéliser son problème en introduisant des nuances dans l'importance de ses contraintes. La solution obtenue lui fournit un emploi du temps le satisfaisant mais ne respectant pas forcément toutes les contraintes en place.

Partie I

Partie I

Introduction

Nous voulons résoudre des problèmes d'emploi du temps avec la Programmation en Logique avec Contraintes (P.L.C.). Il faut pouvoir justifier l'intérêt de cette approche. Nous allons donc présenter les principaux travaux pour résoudre les problèmes généraux d'emploi du temps. Nous constaterons leurs avantages et leurs limitations. Ceci nous permettra de bien dégager la problèmatique de l'emploi du temps. Nous nous rendrons compte que la Programmation en Logique avec Contraintes répond aux besoins indispensables pour aider un humain à concevoir des emplois du temps.

Le premier chapitre introduit la définition de la recherche d'un emploi du temps. Nous présentons ensuite un petit exemple. Celui-ci montre différentes caractéristiques d'un problème d'emploi du temps.

Puis nous montrons les principales résolutions existantes à l'aide de cet exemple. Ceci permet de découvrir leurs apports et leurs manques. Cela nous aide à mettre en évidence les difficultés du problème d'emploi du temps.

Le deuxième chapitre introduit la Programmation en Logique avec Contraintes. Nous montrons qu'il s'agit d'une synthèse entre Programmation en Logique et satisfaction de contraintes. Ce chapitre présente la manière dont nous évoluons de la notion de satisfaction de contraintes vers la Programmation en Logique avec Contraintes.

Ainsi nous pourrons comprendre les langages de la P.L.C.. Nous découvrirons les atoûts qu'ont ces langages pour le problème d'emploi du temps.

CHAPITRE A

Survol des techniques de résolutions d'Emploi du Temps

A.1 Introduction

Dans ce chapitre, nous présentons différentes approches pour résoudre des problèmes d'emploi du temps.

Dans un premier temps, nous introduisons quelques représentations de l'emploi du temps dans la théorie des graphes.

Puis dans un deuxième temps, nous présentons des modèles et les méthodes associés. Ils se fondent sur une résolution exacte. Nous dégageons trois principales méthodes : la méthode de coloration, la méthode de flot et de couplage et la programmation par variables entières.

Dans un troisième temps, des techniques moins exactes sont abordées. La solution trouvée est souvent approchée et non optimale. Nous rencontrons des algorithmes approchés, ceux de recherche opérationnelle et des techniques d'intelligence artificielle.

A chaque fois, nous dégageons les limitations inhérentes aux résolutions présentées. Nous montrons alors la complexité du problème dans sa globalité. Nous nous rendons compte par cette intermédiaire de la problématique d'emploi du temps.

Par la suite, nous distinguerons plusieurs aspects dans la problématique d'emploi du temps. Le terme "représentation d'emploi du temps" indiquera l'ensemble des composants (salle, professeur, groupe d'élèves, ...) et les contraintes répertoriées par le concepteur du problème (par exemple, le professeur Durand fait cours à la classe Terminale A1).

Un même exemple de problème d'emploi du temps illustre ces résolutions tout au long de ce chapitre. Il prend en compte des classes, des professeurs, des salles dans certains cas et de simples contraintes. Les cours sont constitués par ces données.

A.2. Représentation de l'emploi du temps

Les représentations du problème d'emploi du temps les plus communément rencontrées se basent sur la théorie des graphes. Nous trouvons plusieurs modélisations : en graphe (quelconque ou réseau), en multigraphes ou en hypergraphes [BER 63, 73, 87], [GON 85].

Quelques notions et définitions sont introduites. Elles permettent de présenter par la suite, les modèles, méthodes et techniques de résolution du problème d'emploi du temps.

A.2.1 Quelques représentations

a) *Graphes* :

Nous supposons connu la notion de graphe. Ainsi nous donnons un exemple de représentation d'un problème d'emploi du temps à l'aide de graphes.

Exemple : Il s'agit de décrire une représentation d'un problème d'emploi du temps pour des cours. Soit un ensemble de professeurs {P1, P2, P3}, un ensemble de classes {C1, C2}. L'ensemble des cours à planifier est une famille de l'ensemble produit professeurs x classes. Nous avons les données : {(P1C1), (P1C2), (P2C1), (P3C1), (P3C2)}.

La représentation suivante attribue les cours (ou les points (PiCj)) aux sommets. Deux cours sont reliés par une arête s'ils présentent un professeur ou une classe en commun. L'arête exprime que ces deux cours ne peuvent se dérouler simultanément. Les encadrés représentent la raison du lien entre deux sommets.

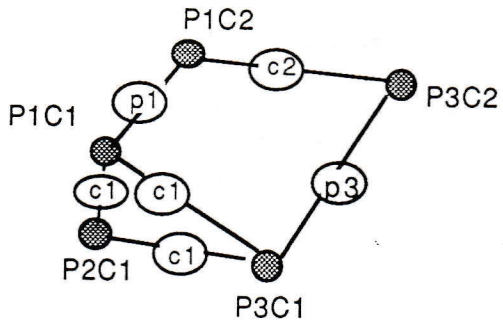


Figure 1. Représentation en graphe non orienté

b) *Multigraphes* :

Les **multigraphes** autorisent plusieurs arêtes entre deux sommets. Nous pouvons ainsi exprimer que deux cours identiques peuvent avoir lieu.

Exemple : l'exemple ci-dessus est repris. Cette fois le cours (P3C2) est représenté deux fois. Ces deux cours différents sont de même composition. Par exemple, si le professeur P3 est celui de physique-chimie, l'un des cours représente l'enseignement de physique, l'autre de chimie.

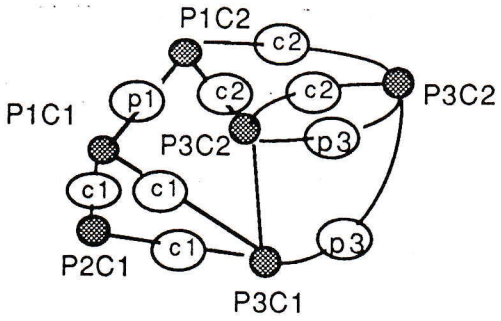


Figure 2. Représentation en multigraphe

c) *Hypergraphes* :

L'apport de cette représentation se trouve dans les relations inter-sommets. Ils ne sont plus binaires mais n-aires. Sa simplicité et sa puissance proviennent du mixage de la théorie des graphes et de la théorie des ensembles. Ces hypergraphes permettent de généraliser certains théorèmes.

Un **hypergraphe** est un couple [E, $\mathcal{F}$]. E est un ensemble non vide d'éléments ou de sommets. $\mathcal{F}$ est une famille des parties de E. Chaque élément de $\mathcal{F}$ constitue une **hyperarête**. Leur union reconstitue E.

Exemple : nous ajoutons à l'exemple précédent deux classes C1 et C2. Ils suivent le même cours avec le professeur P1. Les étiquettes représentent le lien entre chaque sommet.

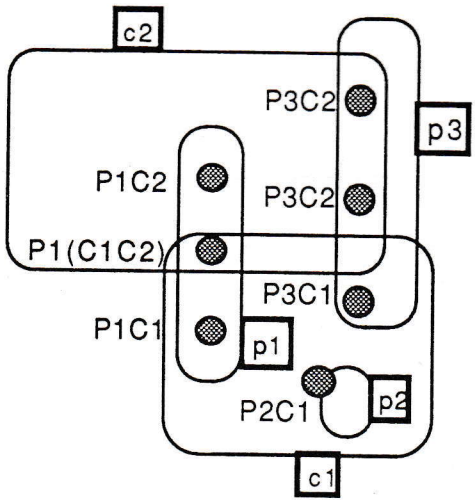


Figure 3. Représentation en hypergraphe

Dans la suite de ce document, nous avons opté pour ce type de représentation. Il sera indiqué si la représentation change.

d) Propriété de dualité

Précédemment les sommets représentent des cours. Les hyperarêtes modélisent les professeurs ou les classes. Ce choix est arbitraire. Dans certains cas, il est plus commode de représenter le problème dans des graphes duaux. La **dualité** d'un graphe s'exprime en inversant le rôle des sommets et celui des arêtes.

Le dual d'un hypergraphe $H = [E, \mathcal{F}]$ est noté $H^* = [\mathcal{F}, E]$. Il est constitué d'un ensemble de points f_i représentant les éléments F_i de $\mathcal{F}$. Ces points forment ses sommets. L'ensemble d'hyperarêtes est $\{E_j\}$ où $E_j = \{f_i / e_j \in F_i\}$. E_j n'est pas vide. L'union des E_j reconstitue $\mathcal{F}$. Le dual H^* est $[\mathcal{F}, \{E_1, ..., E_k\}]$. C'est aussi un hypergraphe.

Dans cette nouvelle représentation, un sommet indique un professeur ou une classe. Une arête relie des sommets pour former le cours correspondant.

Exemple : la figure suivante donne un hypergraphe dual de l'exemple présenté ci-dessus.

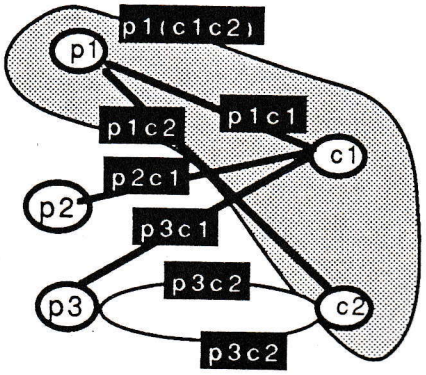


Figure 4. Hypergraphe dual

A.2.2 But du problème de l'emploi du temps

Les exemples précédents sont restreints à des problèmes simples d'emploi du temps scolaire. Nous généralisons la formulation du problème en définissant la notion de **composition**. Elle correspond à un cours, un examen, une visite médicale, ... Les **composants** sont divers éléments tels que des professeurs, des classes, des salles, des infirmières, des malades, ...

Nous décrivons plusieurs ensembles :

- E est un ensemble fini de composants,
- $\mathcal{F}$ est un ensemble fini de compositions,
- I est un ensemble fini de points,
- P est un ensemble de propriétés sur I et sur $\mathcal{F}$; celui-ci représente les contraintes liant les différents composants entre eux,

Ainsi soit un hypergraphe $H = [E, \mathcal{F}]$ et un ensemble fini I. Le but du problème d'emploi du temps [LAU 71, 74a, 74b] est la recherche d'une fonction g de $\mathcal{F}$ dans I vérifiant un ensemble de propriétés P.

En général, les compositions sont des cours et ont des durées connues. I peut être un ensemble de créneaux horaires. Le nombre de cours étant fini, le cardinal de I est au maximum égal à ce nombre. La recherche de la fonction g correspond à chercher alors une partition de $\mathcal{F}$ en respect de P par les points de I.

Quelques exemples simples

Exemple 1 :

Cet exemple traite le problème des examens écrits, schématisé par :

- $E = E_{élèves}$ ou l'ensemble des élèves,
- $\mathcal{F} = \{F_k$ = groupe d'élèves passant le même examen$\}$.
- I = ensemble de périodes unitaires.

La fonction g est celle qui minimise le nombre total de périodes. Une propriété indique que deux examens ayant des élèves en commun n'ont pas le même horaire :

$$\forall\ F_k \in \mathcal{F}, \forall\ F_{k'} \in \mathcal{F} \text{ tel que } k \neq k',\ F_k \cap F_{k'} \neq \emptyset \Rightarrow g(F_k) \neq g(F_{k'}).$$

Exemple 2 :

Cet exemple permet d'attribuer un planning de piqûres entre infirmières et malades. Ainsi les données sont :

- $E = E_{infirmières} \cup E_{malades}$,
- $\mathcal{F} = \{F_k = \{malade_i,\ infirmière_j\}\}$.
- I = ensemble d'heures.

La fonction g est celle qui minimise le nombre total d'heures. Une propriété stipule que deux piqûres avec une même infirmière ou un même malade se passent à des heures différentes :

$$\forall\ F_k \in \mathcal{F}, \forall\ F_{k'} \in \mathcal{F} \text{ tel que } k \neq k',\ F_k \cap F_{k'} \neq \emptyset \Rightarrow g(F_k) \neq g(F_{k'}).$$

Exemple 3 :

Cet exemple permet d'attribuer une flotte minimum d'avions à des vols. Un vol est défini par un intervalle de temps où l'avion est en l'air puis en maintenance :

- $E = E_{temps}$,
- $\mathcal{F} = \{F_k = \{$intervalle de temps pour $vol_k\}\}$.
- I = ensemble d'avions.

La fonction g est celle qui minimise le nombre total d'avions. Une propriété indique que deux vols dont leur intervalle de temps s'intersecte, n'ont pas le même avion :

$$\forall\ F_k \in \mathcal{F}, \forall\ F_{k'} \in \mathcal{F} \text{ tel que } k \neq k',\ F_k \cap F_{k'} \neq \emptyset \Rightarrow g(F_k) \neq g(F_{k'}).$$

Exemple 4 :

Cet exemple permet de minimiser le nombre de couches pour un circuit imprimé. Ce circuit imprimé est composé de points de connexion et de liaisons électriques. Ces dernières relient entre elles des connexions. Le circuit peut se composer de plusieurs niveaux de liaisons (ou couches). Deux liaisons électriques ne doivent pas s'intersecter sur la même couche :

- $E = E_{connexions}$,
- $\mathcal{F} = \{F_k = \{connexion_1, connexion_2\}$ ou liaison électrique$\}$.
- I = ensemble de couches.

La fonction g est celle qui minimise le nombre total de couches. Une propriété garantit que deux liaisons ne s'intersectent pas sur la même couche :

$\forall F_k \in \mathcal{F}, \forall F_{k'} \in \mathcal{F}$ tel que $k \neq k'$, $F_k \cap F_{k'} \neq \emptyset \Rightarrow g(F_k) \neq g(F_{k'})$.

Nous pouvons remarquer que cet exemple n'introduit pas la notion de temps. Il illuste le fait que la problématique d'emploi du temps représente une classe de problèmes assez large.

A.2.3 Un exemple illustratif simple

Un exemple simple est ici présenté. Il est décrit d'abord d'une manière informelle puis d'une façon plus formelle. Il illustre les modèles, méthodes et techniques de résolution de ce chapitre.

Les données de base de cet exemple sont constituées de professeurs, de classes et parfois de salles. Les cours sont des compositions de ces données.

Ainsi l'hypergraphe H $[E,\mathcal{F}]$ représente cet emploi du temps avec :

- $E = E_c \cup E_p$ (et parfois $\cup E_s$) ;
  E_c étant l'ensemble des classes,
  E_p l'ensemble des professeurs,
  E_s l'ensemble des salles.

- $\mathcal{F}$ est l'ensemble des cours F_k, $F_k = \{e_{cj} \in E_c, e_{pi} \in E_p, e_{su} \in E_s\}$. On suppose donc établi à priori les associations classe, professeur et dans certain cas, salle.

Le temps est modélisé par une trame temporelle I. Elle est découpée en intervalles. Nous devons trouver une solution dans cette trame I. Il n'y a aucune minimisation.

La trame temporelle I est représentée par un ensemble de ξ éléments. Chaque élément de cet ensemble correspond à une période de temps unitaire. Cet ensemble est partitionné en w sous-ensembles I_t.

Le but est de chercher dans un temps fini, une fonction g qui place tous les cours.

La fonction g est une fonction de $\mathcal{F}$ dans I sous certaines propriétés que nous allons expliciter.

La fonction g vérifie des propriétés. Elles correspondent à des astreintes de l'établissement sur des cours, des professeurs ou des classes.

Nous énonçons ici les principales propriétés rencontrées dans les établissements scolaires. Cette liste n'est pas exhaustive.

Nous prenons les notations : [x] pour la partie entière par défaut de x, [x]* pour la partie entière par excès et Card(Y) pour le Cardinal de l'ensemble Y.

a) Contraintes générales

Une contrainte générale est une propriété nommée (P_0) qui exprime l'impossibilité pour un composant (professeur, classe ou salle) d'avoir deux cours différents au même moment.

- (P_0) : $\forall F_k \in \mathcal{F}, \forall F_{k'} \in \mathcal{F}$ tel que $k \neq k'$, $F_k \cap F_{k'} \neq \emptyset \Rightarrow g(F_k) \neq g(F_{k'})$.

b) Contraintes de préaffectation

Les contraintes de préaffectation sont des propriétés (P_1) qui préaffectent des cours à des périodes choisies.

- (P_1) : Le cours F_k est affecté à la période de temps t0 est exprimé par :
  $t0 \in I$, $F_k \in \mathcal{F}$, $g(F_k) = t0$.

c) Contraintes d'indisponibilité

Les indisponibilités de professeurs à certaines périodes de temps sont représentées par des propriétés (P_2).

- (P_2) : Le professeur Pi est indisponible au temps t0 est exprimé par :
  $t0 \in I$, $\forall F_k \in \mathcal{F}$ tel que $Pi \in F_k \Rightarrow g(F_k) \neq t0$

d) Contraintes d'équilibrage

Rappelons que I est découpé en w I_t. Les contraintes d'équilibrage sont des propriétés (P_3) et (P_4).

Les propriétés (P_3) consistent à équilibrer les cours de certains composants sur chaque intervalle de temps I_t connu a priori.

Pour respecter un bon équilibrage, il faut que le nombre de cours relatif à chaque composant à chaque intervalle de temps soit le plus proche d'une moyenne. Pour obtenir cette moyenne, nous calculons le nombre de cours total de chaque composant (Cardinal de J_2) que nous divisons par le nombre d'intervalles (w).

- (P_3) : Equilibrage de cours pour les éléments e de E le désirant, sur chaque I_t :
  $\forall I_t \in I$, $e \in E$, soit J_1 et J_2 tel que :
  $J_1 = \{F_k \in \mathcal{F} / \ e \in F_k$, et $g(F_k) \in I_t\}$ et

$$J_2 = \{F_k \in \mathcal{F} / \ e \in F_k\}$$
$$\Rightarrow \quad [Card(J_2) \div w\ ] \leq \ Card(J_1) \leq [Card(J_2) \div w\ ]^*.$$

Les propriétés (P_4) permettent de répartir uniformément les cours de même composition.

• (P_4) : Equilibrage de cours entre les professeurs le désirant et les classes sur chaque I_t de I :

$\forall\ I_t \in I,\ e_1 \in E_c,\ e_2 \in E_p$, soit J_1 et J_2 tel que :
$J_1 = \{F_k \in \mathcal{F} / \ e_1 \in F_k, e_2 \in F_k$ et $g(F_k) \in I_t\}$ et
$J_2 = \{F_k \in \mathcal{F} / \ e_1 \in F_k, e_2 \in F_k\}$
$\Rightarrow \quad [Card(J_2) \div w\ ] \leq \ Card(J_1) \leq [Card(J_2) \div w\ ]^*.$

e) Contraintes de compacité

Certains composants réclament un emploi du temps "compact". Tous ses cours sont réunis dans un nombre de jours qui se suivent. Ces contraintes sont exprimées par des propriétés (P_5).

Nous sommes obligés d'introduire la notion du cours "Max". Il faut simplement noter que c'est le dernier cours du professeur (ou de la classe) e. Cette notion n'est pas nécessaire pour la compréhension de cette propriété.

• (P_5) : Emploi du temps "compact" pour les professeurs le désirant et pour certaines classes :

$e \in E_c \cup E_p$,
$Max = F_j$ tel que $g(F_j) =$ maximum $(\{g(F_k) / \ F_k \in \mathcal{F}, e \in F_k\})$,

$\Rightarrow \forall\ F_k \in \mathcal{F},\ e \in F_k,\ F_k \neq Max,$
$\exists\ F_{k'} \in \mathcal{F}, e \in F_{k'}, g(F_{k'}) - g(F_k) = 1.$

f) Contraintes de limitation

Les contraintes de limitation sont représentées par des propriétés (P_6). Chacune limite le nombre de cours utilisant une salle spécifique. Cette limite est due au nombre de telles salles disponibles.

• (P_6) : Limitation du nombre de salles spécialisées :
soit $Nb\_salles\_e_s$, le nombre de salles e_s pour une matière spécifique :
$\forall\ i \in I$, soit $J_1 = \{F_k \in \mathcal{F} / \ e_s \in F_k, g(F_k) = i\}$
$\Rightarrow Card(J_1) \leq Nb\_salles\_e_s$

N.B. : *L'exemple n'enlève rien à la généralité de ces contraintes. En effet, la contrainte de limitation peut servir à limiter le nombre d'interventions d'un professeur dans une semaine. Cette contrainte peut aussi limiter le nombre de vols d'un avion, etc.*

A.3. Calcul et Calculabilité

Des recherches ont été entreprises depuis les années 60 [HOL 64]. Cependant, le problème d'emploi du temps reste mal résolu dans sa totalité. Toutes les tentatives actuelles se basent sur des méthodes générales et exactes. Il a fallu introduire des limitations. La principale concerne le nombre de composants formant les cours. Une autre limite les types de contraintes manipulées.

Dans cette division A.3, nous présentons les capacités et les limites des principaux modèles et méthodes associées. Trois types de méthodes peuvent être dégagés. La première est la méthode de coloration dans des graphes, des multigraphes ou des hypergraphes. La seconde est la méthode de flot. Nous recherchons des chemins de valeur maximale dans des graphes particuliers. La troisième est la méthode de résolution de systèmes d'équations par la programmation linéaire entière.

Puis dans une dernière sous-division A.3.4, nous présentons diverses techniques. Elles ne cherchent pas une solution exacte. Mais elles essaient de trouver par diverses méthodes, une solution approchée donc non exacte.

A.3.1 Coloration

Dans cette partie, la coloration est abordée. Elle porte sur des graphes simples, des multigraphes ou des hypergraphes. En général, l'ensemble des couleurs est supposé ne pas être ordonné. Le séquencement entre les cours n'est pas pris en compte. Nous ignorons donc l'ordre entre les couleurs. Pour certains cas particulier, l'ordre établi est précisé.

Nous allons introduire des nombres remarquables : nombre de stabilité, nombre chromatique, indice chromatique. Ces recherches fournissent, en même temps, des familles d'ensembles.

Définissons un **ensemble stable** de sommets. Aucun élément d'un ensemble stable n'est adjacent à un autre élément du même ensemble. Chaque élément est un sommet. Par exemple, dans la figure 1, l'ensemble {(P1C1), (P3C2)} est stable.

Le **nombre de stabilité** d'un graphe est le cardinal du plus grand ensemble stable de sommets. Dans l'exemple de la figure 1, il existe ces ensembles stables : {(P3C1), (P1C2)}, {(P2C1), (P3C2)}, {(P1C1), (P3C2)}, {(P1C2), (P2C1)}, {(P3C1)}, {(P1C1)}, {(P2C1)}, {(P1C2)}, {(P3C2)}. Le nombre de stabilité est de deux.

Un autre nombre remarquable est le **nombre chromatique**. Colorons chaque sommet tel que deux sommets adjacents n'aient pas la même couleur. Le nombre chromatique est alors le nombre minimal de couleurs nécessaires. Dans l'exemple de la figure 1, le nombre chromatique est de trois.

Nous remarquons que tous les éléments d'une même couleur ne sont pas adjacents. Ils constituent un ensemble stable. La coloration détermine des ensembles stables indépendants. C'est-à-dire que leur intersection deux à deux est vide. Dans l'exemple de la figure 1, {(P1C1), (P3C2)} peut avoir la couleur bleu, {(P1C2), (P2C1)} la couleur verte, {(P3C1)} la couleur rouge.

D'autres nombres sont présentés au fur et à mesure de la présentation.

Dans notre résolution de problèmes d'emploi du temps, chaque période est

représenté par une couleur. Un ensemble stable est un ensemble de cours ayant lieu à la même période. Le nombre de stabilité est le nombre maximum de cours pouvant se dérouler simultanément. Le nombre chromatique est le nombre minimum de périodes nécessaires pour obtenir un emploi du temps.

N.B. : *Pour plus de compréhension, le lecteur se reportera aux ouvrages : [BER 63, 73, 87], [GON 85] pour les théorèmes et notions non explicités.*

a) Les méthodes et les modèles

Une première méthode consiste à colorer les sommets d'un graphe. Deux sommets adjacents ne doivent pas avoir la même couleur. Nous cherchons alors le nombre chromatique.

Beaucoup d'applications concrètes se basent sur différentes techniques de coloration de sommets [CAR 86][SCH 80]. Elles n'arrivent pas à résoudre d'une façon efficace la coloration d'un grand nombre de sommets (plusieurs centaines).

Une deuxième méthode de coloration porte sur les arêtes. Deux arêtes adjacentes par un sommet ne doivent pas avoir la même couleur.

Chaque couleur représente une heure. Le nombre chromatique donne le minimum d'heures nécessaires pour planifier l'ensemble des compositions.

Les travaux les plus marquants à ce sujet émanent de D. de Werra [DEW 75, 85a, 85b] et de Laurière [LAU 71, 74a, 74b]. Les diverses propriétés portent sur des multigraphes (cf [COU 89]).

b) L'application aux contraintes d'emploi du temps

i) Contraintes générales

Elles expriment que si deux cours ont un même composant alors ils n'ont pas lieu au même moment. Elles s'associent à la propriété (P_0) de notre exemple :

- un professeur ne peut enseigner qu'à une classe à la fois,

- une classe ne peut avoir cours qu'avec un professeur unique à un temps donné.

Première représentation

Supposons que la répartition des cours est connue (Professeur, Classe) par le tableau ci-dessous. Une première représentation se fait par les graphes simples. Nous colorons les sommets. Les arêtes du graphe traduisent les contraintes.

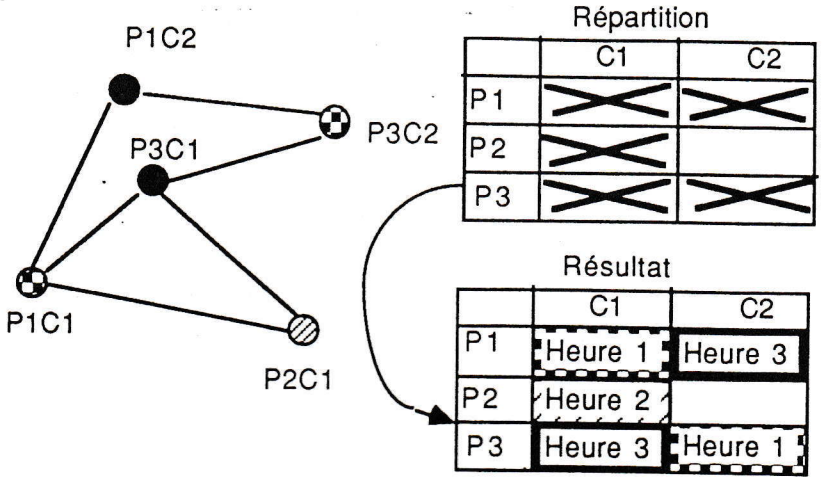


Figure 5. Coloration de sommets

Le nombre d'heures est le nombre chromatique. Il est alors de trois. Le nombre maximum de cours dans une heure est de deux. Dans notre cas, c'est le nombre de stabilité.

Deuxième représentation

Une autre représentation est faite par graphes bipartis. Un graphe biparti a son ensemble de sommets partitionné en deux sous-ensembles distincts. L'un des deux ensembles est l'image de l'autre. Une coloration des arêtes définit une partition de l'ensemble total de ces arêtes en classes d'équivalence.

Par exemple, un premier ensemble est celui des professeurs, un second, celui des classes. Ainsi l'exemple devient :

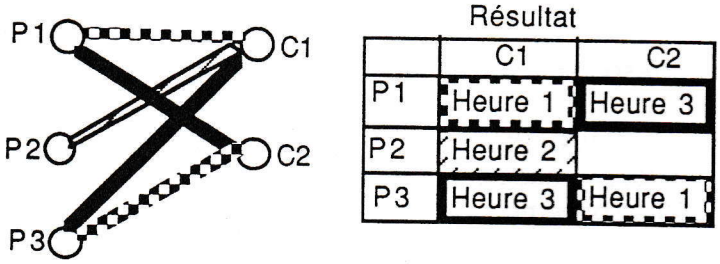


Figure 6. Coloration d'arêtes

Les classes d'équivalence sont des ensembles de cours pouvant se dérouler à la même heure. Sur l'exemple, trois classes d'équivalence sont trouvées : {(P1C2), (P3C1)}, {(P1C1), (P3C2)}, {(P2C1)}.

Troisième représentation

Certains cours sont dits spéciaux. Leur composition n'est pas un élément du produit cartésien des ensembles de composants (professeurs x classes). Souvent ce sont des cours regroupant plusieurs classes en même temps.

Pour représenter de tels cours, la représentation par graphe biparti n'est plus possible. La méthode est alors la coloration de sommets dans un hypergraphe.

La recherche d'une partition de l'ensemble des participants revient à colorer fortement les sommets de l'hypergraphe [LAU 71]. Une **coloration forte** est une coloration qui vérifie que tous les sommets d'une même arête sont tous, deux à deux, de couleurs différentes.

La figure 7 présente l'exemple ci-avant. Nous lui avons ajouté le cours P1(C1C2).

Nous pouvons aussi tenir compte de salles qui font parti a priori de la composition de certains cours.

La modélisation fait appel à un hypergraphe. L'étiquette attachée à chaque hyperarête rappelle la raison du lien correspondant.

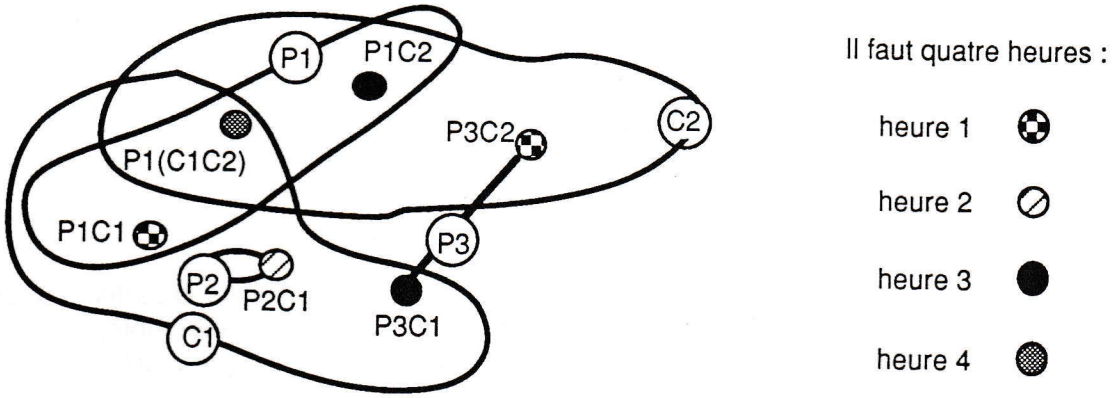


Figure 7. Coloration des sommets d'un hypergraphe

ii) Contraintes de préaffectation

Certains cours doivent être placés à certaines heures. La propriété (P_1) correspond à ces contraintes.

Nous supposons qu'une heure particulière correspond à chaque couleur. Prenons la couleur associée à l'horaire voulu de préaffectation. Nous attribuons cette couleur au cours concerné.

iii) Contraintes d'indisponibilité

Certains professeurs peuvent être indisponibles à certaines heures. Il faut alors que tous les cours les concernant ne prennent pas place à ces heures. Ces contraintes correspondent à la propriété (P_2) de notre exemple.

Une solution est d'ajouter des cours fictifs. Ces derniers sont préassignés aux heures d'indisponibilité. Nous créons une contrainte générale entre chaque cours fictif préassigné et les cours indisponibles.

iv) Contraintes d'équilibrage

Une autre notion de coloration existe : la coloration **équitable**. Prenons comme exemple la coloration équitable d'arêtes. Prenons les ensembles d'arêtes issus de chaque sommet. Retenons de ces derniers les sous-ensembles de même couleur. Tous les

- sous-ensembles possibles ont tous le même cardinal à une unité près. Il en est de même pour la coloration de sommets.

Soit un hypergraphe $H^* = [\mathcal{F}, E]$ avec E un ensemble d'hyperarêtes $\{E_i\}$. Une w-coloration équitable est une partition $(S_1, ..., S_w)$ de $\mathcal{F}$ en w classes d'équivalence. A chaque hyperarête E_i, nous avons :

$$[Card(E_i) \div w] \leq Card(E_i \cap S_j) \leq [Card(E_i) \div w]^* \ (\forall j \ ; \ j=1, ..., w).$$

Revenons à notre exemple de la partie A.2.3. Les couleurs ou (I_t) ne sont plus des heures. Il s'agit de journées (ou de semaines). Soit l'hypergraphe $H^* = [\mathcal{F}, E]$ dual de celui de notre exemple. $H^* = [\mathcal{F}, E]$ est l'hypergraphe schématisé par la figure 7. Les $F_u \in \mathcal{F}$ sont les cours et les $e_i \in E$ sont les composants. L'inéquation ci-dessus exprime donc la bonne répartition des cours de même composant dans chaque journée. La propriété (P_3) est exprimée par cette inéquation.

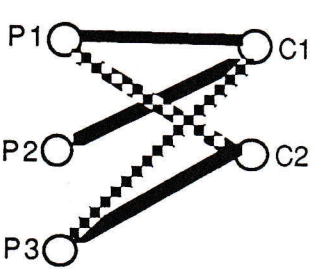
Pour exprimer la propriété (P_4), nous posons l'inéquation suivante. Il porte sur le même hypergraphe $H^* = [\mathcal{F}, E]$. Nous supposons que E_{pi} est l'ensemble des cours du professeur pi et E_{cj} celui de la classe cj :

$$\forall E_{pi}, \forall E_{cj}, \forall v \ ; v=1, ..., w$$
$$[Card(E_{pi} \cap E_{cj}) \div w] \leq Card((E_{pi} \cap E_{cj}) \cap S_v) \leq [Card(E_{pi} \cap E_{cj}) \div w]^*.$$

Ceci tient compte des différentes fréquences de cours de même composition. Si un cours doit se répéter plusieurs fois dans la semaine, il faut uniformément répartir ses différentes occurrences dans les w jours.

Exemple :

Prenons le graphe suivant. Cette fois, une semaine est associée à chaque couleur. Supposons que le nombre de couleurs (semaines) est limité à 2.



Résultat

	C1	C2
P1	Semaine1	Semaine2
P2	Semaine1	
P3	Semaine2	Semaine1

Figure 8. Coloration équilibrée

Le placement des cours se fait en deux étapes.

En première étape, nous utilisons une première palette de couleurs. Pour faire un équilibrage sur une semaine, nous utilisons les contraintes d'équilibrage.

La deuxième étape s'effectue sur les cours de toutes les journées de chaque semaine. Nous utilisons une deuxième palette de couleurs. À ce moment, la coloration est forte.

v) Contraintes de compacité

Certains professeurs veulent regrouper leurs cours dans une même partie de la semaine. Par exemple, ça peut être du Lundi au Mercredi. Leur emploi du temps est alors dit compact. La même contrainte existe aussi pour les classes.

Le désir exprimé par ces contraintes est de planifier les cours d'un même professeur et/ou d'une même classe à des périodes consécutives. C'est la propriété (P_5). Ceci implique un ordre dans les couleurs (ou intervalles de temps).

La résolution approchée de ce problème [DEW 85b] groupe les cours d'un professeur ou d'une classe par deux.

vi) Contraintes de limitation

Elles sont dues au nombre de salles spécialisées. Prenons un établissement avec un nombre restreint de salle de même type (laboratoire de physique, de chimie, ...). Ceci oblige à limiter, à chaque période de temps, le nombre de cours dans ces salles. Ces contraintes sont identiques à la propriété (P_6) de notre exemple.

Une solution [DEW 85a] est de prendre un modèle particulier d'hypergraphe $H^* = [\mathcal{F}, E]$ défini dans le paragraphe (iv). Une hyperarête E_{su} définit les cours à enseigner dans une salle *su* d'un type particulier. Soit σ salles de ce type dans l'établissement. L'hyperarête E_{su} est étiquetée d'un nombre $e_i = \sigma$. La contrainte est :

$$|E_{su} \cap S_j| \leq \sigma \ (\forall j \ ; \ j=1, \ ..., \ \xi).$$

c) Critiques

La méthode de coloration présente cependant certaines restrictions. Des hypothèses doivent être respectées pour résoudre efficacement ces problèmes d'emploi du temps.

Prenons la matrice A d'incidence du graphe représentant l'emploi du temps. Cette matrice est constituée d'éléments a_{ij} selon le principe suivant :

$a_{ij} = 1$ si le sommet x_i appartient à l'arête E_j
$= 0$ sinon.

La matrice ainsi constituée est dite **totalement unimodulaire** si tout déterminant extrait vaut +1, 0 ou -1. Le graphe, multigraphe ou hypergraphe issu de cette matrice est alors dit unimodulaire.

Pour obtenir une résolution en un temps polynomial, une des principales hypothèses est que tout graphe est unimodulaire. Des résolutions efficaces existent pour les graphes unimodulaires. Par exemple, un graphe biparti a sa matrice totalement unimodulaire. Les recherches actuelles sont faites autour de graphes bipartis [DEW 85b].

Il existe des résolutions efficaces pour d'autres cas. Even, Itai et Shamir° donnent un algorithme en $o(n^2)$ pour des problèmes d'emploi du temps particuliers. Pour ces problèmes, chaque professeur n'admet que deux possibilités maximum pour leurs horaires.

° tiré de [DEW 85a]

Cependant, les graphes construits sur des exemples réels et concrets n'ont pas ces propriétés. Laurière a fait une autre tentative de classification de ces matrices. Il a surtout étudié les hypergraphes s-parfaits [LAU 71].

De nombreuses études ont démontré que le problème de coloration est NP-complet [DEW 85a][GON 85]. La complexité de résolution devient exponentielle quand le nombre de composants est supérieur à deux, ou quand les contraintes font perdre l'unimodularité de la matrice d'incidence.

A.3.2 Flot et Couplage

Le couplage est une des premières méthodes à être développée. Gotlieb & Csima [HOL 64] ont utilisé cette méthode. Elle a suscité de nombreuses critiques par Lions [LIO 66] et des améliorations dues à Dempster [AUS 76].

Cette méthode va être décrite brièvement. La méthode de couplage est généralisée par la notion de recherche de flot à coût minimum dans un réseau de transport.

Les contraintes précédentes et les propriétés associées sont reprises ici.

a) Les méthodes et les modèles

Un couplage est un ensemble E_0 d'arêtes. Deux éléments de E_0 ne sont pas adjacents.

Dans le cas de l'emploi du temps, le but est de rechercher plusieurs couplages de cardinalité maximale.

La figure suivante illustre cette définition sur un graphe biparti :

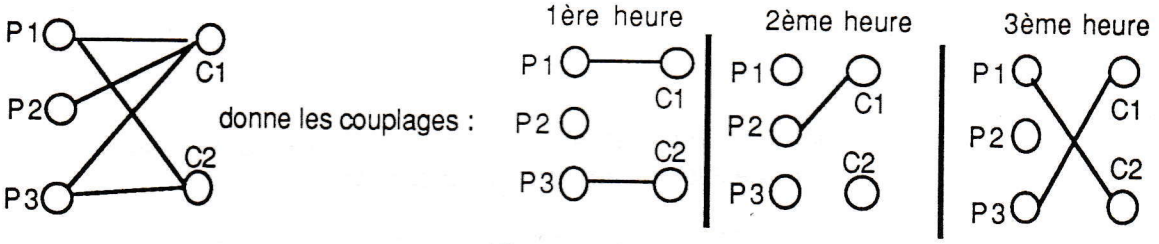


Figure 9. Couplages

La méthode de flot permet de généraliser la notion de couplage. Cette méthode s'applique à un type de graphe particulier. Il est appelé réseau de transport.

Un réseau de transport est un graphe fini orienté sans boucle R = (X, U). Chaque arc u a deux bornes positives : une maximale c(u) et une minimale b(u). De plus, deux sommets jouent un rôle particulier. Le premier est le sommet source (ou entrée). Aucun arc n'y entre. Le deuxième est le sommet puits (ou sortie). Aucun arc n'en sort.

Ce graphe sert alors à chercher une valeur de flot. Soit $U^+(x)$, l'ensemble des arcs incidents au sommet x et $U^-(x)$ l'ensemble des arcs sortants de x. Un flot φ est une fonction telle que :

- $\Sigma_{[u \in U^-(x)]} \varphi(u) - \Sigma_{[u \in U^+(x)]} \varphi(u) = 0$; ($x \neq$ source et $x \neq$ puits)
- $b(u) \leq \varphi(u) \leq c(u)$; ($u \in U$)

La valeur du flot est $\Sigma_{[u \in U^-(source)]} \varphi(u) = \Sigma_{[u \in U^+(puits)]} \varphi(u)$.

Nous prenons ici les flots à valeur entière. Le but de la recherche est de trouver une valeur maximale du flot. Le problème de l'emploi du temps peut aussi introduire des coûts $\gamma(u)$ de passage d'une unité de flot sur chaque arc u. Le but revient alors à trouver un ensemble de flots (multi-flot) à coût minimum. A chaque flot, nous cherchons à minimiser la fonction suivante :

$$\Sigma\ \varphi(u).\ \gamma(u)\ ;\ (u \in U)$$

Dans la suite, l'utilisation des flots est illustrée sur notre exemple de référence. Chaque type de contrainte est analysée. Puis nous discutons des limitations et des avantages de la méthode de flot.

b) L'application aux contraintes d'emploi du temps

i) Contraintes générales

En reprenant les mêmes données que pour la méthode de coloration, le réseau de transport se forme de la façon suivante :

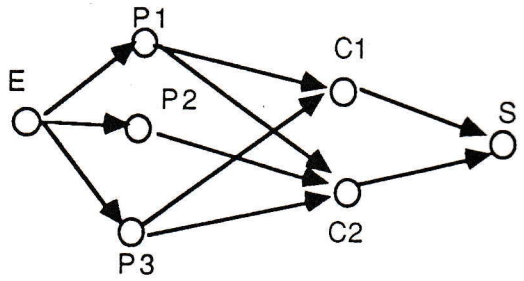


Figure 10. Flot avec deux composants

Pi joint Ca si le professeur i enseigne à la classe a. Les capacités et le coût de chaque arc sont alors définis suivant le tableau ci-dessous :

arcs(x,y)	capacité(x, y)	coût (x, y) au temps k
(E, Pi)	(1, 1)	1 si tous les cours ne sont pas encore ordonnés infini sinon
(Pi, Ca)	(1, 1)	1 si tous les cours ne sont pas encore ordonnés infini sinon
(Ca, S)	(1, 1)	1 si tous les cours ne sont pas encore ordonnés infini sinon

Figure 11. Tableau de coûts et de capacités

Chaque quadruplet (E, Pi, Ca, S) représente un cours. Il correspond à un chemin dans le réseau de transport. Le but est de former des ensembles de chemins allant de E à S. A chaque ensemble de cours correspond une heure.

Le procédé se fait itérativement. Nous allons calculer heure par heure, un flot de E à S de coût minimum. Nous attribuons une pénalité infinie aux cours déjà ordonnés. Ceci permet de ne plus les choisir aux heures ultérieures. Ça se traduit par le changement des

coûts des arcs choisis.

La propriété (P_0) correspond aux équations sur les arcs entrant et sortant de chaque sommet. Un seul arc, de valeur 1, entre à chaque sommet Pi. Ainsi un seul arc sortant est choisi.

Si certaines salles composent aussi ces cours, le réseau de transport se transforme en ajoutant un niveau. Le principe reste toujours le même. Il faut rechercher une famille de chemins de E à S.

Une autre méthode duplique les sommets des professeurs en Pi et P'i. Nous les relions par un arc de capacité de 1. Les cours sont des chemins (E, Ca, Pi, Pi', Salj, S). Les professeurs Pi et Pi' doivent être les mêmes.

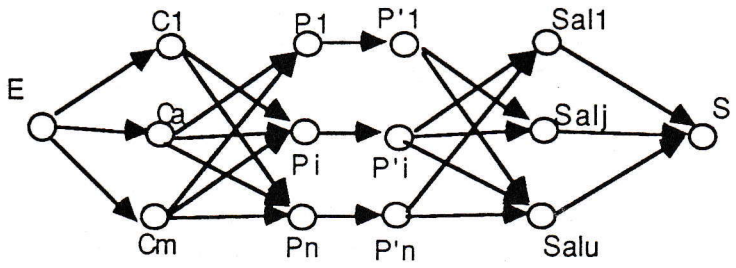


Figure 12. Flot avec trois composants

ii) Contraintes de préaffectation

Cette fois, le rôle de cette contrainte est de préaffecter les cours à des emplacements horaires. Prenons les arcs des cours à préassigner. Leur coût est à 0 pour le temps k et infini pour tous les autres temps.

iii) Contraintes d'indisponibilité

Un professeur (respectivement une classe) ne veut pas faire cours à certaines périodes k. Dès que la recherche de placement se fait pour les temps k, nous mettons un coût infini aux arcs de ce professeur (ou classe).

iv) Contraintes d'équilibrage

Ces contraintes garantissent une répartition uniforme des cours sur des intervalles de temps (par exemple, des jours). Le but de la solution tourne autour de deux principes :

1) Soit η le nombre de cours d'un professeur (respectivement d'une classe) dans un intervalle de temps I_t. Soit w le nombre de tels intervalles et soit N le nombre de tous les cours du professeur. η est encadré par la partie entière inférieure et supérieure du rapport $N \div w$.

2) Soit χ le nombre de cours entre un professeur et une classe dans un intervalle de temps I_t. Soit w le nombre de tels intervalles et soit M le nombre de tous les cours que partage ce professeur et cette classe. χ est compris entre les parties entières inférieure et supérieure de $M \div w$.

Ces contraintes sont similaires à celles de la coloration. Il faut noter que les

intervalles de temps ne sont pas des périodes simples. Il s'agit d'ensembles de périodes (par exemple, un intervalle correspond à une journée).

La première solution consiste à introduire deux capacités à chaque arc [DEW 85a].

Une deuxième solution pénalise les arcs suivant une fonction garantissant les deux principes ci-dessus.

Chahal & de Werra utilisent cette dernière solution dans un exemple précis [CHA 89]. En général, nous mettons à 0 le coût d'un arc entre le sommet d'un professeur Pi et celui d'une classe Ca. Cependant, le coût au temps k devient infini pour les arcs (Pi, Ca) si Pi a déjà été assigné à Ca aux temps k-2 et k-1 ou aux temps k+1 et k+2. Bien entendu, plusieurs arcs peuvent exister entre deux sommets.

v) Contraintes de compacité

Ces contraintes ont pour objectif de regrouper tous les cours d'un professeur (ou d'une classe).

Au temps k, le coût γ d'un arc (E, Pi) (resp. (Ca, S)) diminue d'une constante q si Pi (resp. Ca) a été impliqué au temps k-1 ou au temps k+1. Ceci permet de placer les cours par couple [CHA 89]. Nous prenons donc la façon approchée définie dans la coloration.

vi) Contraintes de limitation

Ces contraintes font intervenir des salles spécialisées. Soit un sommet Sal_S qui représente toutes ces salles. L'arc (Sal_S, S) reçoit une borne supérieure équivalente au nombre de telles salles spécialisées.

c) Critiques

L'avantage de cette méthode est qu'il existe des algorithmes efficaces pour la recherche d'un flot. Dans le cas de la construction d'un emploi du temps en ξ périodes, il s'agit de chercher ξ paquets de chemins indépendants donc autant de flots. Ce problème est alors à nouveau NP-complet [PLA 74].

Supposons que la matrice du système d'équations caractérisant le problème est totalement unimodulaire. Le problème est alors unimodulaire. Si le problème décrit est unimodulaire [TRI 84], il existe des algorithmes de complexité polynomiale.

C'est aussi le cas quand les sommets sont classés en deux parties (hormis l'entrée E et la sortie S) et où l'un des ensembles est image de l'autre. Cependant, cela devient NP-complet [DEW 85a] dès qu'il y a des contraintes de préaffectation et d'indisponibilité.

Par la suite, nous trouvons d'autres résolutions basées sur ce modèle de flot. Ces résolutions s'effectuent sur des problèmes plus généraux.

Là encore, il n'est pas facile d'intégrer tous les types de contraintes. Souvent celles prises en compte sont des cas particuliers liés aux établissements rencontrés.

A.3.3 Programmation par variables entières

Nous n'abordons dans cette partie que la programmation linéaire par variables entières. Les équations ou inéquations posées sur chaque variable sont linéaires. Les variables ne doivent prendre que des valeurs entières.

Il est à signaler qu'il existe un autre modèle basé sur les intersections de matroïdes. Il n'est pas détaillé ici (voir [COU 89] ou [DEF 78]). Ceci est dû au peu d'applications concrètes fondées sur ce modèle.

a) Les méthodes et les modèles

Le modèle tient compte d'une fonction objectif à optimiser (maximiser ou minimiser). Cette fonction porte sur des variables à valeur entière et sur des coûts. Dans le cas de l'emploi du temps ces variables ne prennent que deux valeurs : 0 ou 1.

Ces variables sont issues d'une matrice entre composants. Cette matrice $R = (r_{ij})$ est celle de répartition de service. Elle fournit le nombre de cours que doit assurer le professeur i à la classe j. Par exemple, soit une variable x_{ijk}. Elle est positionnée à 1 si le professeur i enseigne à la classe j au temps k. Elle est à 0 dans le cas contraire.

Ces variables sont alors contraintes par des équations ou inéquations. La programmation est linéaire ou non suivant le type de ces inéquations.

Il existe des algorithmes de résolutions des systèmes linéaires (Simplex). Donc le problème se formule souvent en termes d'expressions linéaires (par exemple [FER 85], [DEW 85a], [AKK 73]).

b) L'application aux contraintes d'emploi du temps

i) Contraintes générales

Nous allons étudier ces contraintes avec une matrice de répartition de service $R = (r_{ij})$ et des variables bivalentes x_{ijk}. Le nombre de professeurs est n. Le nombre de classes est m et le nombre de périodes ξ. Un coût c_{ijk} est déterminé à chaque affectation de cours à une période.

Les contraintes générales ont l'expression suivante :

$$\sum_{i=1}^{n} x_{ijk} \leq 1 \quad (\forall j\ j = 1, \ldots, m\ ;\ \forall k\ k = 1, \ldots, \xi),$$

$$\sum_{j=1}^{m} x_{ijk} \leq 1 \quad (\forall i\ i = 1, \ldots, n\ ;\ \forall k\ k = 1, \ldots, \xi).$$

La première inéquation exprime qu'au plus un professeur i enseigne à un temps k donné et à une classe j précise. La seconde inéquation signifie qu'une seule classe j, à une période k donnée reçoit un enseignement d'un professeur particulier i.

Sur cette base d'inéquations, le système à résoudre devient le suivant :

$$\text{Min.} \sum_i \sum_j \sum_k c_{ijk}\, x_{ijk},$$

$$\sum_{k=1}^{\xi} x_{ijk} = r_{ij} \quad (\forall j\ j = 1, \ldots, m\ ;\ \forall i\ i = 1, \ldots, n),$$

$$\sum_{i=1}^{n} x_{ijk} \leq 1 \quad (\forall j\ j = 1, \ldots, m\ ;\ \forall k\ k = 1, \ldots, \xi),$$

$$\sum_{j=1}^{m} x_{ijk} \leq 1 \quad (\forall i\ i = 1, \ldots, n\ ;\ \forall k\ k = 1, \ldots, \xi),$$

$x_{ijk} = 0$ ou 1 $(\forall i\ i = 1, \ldots, n\ ;\ \forall k\ k = 1, \ldots, \xi\ ;\ \forall j\ j = 1, \ldots, m)$.

La notion de salle est introduite par un indice supplémentaire s (variable x_{ijsk}).

ii) Contraintes de préaffectation

Le cours du professeur i et de la classe j est ordonné au temps k. Ceci se fait en attribuant à 1 la variable $x_{ijk} = 1$.

iii) Contraintes d'indisponibilité

Nous forçons à zéro les variables du professeur i (resp. de la classe j) impliquées au temps k. Ainsi la somme de ces variables doit valoir 0. L'inéquation se pose sous la forme suivante :

$$\sum_{j=1}^{m} x_{ijk} = 0$$ au temps k, pour le professeur i.

Une autre approche s'appuie sur des cours fictifs. Ceux-ci sont préassignés au temps k. Ils sont en disjonction avec ceux du professeur i.

Nous pouvons donner une autre formulation du problème. Nous intégrons préaffectations et indisponibilités dans la modélisation des données. Cette formulation permet de limiter le nombre de contraintes. De nouvelles variables sont alors définies :

$y_{ijk} = 1$ si le professeur i rencontre la classe j au temps k, 0 sinon ;
$p_{ijk} = 1$ si le professeur i et la classe j sont préassignés au temps k, 0 sinon ;
$b_{jk} = 1$ si la classe j est disponible et non préassigné au temps k, 0 sinon ;
$c_{ik} = 1$ si le professeur i est disponible et non préassigné au temps k, 0 sinon.

A partir de là, le nouveau système devient :

$$\sum_{k=1}^{\xi} y_{ijk} = r'_{ij} \quad (\forall j\ j = 1, \ldots, m\ ;\ \forall i\ i = 1, \ldots, n),$$

$$\sum_{i=1}^{n} y_{ijk} \leq b_{jk} \quad (\forall j\ j = 1, \ldots, m\ ;\ \forall k\ k = 1, \ldots, \xi),$$

$$\sum_{j=1}^{m} y_{ijk} \leq c_{ik} \ (\forall i \ i = 1, ..., n \ ; \forall k \ k = 1, ..., \xi),$$

$$y_{ijk} = 0 \text{ ou } 1 \ (\forall i \ i = 1, ..., n \ ; \forall k \ k = 1, ..., \xi \ ; \forall j \ j = 1, ..., m).$$

$$\text{où } r'_{ij} = r_{ij} - \sum_{k=1}^{\xi} p_{ijk}$$

iv) Contraintes d'équilibrage

Prenons un entier positif a_j (resp. b_i) pour chaque classe j (resp. professeur i). Il représente le nombre maximum de cours dans laquelle j est impliquée (resp. i) durant chacun des w jours.

La résolution se fait en deux étapes : affecter un jour à chaque cours puis affecter une période à chaque cours. La période d'un cours doit appartenir au jour de ce cours.

Plaçons nous dans le problème d'affectation des jours. Ainsi x_{ijk} est un nombre entier. C'est le nombre de cours impliquant la classe j et le professeur i au jour k.

Le problème devient :

$$\sum_{k=1}^{w} x_{ijk} = r_{ij} \ (\forall j \ j = 1, ..., m \ ; \forall i \ i = 1, ..., n),$$

$$\sum_{i=1}^{n} x_{ijk} \leq a_j \ (\forall j \ j = 1, ..., m \ ; \forall k \ k = 1, ..., w),$$

$$\sum_{j=1}^{m} x_{ijk} \leq b_i \ (\forall i \ i = 1, ..., n \ ; \forall k \ k = 1, ..., w),$$

$$x_{ijk} \geq 0 \ (\forall i \ i = 1, ..., n \ ; \forall k \ k = 1, ..., w \ ; \forall j \ j = 1, ..., m).$$

Dans le cas du problème ci-dessus, nous répartissons les cours dans chaque journée avec la contrainte d'équilibrage.

Nous ajoutons au problème, les contraintes suivantes :

$$[\sum_{i=1}^{n} r_{ij} \div w] \leq \sum_{i=1}^{n} x_{ijk} \leq [\sum_{i=1}^{n} r_{ij} \div w]^* \ (\forall j \ j = 1, ..., m \ ; \forall k \ k = 1, ..., w),$$

$$[\sum_{j=1}^{m} r_{ij} \div w] \leq \sum_{j=1}^{m} x_{ijk} \leq [\sum_{j=1}^{m} r_{ij} \div w]^* \ (\forall i \ i = 1, ..., n \ ; \forall k \ k = 1, ..., w),$$

$$[r_{ij} \div w] \leq x_{ijk} \leq [r_{ij} \div w]^* \ (\forall i \ i = 1, ..., n \ ; \forall j \ j = 1, ..., m \ ; \forall k \ k = 1, ..., w).$$

v) Contraintes de limitation

Soit une constante entière l_σ. Elle limite le nombre de salles d'une spécialité σ. A chaque tranche horaire, l_σ borne le nombre de cours utilisant ces salles. Soit S_{lp}, l'ensemble des professeurs utilisant ces salles spécialisées. Soit S_{lc}, l'ensemble des classes utilisant ces salles. Nous avons alors :

$$\sum_{i \in S_{lp}} \sum_{j \in S_{lc}} x_{ijk} \leq l_\sigma \quad (\forall k \; k = 1, \ldots, \xi).$$

c) Critiques

Seul un expert (informaticien connaissant les emplois du temps) peut établir ces équations. Chaque établissement a sa propre fonction critère ou économique. Supposons que nous voulons ajouter de nouveaux types de contraintes [FER 85],[FER 86]. Il faut alors redéfinir cette fonction. Il en est de même quand il s'avère nécessaire de modifier certains paramètres.

Le vrai problème réside dans la taille des données et des contraintes. Les méthodes de résolutions linéaires ne convergent pas forcément dans un temps acceptable. La matrice d'incidence est en général dégénérée.

Il existe des essais de résolution du système linéaire par diverses méthodes de recherche opérationnelle. Ils n'éliminent pas les défauts de taille, de rigidité, ...[TRI 84].

Le problème ci-dessus est simplifié. Tripathy [TRI 80, TRI 84] utilise cette modélisation avec des étudiants. Il introduit alors une formulation où les étudiants sont rassemblés par groupe. Ceci limite la taille du problème. Sa complexité est donc limitée. Ces essais sont présentés plus loin dans ce chapitre.

D'autres tentatives modélisent le problème par des équations non linéaires sur des variables bivalentes (0-1) [WHI 75]. Ils sont encore plus complexes à résoudre.

A.3.4 Diverses Techniques

Les méthodes ci-dessus incluent des hypothèses restrictives. Le nombre de composants est limité à deux. Le graphe est biparti. La matrice du graphe est unimodulaire. Les contraintes sont de types particuliers. Ces hypothèses sont difficilement conciliables avec des cas réels, même simples. Il est nécessaire de posséder une certaine souplesse de description. Il faut que la résolution soit facilement évolutive.

Le problème général est classé comme faisant partie de la classe NP-complet [GON 85]. C'est pourquoi, des techniques sont apparues depuis plusieurs années. Elles ne cherchent pas la solution exacte. Mais elles essaient de trouver par diverses méthodes, une solution approchée donc non exacte.

Beaucoup d'applications utilisent ces techniques. Ils autorisent une certaine souplesse dans la résolution. Des problèmes particuliers ont été résolus soit par des algorithmes approchés, soit par des algorithmes d'énumération, soit par des techniques d'intelligence artificielle. Nous passons en revue les nombreuses approches existantes. En général, elles sont basées sur les modèles vus précédemment.

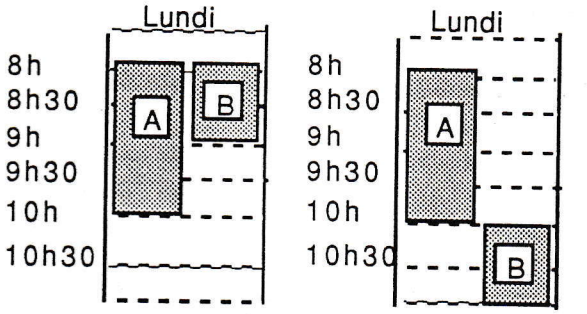
a) Algorithmes approchés

Ces algorithmes tiennent compte d'une des difficultés principales du problème d'emploi du temps. Il s'agit d'un système "surcontraint". Celui-ci ne possède, en général, aucune solution. Dans ce cas, il faut donner une solution approximative. Celle-ci doit satisfaire le plus possible tous les intéressés.

Il est intéressant de partir d'une solution satisfaisant un ensemble de personnes et un minimum de contraintes. Puis nous l'améliorons par itérations successives. Souvent, dans le problème d'emploi du temps, il n'existe que des solutions considérées comme bonnes par l'utilisateur (qui a souvent tendance à exagérer ces contraintes). Ceci nous amène à la notion de résolution par amélioration [AUS 76]. Elle est opposée à la notion de placement absolu ou définitif.

Le mécanisme d'amélioration est basé sur des techniques d'échange. Quand nous ne pouvons plus mettre de cours par manque de places, des cours sont échangés jusqu'à trouver une place disponible. C'est-à-dire une solution satisfaisante. Une manière de faire [LAP 84] est de chercher une solution libre de conflit. Si un blocage se produit, certains cours sont réassignés tout de suite ou plus tard. Ils sont remis en queue de liste des cours à planifier. Une fois un emploi du temps trouvé, une fonction de coût est appliquée. Elle permet de connaître l'échange qui fournira la meilleure amélioration.

Le travail de Smith [SMI 76] porte sur une application médicale. Son principe est de chercher d'abord une solution, même incomplète. Les temps de repos et de travail des équipes d'infirmières sont planifiés. Mais la solution incomplète peut violer certaines contraintes. Afin d'éliminer ces conflits, il améliore ensuite son résultat par échange.



Avant : recouvrement Après : glissement

Figure 13. Heuristique de glissement

Le principe d'Aust [AUS 76] est de faire une itération avec un test de choix. A chaque choix, les conflits doivent diminuer.

Ferland [FER 85] itére sur deux sous-problèmes. Le premier est la formation de groupes d'élèves. Le deuxième est leur ordonnancement. Pour chacun des sous-problèmes, une fonction objectif est à minimiser. Cette fonction intégre des pénalités pour les contraintes en conflit. Elle prend des coûts d'assignation pour les préférences des professeurs. Un réordonnancement a lieu sur un ou deux cours simultanément. Il est résolu par une procédure d'échange.

Rappelons la définition informelle d'un critère suivant Roy et Bouyssou [ROY 88] : "... un critère vise à résumer, à l'aide d'une fonction, les évaluations d'une action sur diverses conséquences pouvant se rattacher à un même "axe de signification" concret".

Certains logiciels permettent suivant un critère de déterminer la meilleure place d'une façon définitive. Le problème devient celui du choix du critère. Ceci n'est pas aisé. Il en existe de multiples et pas toujours explicités.

D'autres méthodes reposent sur des systèmes de pondération pour le choix des affectations. Une priorité est attribuée à chaque cours. Elle est fonction de certains critères. A partir de là, nous associons un poids à chaque composant. Ce poids dépend des cours de ce composant [SAB 86]. Le guidage peut, par ce biais, se faire interactivement ou automatiquement. Etape par étape, des possibilités sont proposées par [LAM 85].

De nombreux articles se basent sur ce type d'algorithme bâti autour d'une fonction objectif ou d'un critère d'optimisation. Ferland, par l'extension de son modèle [FER 85], [FER 86], [AUB 86] montre qu'il faut recentrer cette fonction pour introduire de nouvelles contraintes.

Le temps d'exécution peut être acceptable pour des tailles moyennes. Malheureusement, nous n'obtenons pas toujours une solution. L'approche n'est pas appropriée pour de gros volumes de données évolutives.

Les algorithmes approchés sont intéressants. Nous ne recherchons pas une solution nécessairement exacte. C'est une solution incomplète satisfaisant des critères. Cependant, les critères ne sont pas toujours très clairs. La notion d'optimum est très variable suivant les établissements.

Nous remarquons aussi qu'utiliser ces méthodes peut revenir à s'en servir pour la coloration de sommets d'un graphe. Le problème de la détermination approchée du nombre chromatique est aussi un problème NP-complet [GON 85]. Nous comprenons les difficultés rencontrées.

b) Algorithmes de Recherche Opérationnelle

Diverses techniques de recherche opérationnelle ont été utilisées pour prendre en compte tous les choix possibles pour les affectations. Pour une vue d'ensemble de ces techniques, il est possible de se reporter à [GON 85] ou [POR 88].

i) Procédure de Séparation et d'Évaluation

Une première technique utilise des procédures de séparation et d'évaluation. Ce sont des procédures d'énumération implicites. Le problème est séparé en sous-ensembles de problèmes. Chaque sous-ensemble est indépendant des autres. La résolution est modélisée par une arborescence. Chaque sous-problème est un noeud.

Nous évaluons les solutions de chaque problème. Si un problème a la meilleure évaluation, nous le séparons en sous-problèmes indépendants. Le processus recommence.

Jusqu'à présent, cette technique donne les meilleurs résultats pour des problèmes de grandes tailles fortement contraints. Elle est appliquée particulièrement aux problèmes NP-complets [LAU 71].

Laurière [LAU 74a] utilise cette technique. Il utilise deux principes de séparation. Le premier principe de séparation affecte les demi-journées aux professeurs. Le second principe attribue les heures suivant un tableau de disponibilités. Il utilise des marges de placement. Elles sont recalculées à chaque instant. Elles ne doivent pas être négatives.

La technique se heurte à des problèmes de taille de stockage. La gestion de la mémoire rend délicate les opérations de remontée dans l'arbre.

Exemple :

Définissons les marges μ_m avec les constantes σ_m et L_m selon [LAU 71] :

σ_m correspond à la charge du composant m. Un composant est soit un professeur, soit une classe, soit une salle. Si m est un professeur, i et j représentent les classes et les salles.

$$\sigma_m = \sum_i \sum_j x_{ijm} \quad \forall \ m \ ; \ m \text{ est un professeur, une classe ou une salle.}$$

La valeur L_m s'associe aux libertés de m. La marge μ_m est la différence entre le nombre de périodes disponibles et le nombre de périodes à assurer :

$$\mu_m = ||L_m|| - \sigma_m \text{ pour le composant m.}$$

Nous pouvons constater que trois cas de marges s'en dégagent :

1) $\mu_m > 0 \Rightarrow$ La marge de l'élément m est strictement positive. Il existe plusieurs possibilités de placement. Cet élément n'est pas prioritaire.

2) $\mu_m = 0 \Rightarrow$ La marge de l'élément m est nulle. Ses possibilités de cours correspondent exactement à sa charge de travail. Cet élément est prioritaire.

3) $\mu_m < 0 \Rightarrow$ L'élément m est impossible à placer. Le nombre de cours qu'il doit faire est plus important que ses disponibilités.

Cette notion de priorité de placement se trouve dans beaucoup de méthodes [CHA 89].

ii) Relaxation Lagrangienne et Méthode du sous-gradient

Nous survolons les différents essais de Tripathy [TRI 84]. Ceci introduit brièvement les notions de relaxation Lagrangienne et de méthode de sous-gradient. Le lecteur peut se référer à [GON 85] pour de plus amples informations sur ces méthodes.

L'intérêt de cette présentation provient de la manière mathématique de résoudre ces problèmes. Nous pouvons en dégager l'avantage de rapidité. Nous remarquons aussi les inconvénients dus à une intégration trop forte et figée pour un établissement donné.

Dans le problème d'emploi du temps, nous trouvons plusieurs sortes de relaxation de contraintes. Le principe de la relaxation est le fait de plonger le problème à résoudre dans un autre plus général. Il est plus facile à résoudre. Il est obtenu en éliminant un ensemble de contraintes. Nous calculons la solution du nouveau problème. Celle-ci est une solution incomplète du problème initial. Puis les contraintes relaxées sont vérifiées.

Par exemple, nous relaxons les conditions d'intégrité sur les variables. Celles-ci sont les conditions imposant une variable à être entière.

Une autre méthode est de maintenir l'effet de cet ensemble de contraintes relaxées

en les "dualisant". Nous leur donnons un coût de relaxation. Ce problème forme un problème de relaxation Lagrangienne.

Nous allons présenter cette méthode. Le but de cette présentation est de bien mettre en évidence que l'application de cette méthode aux problèmes d'emploi du temps est complexe et demande un savoir qui n'est pas à la portée de tous (donc peu naturel). Nous constatons aussi que c'est un système très intégré. Modifier le problème revient à refaire une analyse ardue.

Exemple : Soit un problème (PE) en Programmation Entière Linéaire :

$$(PE) \qquad \max_{x} cx, \text{ tel que } x \geq 0,\ Ax \leq b,\ Bx \leq d,\ x = (x1, \ldots, xn),\ \forall j,\ xj \text{ entier}$$

Le problème de relaxation transforme (PE) en (ΠE) :

$$(\Pi E) \qquad \max_{x} cx, \text{ tel que } x \geq 0,\ Ax \leq b,\ Bx \leq d,\ x = (x1, \ldots, xn)$$

La contrainte d'intégrité ($\forall j$, xj entier) est vérifiée après résolution de (ΠE). Une solution de (ΠE) qui vérifie cette contrainte est aussi une solution de (PE).

Un problème de relaxation Lagrangienne peut transformer (PE). ($Ax \leq b$) est dualisée avec un vecteur λ positif ou nul. Le problème est :

$$(PER_\lambda) \qquad w(\lambda) = \max_{x} (cx + \lambda.(b - Ax)) \text{ tel que}$$
$$x \geq 0,\ Bx \leq d,\ x = (x1, \ldots, xn),\ \forall j,\ xj \text{ entier}$$

Dans ce problème, pour toute solution de (PE), λ est positif ou nul, b - Ax aussi. Donc pour toute solution de (PE), $\forall x$, $w(\lambda) \geq cx$. Il en est de même pour la solution optimale de x ou x^*. Pour le choix optimal de λ ou λ^*, nous avons : $w(\lambda^*) \geq cx^*$.

Le vecteur doit être choisi tel que la solution optimale de (PER_λ) se rapproche de celle de (PE). Idéalement, le choix de λ doit optimiser la solution du problème :

$$(D) \qquad \min_{\lambda \geq 0} w(\lambda)$$

Le problème (D) est le problème dual de (PE) en respect des contraintes ($Ax \leq b$).

Il est intéressant de trouver une solution de (D), même incomplète de (PE). Celle-ci fournit une très bonne borne pour les solutions de (PE). La résolution de ce problème dual ne se fait pas par une méthode de programmation linéaire. Car elle fournit une solution exacte à des coûts élevés. Mais elle se fait par une méthode de sous-gradient.

Le sous-gradient γ de w au point λ fournit une très bonne direction de déplacement. Si la fonction w est différentiable, nous prenons le gradient. Dans le cas contraire, nous choisissons le sous-gradient.

Rappelons que $\gamma = (\gamma_1, \ldots, \gamma_m)^T$ est un sous-gradient de w au point λ si, $\forall\ \lambda' \geq 0$, nous avons : $w(\lambda) \geq w(\lambda') + (\lambda - \lambda').\gamma$.

Un sous-gradient évident est la contrainte relâchée (Ax - b). x est une solution quelconque de (PER_λ).

L'idée du sous-gradient est de générer une séquence λ^r. Il est montré [GON 85] que :

$$\lambda^r_{i+1} = \max \{\lambda^r_i + \theta_r . \gamma^r_i, 0\}, \text{ pour } i = 1,.., m$$

Tripathy reprend les travaux existants sur la recherche de θ_r et de γ^r. En résumé [TRI 84][GON 85], nous obtenons :

$$\gamma^r = Ax^r - b,\ \lambda^{r+1} = \max \{ \lambda^r + \theta_r.\gamma^r, 0\},\ \theta_r = \rho_r.(w\ (\lambda^r) - w^*) / |\gamma^r|^2$$

où $w^* > \min w(\lambda)$ et $0 < \rho_r \leq 2$.

Les ρ_r sont générés par différentes méthodes (par pas d'unité, par dichotomie, ...).

La vitesse de convergence est très lente. La séquence $w(\lambda)$ n'est pas monotone. Sa progression par sous-gradient ne garantit pas sa monotonie. Il est plus simple de calculer une borne. C'est souvent une solution de la fonction objectif pour une certaine valeur λ. Nous utilisons alors une méthode de "Branch and Bound" pour trouver une solution optimale. Cette procédure intervient dans le choix des variables à positionner à 1 ou à 0. Nous cherchons les "meilleures" solutions à partir de la borne.

Prenons l'exemple exprimé par un système nommé PLE de variables bivalentes :

$$\text{Max.}_i \quad \sum_k \sum_i C_{ik}\, y_{ik} ,$$

$$\sum_{k=1}^{\xi} y_{ik} = k_i \ (i = 1, ..., q), \qquad (P\text{-}I)$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \ (k = 1, ..., \xi), \qquad (P\text{-}II)$$

$$\sum_{i \in S_v} y_{ik} \leq 1 \ (v = 1, ..., r\ ;\ k = 1, ..., \xi), \qquad (P\text{-}III)$$

$$y_{ik} = 0 \text{ ou } 1$$

Cet exemple est un emploi du temps. Il planifie r groupes d'étudiants. $y_{ik} = 1$ indique que l'étudiant i a cours à la période k.

(P-I) exprime la contrainte sur le nombre de périodes (k_i) requis par semaine et par étudiant.

(P-II) est la contrainte pour la disponibilité des salles. Le nombre de places des salles est limité à l_k à chaque période de temps k.

(P-III) est la contrainte générale pour chaque étudiant i de chaque groupe S_v.

Tripathy remarque que le problème décrit par les contraintes (P-I) et (P-II) est unimodulaire. Ainsi les contraintes (P-III) sont relaxées. Le problème devient PLE_λ :

$$\text{Max.} \quad \sum_i \sum_k C_{ik} y_{ik} + \lambda_{kv}\Big(1 - \sum_{i \in S_v} y_{ik}\Big) \; [v = 1, \ldots, r \; ; \; k = 1, \ldots, \xi]$$

$$\sum_{k=1}^{\xi} y_{ik} = k_i \; (i = 1, \ldots, q),$$

$$\sum_{i=1}^{q} y_{ik} \leq l_k \; (k = 1, \ldots, \xi),$$

$$y_{ik} = 0 \text{ ou } 1$$

Dû au fait de l'unimodularité, le problème a des propriétés d'intégrité. Les y_{ik} ont 0 ou 1. Il est résolu efficacement.

Tripathy continue et applique la méthode des arcs non-conformes (*out-of-kilter* en anglais). Trouver un flot au problème relaxé s'avère rapide sur le plan d'exécution. Mais il n'apporte pas d'amélioration en terme de réduction de conflits.

Plusieurs avantages existent. Il y a un modeste espace d'occupation mémoire. L'algorithme est simple. Il résoud plus facilement le programme linéaire que d'autres méthodes basées sur le Simplex.

Pour cette méthode des arcs non conformes, le problème relaxé se change en introduisant des capacités et de nouvelles variables. La contrainte générale a été relaxée.

$$\text{Min} \quad \sum_i \sum_k (-C_{ik}) y_{ik} \text{ tel que}$$

$$\sum_{k=1}^{\xi} -y_{ik} + \alpha_i = 0, \qquad \text{avec } k_i \leq \alpha_i \leq k_i$$

$$\sum_{i=1}^{q} y_{ik} - \beta_k = 0, \qquad \text{avec } 0 \leq \beta_k \leq l_k$$

$$\sum_{i=1}^{q} \alpha_i - \sum_{k=1}^{\xi} \beta_k = 0 \qquad \text{et } 0 \leq y_{ik} \leq 1$$

Le but est de maintenir le flot nul à chaque sommet. Les sommets sont les contraintes. Les arcs représentent les variables.

Cette technique résoud le problème (PLE_λ) à un λ_{kv} donné. Malheureusement à chaque changement des coefficients λ_{kv}, il faut un temps de calcul considérable. La place consommée est importante. De plus, le nombre d'insatisfactions de contraintes ne décroit plus après environ dix d'itérations. Tripathy introduit, à ce moment, sa méthode de "Branch and Bound". Elle porte sur le choix des variables à positionner à 0 ou 1.

En conclusion, nous constatons que le problème est traité par un système intégré et très complexe. Rajoutons un nouveau type de contraintes. L'analyse est à refaire et tout est à recalculer.

Cependant, si ce système est presque figé, il a l'avantage d'être efficace. Il ne prend

que peu de volume dans les données.

D'autres applications ont été guidées par les expériences des concepteurs manuels d'emploi du temps.

Maintenant, nous présentons des techniques plus récentes basées sur des systèmes experts, des heuristiques.

c) Couplage Base de données et Base de connaissances

Plusieurs systèmes experts ont été expérimentés [VAL 84] sans toutefois donner des résultats intéressants.

D'autres systèmes experts plus performants ont fourni des bons résultats. Ils traitent des cas d'université. Leur originalité se place dans le couplage d'une base de connaissances avec une base de données. C'est donc le couplage d'un système expert et d'un S.G.B.D.. Les propriétés de chacun sont gardées.

L'apport de l'approche système expert est connu. Il permet d'utiliser la connaissance de l'expert. La connaissance et les données peuvent évoluer. Plusieurs applications se sont développées dans ce sens [SCH 87].

Le recours aux bases de données permet :

1) d'avoir une gestion efficace des données sur disque,

2) de déduire certaines informations,

3) d'avoir une garantie minimale de cohérence [HAN 88], [HER 86].

La première application, Sedlex, [EME 87] utilise une base de données orientée objet-attribut-valeur. Les règles et les faits sont stockés dans le même formalisme. Ils profitent ainsi des mêmes fonctionnalités.

Une seconde application [HAN 88] de SIAD (Système Interactif d'Aide à la Décision) met en évidence l'intérêt d'avoir une base de données. La connaissance est hiérarchisée.

La différence essentielle entre Sedlex et l'application de Hanachi porte sur la gestion des règles et des faits. Ces derniers sont en plus grand nombre dans les emplois du temps. Dans le SIAD, ils sont les seuls à être manipulés par le S.G.B.D..

La résolution utilise des priorités aux enseignements. Elle calcule des coefficients de mobilité aux cours placés. Ainsi, un ordre est établi suivant ces priorités, sur le placement des différents cours. Lors de blocage, une heuristique est utilisée. C'est celle de glissement guidée par les coefficients de mobilité.

L'attrait principal de ce type de méthode est de tirer profit du S.G.B.D.. Il résoud le handicap du volume des données et de leur gestion.

Hanachi utilise simplement l'heuristique de glissement (cf A.3.4 a). Ceci montre l'intérêt des heuristiques pour diminuer le caractère combinatoire [HAN 88].

d) Heuristiques

Maintenant, nous présentons un panorama des différentes heuristiques rencontrées. Une étude a été effectuée après l'utilisation pendant quelques années d'un logiciel dans un hôpital. Il a démontré que l'emploi d'heuristiques est plus "digeste" que celui de modèles mathématiques. Les heuristiques sont aussi plus faciles sur le plan de l'expression [SMI 79]. Seules les principales heuristiques sont citées ici.

La plus employée est celle du glissement. Elle peut être combinée avec des coefficients de mobilité ou de priorité. N'importe quelle affectation est remise en cause. L'avantage est que nous ne remontons dans la résolution que d'un seul niveau.

Les cours peuvent être assignées à plusieurs places possibles. Il existe plusieurs heuristiques dites de "priorité". Elles permettent de savoir quelle place il faut assigner en premier. Ainsi, la plus prioritaire peut se calculer d'après un système de poids. Nous pouvons aussi avoir la place reliée à un minimum de contraintes.

Dans le "Manchester Business School" [BAR 78], chaque étudiant doit choisir six options. Des places réservées dans le temps correspondent aux options. Seulement le modèle ne satisfait pas tout le monde. Certains élèves ont toutes leurs options satisfaites. Ce qui n'est pas le cas pour d'autres. Barham décrit une heuristique permettant de ne prendre en compte que quatre options sur six.

Carter [CAR 86] montre l'équivalence entre le simple modèle d'emploi du temps et celui de la coloration de sommets. Dans ce cadre, il propose un survol de différentes heuristiques. Précisons que le **degré** d'un sommet est le cardinal de l'ensemble des arêtes reliant ce sommet. Les couleurs sont numérotées par des entiers à partir de 1 :

1) "Le plus grand degré d'abord". Le sommet avec le plus haut degré est coloré avec la couleur admissible de plus petit numéro. Ceci ne doit pas créer de conflit.

2) "Les plus grands degrés d'abord". On colore tous les sommets de plus haut degré pouvant avoir la plus petite couleur.

3) "Les plus grands degrés d'abord, récursivement". Cette méthode est identique à la précédente. Mais les sommets coloriés sont enlevés. Ceci implique un nouveau calcul des degrés. Un nouveau tri des sommets est fait après chaque coloration.

4) "Le plus grand degré modifié d'abord". Une idée simple est ici mise en oeuvre. Un sommet est d'autant plus critique à colorer que ses voisins le sont.

La propriété d'un noeud à être critique est définie par son degré modifié. La définition du degré modifié est :

Degré modifié d_{k+1} du sommet v_i : $d_{k+1}(v_i) = \Sigma\, d_k(v_j)\ \forall\ v_j$ voisins de v_i ;
$d_0(v_i)$ étant les degrés de chaque sommet. d_{k+1} est ensuite normalisé.

Il est démontré [CAR 86] que les valeurs d_k convergent vers le vecteur propre de la matrice d'incidence. Cette heuristique est meilleure que celle du simple "plus grand degré". Elle donne moins de couleurs. Elle est plus chère autrement en temps de calcul.

5) "Le plus grand 1-degré modifié d'abord". On se limite aux degrés modifiés $d_1(v_i)$. Les récursions suivantes apportent peu d'amélioration.

6) "Le plus petit degré en dernier, récursivement". Cette heuristique est la même que le numéro (3). Cependant, le sommet recherché en premier est celui qui a le degré le plus petit. Ce sommet est alors placé en fin de la liste des sommets à colorer. Une fois la liste construite, le premier sommet colorié est le premier en liste.

7) "Le moins grand en dernier avec échange". L'échange peut aussi s'appliquer aux autres techniques.

Supposons un conflit inévitable avec le sommet à colorer c_i. Il faut rechercher une couleur k_j pour laquelle il n'existe qu'un sommet c_j en conflit avec c_i. Si possible la couleur k_j est attribuée à c_i. c_j est recoloré.

Sinon on cherche un ensemble C_r de sommets critiques avec c_j. C'est-à-dire qu'il serait en conflit s'il avait la même couleur. Les sommets de C_r ne doivent pas être critiques avec c_i. A ce moment il y a lieu à :

- un échange entre la couleur de C_r et k_j,
- c_j prend cette couleur k_r (de C_r),
- c_i prend la couleur k_j.

Si aucun des cas ci-dessus ne s'applique, une couleur est créée. Cette heuristique est légérement plus chère que les autres méthodes. Elle utilise moins de couleurs.

8) "Le plus grand degré avec saturation d'abord". Les sommets non coloriés sont triés par le nombre de voisins coloriés de façon différente et par leur degré dans le sous-graphe non colorié. Les sommets qui ont le plus haut nombre ont le moins de choix possible. Ainsi celui parmi eux qui aura le plus grand degré est le plus dur à colorer. Cette heuristique est celle qui utilise le moins de couleurs.

Ces heuristiques ne donnent pas forcément le nombre chromatique. Elles essaient de l'approcher. Les heuristiques ont le gros avantage de fournir une solution rapidement. Celle-ci n'est pas forcément la "meilleure". Des retouches manuelles sont alors possibles.

Signalons aussi que les solutions peuvent être écartées par certaines heuristiques. Elles éliminent a priori certaines solutions partielles [DEG 81], [DES 71]. Un autre problème pour les heuristiques est celui de la prise en compte de la diversité et de l'importance des contraintes.

A.4 Conclusion

Nous avons survolé les modèles, méthodes et techniques principalement utilisés. A partir de ce survol, nous avons dégagés les limitations des résolutions. Le nombre de composants est limité. La matrice d'incidence est totalement unimodulaire. La durée des cours est unitaire. Le stockage n'est pas simple. La gestion des données augmente considérablement le temps de résolution. Il faut rechercher une formulation précise. L'évolution des données peut signifier de tout refaire. Le problème a une résolution rigide. La combinatoire est trop grande. etc.

Nous constatons que les "points noirs" du problème d'emploi du temps sont :

- Des données trop diversifiées (en particulier les contraintes),

- La nature combinatoire du problème,

- L'aspect grande dimension et fortement contraint du problème,

- L'utilisation de critères flous, mal définis,

- Les changements intempestifs.

Il est nécessaire de disposer d'un système "programmable". Il faut un langage déclaratif. Il doit permettre une évolution facile des données. Ce langage doit aussi avoir une bonne gestion de l'indéterminisme. Il doit intégrer une gestion de la combinatoire. Il doit être facile de décrire des problèmes sans critère bien défini et sans une analyse poussée.

Nous présentons maintenant un langage intégrant ces besoins. Il posséde diverses propriétés. Les données sont prises en compte d'une manière dynamique et évolutive. C'est un langage déclaratif par nature. Il gére la combinatoire par un système intégré de retour arrière (backtrack). La notion de satisfaction de réseau de contraintes permet de tenir compte de critères flous, de critères mal déterminés, ...

CHAPITRE B

La Programmation en Logique avec Contraintes :

La réponse aux besoins de l'Emploi du temps?

Ce chapitre introduit la Programmation en Logique avec Contraintes et le langage CLEF v1.

Nous présentons le paradigme de la Programmation en Logique avec Contraintes. Celui-ci n'est qu'un cas particulier des modèles de Programmation par Contraintes.

La Programmation en Logique avec Contraintes se base sur les problèmes de Satisfaction de Contraintes et sur la Programmation en Logique.

Dans un premier temps, les problèmes de satisfaction de contraintes sont donc présentés. Ceci nous amène à comprendre la notion de langage de programmation par contraintes.

Dans un second temps, nous rappelons la Programmation en Logique. Elle induit déjà des langages de programmation par contraintes sur un domaine de calcul particulier. C'est l'univers d'Herbrand.

Dans un troisième temps, nous définissons un schéma de langage de Programmation en Logique par Contraintes (ou P.L.C.). Dans ces langages, plusieurs domaines de calcul existent. L'univers d'Herbrand est gardé. Souvent, un unique domaine $\mathcal{D}$ est ajouté. Ce domaine de calcul $\mathcal{D}$ diffère suivant les problèmes à résoudre. Ceci explique l'existence de

différents langages de la P.L.C..

Enfin, un langage particulier est présenté. Il intégre un domaine de calcul correspondant aux domaines finis discrets. Nous l'utilisons pour résoudre les problèmes d'Emploi du Temps.

B.1 Les problèmes de Satisfaction de Contraintes

Nous allons présenter la notion de satisfaction dans son ensemble. Il s'agit de construire un réseau où les sommets sont les variables et les arêtes sont des contraintes reliant les variables entre elles. Les valeurs des variables se restreignent à appartenir à un ensemble fini.

Il faut alors chercher des solutions qui satisfassent toutes les contraintes du réseau. Les différentes manières de satisfaire un réseau sont présentées dans une deuxième division.

Nous discuterons alors du lien entre satisfaction de contraintes et problème général d'emploi du temps.

B.1.1 Quelques notions de satisfaction de contraintes

Ces définitions sont structurées et classifiées en cinq divisions. Elles sont reprises à chaque fois qu'un langage de programmation par contraintes est décrit. Ainsi, nous fairons le lien entre le problème de satisfaction de contraintes et chaque langage.

a) Problème de satisfaction de contraintes et réseaux de contraintes

Un ***problème de satisfaction de contraintes*** se définit par un ensemble de variables $\mathcal{V} = \{v_1, ..., v_n\}$, un ensemble de contraintes $\mathcal{C}$ et un espace de recherche S.

Un ***réseau*** de contraintes décrit le problème à résoudre. Il est modélisé par un hypergraphe $R(\mathcal{V}, \mathcal{C})$. $\mathcal{V}$ est l'ensemble des sommets de l'hypergraphe qui représente l'ensemble des variables du réseau. $\mathcal{C}$ est l'ensemble des hyperarêtes qui représentent les contraintes de ce même réseau.

Suivant le choix des contraintes, nous pouvons avoir sur les mêmes variables $\mathcal{V}$ plusieurs réseaux.

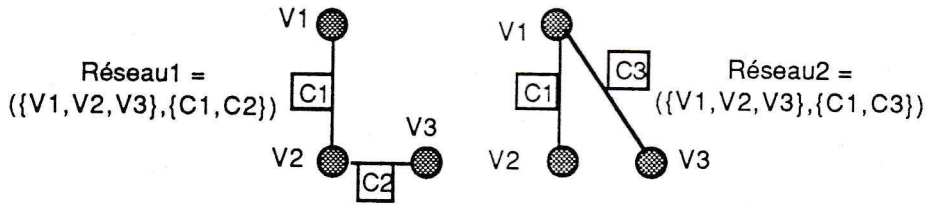


Figure 14. Différents réseaux

b) Domaine, variables sous domaine et Espace de recherche

Nous appelons ***domaine*** D_i associé à une variable v_i, un ensemble de valeurs où la variable est autorisée à prendre ses valeurs.

Par la suite, nous appellerons "Domaine d'une variable", l'ensemble des valeurs que

peut prendre une variable.

L'***espace de recherche*** S est un ensemble de points $s = <s_1, ..., s_n>$. C'est le produit cartésien des domaines des variables.

c) Contraintes, satisfaction d'une contrainte et domaine de calcul

Une ***contrainte*** relie entre elles un sous-ensemble de variables de $\mathcal{V}$. Elle caractérise un sous-espace de S.

Le ***domaine de calcul*** d'une contrainte $C \in \mathcal{C}$ est un univers $\mathcal{D}$. Le domaine de chaque variable est inclus dans cet univers.

Ainsi C est définie sur n variables par :

$$C : D_1 \times ... \times D_n \longrightarrow E_1 \times ... \times E_n$$
$$<v_1, ..., v_n> \longrightarrow C(v_1, ..., v_n).$$

et $E_1 \subseteq \mathcal{D}, ..., E_n \subseteq \mathcal{D}$.

La ***satisfaction*** d'une contrainte C sur n variables en un point $s \in S$ est telle que C(s) existe.

Si C(s) existe, nous disons que C est ***satisfaite*** au point s. Dans le cas contraire, C est ***insatisfaite*** au point s.

d) Méthode de satisfaction, Ensemble de rejet, Ensemble d'agrément

Pour mieux cerner les problèmes posés par la satisfaction de contraintes, nous introduisons quelques définitions qui nous semblent utiles.

La satisfaction d'une contrainte nécessite diverses ***méthodes*** (filtrage, élimination, ...). Nous présentons, par la suite, les différentes méthodes existantes.

Nous définissons un ***ensemble de rejet*** de C sur S par la méthode m, $S_m^{C'}$ tel que :

$\forall s \in S_m^{C'}$, s ne satisfait pas la contrainte C par la méthode m.

Nous définissons l'***ensemble d'agrément*** de C sur S par m, S_m^C tel que :

$\forall s \in S_m^C$, s satisfait la contrainte C par la méthode m.

$S_m^{C'}$ et $S_m^C = S$ sont complémentaires.

Nous appelons C' la contrainte ***complémentaire*** de C dans l'ensemble S.

Le ***déclenchement*** d'une contrainte est l'application d'une méthode. Elle teste la satisfaction de cette contrainte.

N.B. : *Remarquons que, dans le cas général, il est plus simple de trouver les éléments de l'ensemble $S_m^{C'}$ que ceux de l'ensemble S_m^C.*

e) Solution. Satisfaction de réseaux et méthodes associées

La ***satisfaction du réseau*** consiste à trouver tous les points de S satisfaisant toutes les contraintes C. L'ensemble de ces points est noté S^{C}_{m}.

Tout point s est une ***solution*** du réseau $R(\mathcal{V}, C)$ si s est un point de S et fait partie de l'ensemble d'agrément de toutes les contraintes du réseau.

Le but du problème de la satisfaction de contraintes est de chercher toutes ces solutions. Nous pouvons regrouper toutes les méthodes des différentes contraintes en une seule méthode dite globale. Le but revient donc à appliquer une méthode globale pour satisfaire tout le réseau.

Par exemple, la recherche des solutions se fait par une ***énumération*** d'instanciation de variables. Nous énumérons les différentes valeurs de leur domaine. Cette énumération se fait plus ou moins intelligemment par des heuristiques ou par des stratégies.

B.1.2 Quelques techniques ou méthodes de satisfaction

Ce paragraphe décrit les méthodes employées en Intelligence Artificielle pour effectuer la satisfaction de contraintes. Les différentes approches ont toutes un même objectif. Il faut réduire l'espace de recherche.

a) Technique de propagation locale

Cette technique est introduite avec Sussman et Steele [SUS 80]. Ils ont créé un langage intégrant cette technique. Ils l'appliquent principalement à la conception de circuits électriques.

Leur langage est déclaratif et maintient des relations entre les objets. Ces relations expriment des dépendances et représentent des contraintes. Nous savons alors pourquoi tel ou tel objet prend telle ou telle valeur.

Introduisons deux notions : les contraintes consommatrices et productives. Les contraintes consommatrices attendent que tous leurs arguments aient une valeur. Les contraintes productrices jouent un rôle actif dès qu'elles ont assez d'informations. Elles produisent des valeurs pour certains de leurs arguments.

Le principal atout de la propagation vient d'une utilisation des contraintes en tant que productrices et non plus en tant que consommatrices.

Une contrainte est décrite par une série de règles. Par exemple, X = Y + Z est exprimée en interne de la manière suivante :

si X et Y sont connus alors Z <- X - Y,
si X et Z sont connus alors Y <- X - Z,
si Y et Z sont connus alors X <- Y + Z.

Ce mécanisme s'appelle la **propagation locale de valeurs.**

Le principal intérêt de cette technique est que la résolution est dirigée par les données. Cependant, plusieurs limitations peuvent être trouvées :

1) Une contrainte est implantée comme un ensemble de règles. Il faut donc s'assurer qu'il existe une fonction inverse.

2) La propagation ne concerne que celle de valeur. C'est très insuffisant pour déduire toutes les valeurs satisfaisantes la contrainte.

Exemple : Soient les quatre variables V, W, X, Y et les contraintes suivantes :

$$V + W = X, W + X = Y.$$

Nous déduisons Y de V et W. Il en est de même pour V déduit de W et Y. Nous n'arrivons pas à déduire W de V et Y par cette simple propagation.

3) La contrainte disparaît de la résolution dès qu'elle est prise en compte. C'est-à-dire après son déclenchement.

Exemple :

Soient les deux variables X et Y. Elles prennent leur valeur dans l'ensemble {1, 2, 3}. Soit X = Y + 1, la contrainte qui les relie.

Nous déduisons immédiatement que la valeur 1 n'est pas valide pour X. La valeur 3 n'est pas non plus valide pour Y.

Après le déclenchement de la contrainte, les domaines sont reduits. Le domaine de la variable X devient {2, 3}. Celui de la variable Y devient {1, 2}.

Le nouvel espace de recherche est réduit à ({2, 3} x {1, 2}). Si nous énumérons les points, nous avons la fausse solution <2, 2> pour <X, Y>.

Pour résorber ces inconvénients, nous pouvons ajouter des informations redondantes. Nous pouvons aussi avoir des mécanismes de manipulation algébrique.

Le problème peut être aussi résolu par la méthode "**générer et tester**". Elle comporte deux phases. D'abord, nous générons de façon exhaustive toutes les valeurs de chaque objet. Puis, nous testons les contraintes.

b) Technique de cohérence de réseau

Cette technique essaie d'éliminer les incohérences [MON 74], [MAC 77]. Ces éliminations réduisent l'espace de recherche. Ceci nécessite l'utilisation de différentes techniques de vérification de cohérence.

Nous distinguons plusieurs techniques de cohérence : la cohérence de noeud (ou de variable), d'arête (ou de paire de variables) et de chemin (ou de k variables) appelée k-cohérence.

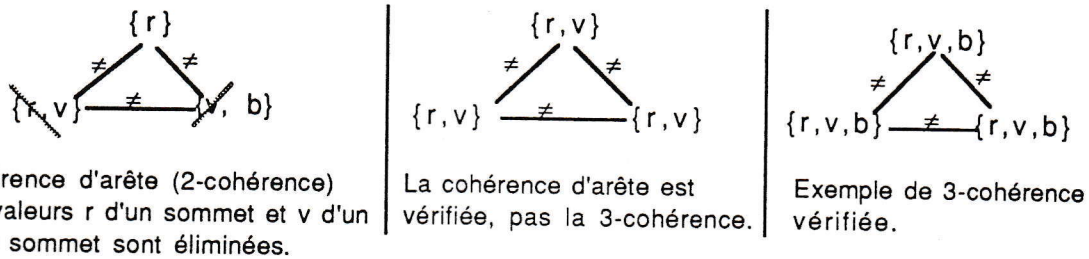
Ces incohérences sont des valeurs du domaine attachée aux différentes variables du réseau. Le principe de la k-cohérence consiste à enlever toutes les valeurs incohérentes impliquant tous les sous-ensembles de k variables.

Par exemple, la cohérence d'arête est appelée 2-cohérence. Elle vérifie toute cohérence sur les variables deux à deux. La propriété suivante doit rester vérifiée :

Soit $\mathcal{V}$ l'ensemble des variables et C celui des contraintes,

$\forall v_i \in \mathcal{V}, \forall v_j \in \mathcal{V}, \forall C \in C$
C relie v_i et v_j,
$\forall d_1 \in D_i, \exists d_2 \in D_j$, C (d_1, d_2) satisfait.

La figure suivante exprime cette notion :



Cohérence d'arête (2-cohérence) Les valeurs r d'un sommet et v d'un autre sommet sont éliminées.

La cohérence d'arête est vérifiée, pas la 3-cohérence.

Exemple de 3-cohérence vérifiée.

Figure 15. Diverses cohérences

Ces travaux ont été décrits par Montanari [MON 74]. L'algorithme de filtrage de Waltz [DIN 86] est un cas particulier de la technique de cohérence d'arête. Elle a une complexité polynomiale [RIT 88] [FEL 90]. Rit [RIT 88] utilise ces idées pour ses travaux de planification à l'aide de contraintes temporelles.

Le test de cohérence est local au niveau d'une contrainte. La vérification est globale au niveau du réseau. Malheureusement, l'algorithme de vérification de cohérence globale a une complexité exponentielle [RIT 88].

c) Technique de recherche dans un arbre

D'autres techniques se sont développées. Elles sont basées sur le parcours d'un arbre. Ces techniques s'appuient sur une idée simple. Il s'agit de **générer** des valeurs aux variables. Puis il faut vérifier (ou **tester**) les contraintes en cause. Cette opération est répétée jusqu'à l'obtention d'une solution. Cette dernière satisfait toutes les contraintes.

Si une insatisfaction est détectée, ces mécanismes permettent de revenir sur la dernière variable instanciée. Nous lui changeons alors de valeur d'instance. Ce "retour arrière" se nomme "**backtrack**".

Une amélioration de ces mécanismes est de créer une méthode hybride : backtrack + test de cohérence (en anglais "**forward checking**"). Lors de l'instanciation d'une variable, cette méthode élimine des valeurs. Elles sont incompatibles pour les autres variables restant à instancier.

Il est inutile d'instancier toutes les variables pour détecter une incohérence. Si le domaine d'une variable devient vide, le réseau est incohérent. Cette méthode a donc une participation active dans la résolution.

Une autre approche est celle d'ALICE. ALICE est un langage développée par Laurière [LAU 78]. Il réalise un mélange entre satisfaction de contraintes et heuristiques. Ces

dernières choisissent les instanciations aux variables.

Ce langage se rapproche plus d'un système intégré. Il permet de spécifier les problèmes sous forme graphique et algébrique. Il dispose des mécanismes d'optimisation de Recherche Opérationnelle, d'une vingtaine d'heuristiques et de deux méta-heuristiques. Il utilise une énumération locale et globale. L'énumération se fait sur les variables d'une contrainte en soi. L'énumération globale porte sur toutes les variables du problème.

Sa résolution d'un problème se fait en trois phases :

1) Recherche d'une solution réalisable,

2) A partir de la solution de la phase 1, recherche d'une solution proche de l'optimum souhaité suivant un critère,

3) Recherche d'une solution optimale.

Cependant, ce langage est uniquement descriptif. Il n'a pas d'instruction. Ce n'est pas un vrai langage de programmation. C'est plutôt un système pour résoudre au mieux les problèmes combinatoires. Les heuristiques ne peuvent pas évoluer. Ainsi un expert ne peut pas exprimer la connaissance acquise sur un type de problèmes.

B.1.3 Lien avec le problème d'Emploi du temps

Le réseau de contraintes est constitué de variables, de domaines de valeurs pour ces variables et des contraintes.

Nous associons des variables aux cours. Chaque domaine constitue un ensemble de possibilités d'ordonnancement. Les contraintes représentent alors les diverses propriétés liées aux cours.

Nous identifions la satisfaction du réseau de contraintes R($\mathcal{V}$, $\mathcal{C}$) au but de la résolution du problème d'emploi du temps défini dans le paragraphe A.2.2.

Rappelons que ce but fait intervenir un hypergraphe H = [E, $\mathcal{F}$] sur les cours $\mathcal{F}$ à ordonner, un ensemble de places I et une fonction g à chercher de $\mathcal{F}$ dans I sous des propriétés P.

Soit n variables de cours dans le réseau. Nous prenons un espace de recherche S = I^n. L'ensemble de variables $\mathcal{V}$ représente les cours $\mathcal{F}$. Les propriétés P de g sont assimilées aux contraintes $\mathcal{C}$. La fonction g est alors la méthode globale de satisfaction.

Feldman & Golumbic [FEL 89, 90] ont essayé les diverses techniques de backtrack et de "forward checking" sur des séries d'exemples d'emploi du temps.

Ils ont des structures de cours (professeur, matière, périodes). Leur problème consiste à ordonner ces structures pour chaque étudiant.

Les variables sont les cours des étudiants. Chaque domaine est l'ensemble des structures possibles.

Leur problème d'emploi du temps a une propriété similaire à celle (P0) de notre exemple illustratif en A.2.3. Elle représente les astreintes entre cours de composante

commune. Il y a aussi des indisponibilités (propriété (P1)). Nous trouvons encore des contraintes de limitation (propriété (P6)). Chaque élève doit faire et ne pas dépasser un quota d'heures.

Leur travail porte sur 3000 problèmes de 30 cours. Il met en évidence que l'ordre d'instanciation des variables joue un rôle très important dans le temps d'exécution. Ainsi le backtrack simple est avantageux pour peu de cours (environ 19). Au-delà, il faut utiliser la technique de "forward checking".

Plusieurs systèmes ont intégré cette notion de contraintes :

- Constraints [SUS 80] permet d'énoncer le problème sous forme de réseaux de contraintes. Leur méthode de satisfaction est la propagation locale,

- ALICE [LAU 78] déjà présenté,

- Molgen [STE 81] est un système basé sur la planification. Sa principale méthode rajoute des contraintes, même redondantes. Ceci élimine au plus tôt les mauvais choix. Cette méthode décide à quel moment une contrainte doit être déclenchée.

Ces systèmes expriment une variété de problèmes de même genre. Ils ne se restreignent pas à résoudre un seul problème. C'est la raison pour laquelle il est utile de posséder un langage de programmation par contraintes utilisant ces techniques. Ce langage est alors consacré au type de problèmes que l'on désire résoudre.

Pour les problèmes d'emploi du temps, ces langages doivent posséder les propriétés suivantes :

- Evolutivité des données,

- Prise en compte dynamique des données,

- Déclarativité,

- Bonne gestion de l'indéterminisme,

- Différentes stratégies ou heuristiques pour mieux contrôler la résolution du problème donc la recherche des solutions.

B.2 Une programmation par contraintes : la programmation en logique

Cette partie montre que la programmation en logique (P.L.) à travers Prolog, est une programmation par contraintes. Les notions de satisfaction de contraintes sont reprises. Ils sont comparées aux notions de la programmation en logique [COL 83].

B.2.1 Lien avec le problème de satisfaction de contraintes

a) Problème de satisfaction de contraintes et réseaux de contraintes

Un ***problème de satisfaction de contraintes*** en P.L. est constitué d'un programme et d'une requête. Il est composé de termes de la logique des prédicats du premier ordre et de variables. Les termes et variables prennent leur valeur dans l'Univers d'Herbrand. Je note celui-ci UH.

Définissons la notion de **règle**. La forme d'une règle est la suivante :

Règle : Préd.$_1$:- préd.$_2$, ..., préd.$_m$.

Si la partie droite préd.$_2$, ..., préd.$_m$ est inexistante, la règle est un **fait**. Si la partie gauche (Préd.$_1$) n'existe pas, la règle est appelée **but** ou **requête**. Elle correspond à la question. Par l'intermédiaire de la résolution de la P.L., elle s'associe à la pose de réseaux de contraintes à satisfaire.

Nous pouvons associer à chaque règle un réseau de contraintes. Chaque prédicat préd.$_i$ représente une contrainte. Si le prédicat Préd.$_1$ apparait dans la résolution, cela équivaut au déclenchement d'une contrainte. Dans ce cas, nous posons le réseau de contraintes, préd.$_2$, ..., préd.$_m$.

Un programme en P.L. est un ensemble de règles. C'est un ensemble de ***réseaux de contraintes***.

La syntaxe utilisée est celle d'Edimburg. Nous prenons une exception. Les variables sont préfixées par un astérisque. *v1 symbolise la variable V1.

Si un changement de syntaxe se produit, il sera énoncer en indiquant la raison.

b) Domaine, variables sous domaine et Espace de recherche

Le ***domaine*** des variables est l'univers d'Herbrand UH.

La résolution d'un problème en P.L. est la recherche de solutions vis à vis d'une question. S'il existe n variables, cette recherche se fait dans l'***espace de recherche*** S égal à UH^n.

c) Contraintes, satisfaction d'une contrainte et domaine de calcul

Nous savons qu'une contrainte relie un ensemble de variables. Elle est associée à un domaine de calcul. Le ***domaine de calcul*** en P.L. est l'univers d'Herbrand. Un prédicat définit une ***contrainte***.

La contrainte élémentaire correspond à une contrainte d'unification.

La ***satisfaction d'une contrainte*** est l'interprétation de cette contrainte dans l'ensemble {vrai, faux} avec réduction des domaines des variables reliées.

C est ***satisfaite*** si la valeur d'interprétation de cette contrainte est la valeur "vrai", ***insatisfaite*** si la valeur d'interprétation est "faux".

d) Méthode de satisfaction, Ensemble de rejet, Ensemble d'agrément

La ***méthode de satisfaction*** est l'**unification** de termes. Je la noterai $=_u$. L'unification est une résolution d'équations structurées [VAN 84]. L'unification résoud la contrainte (t1 $=_u$ t2) entre les termes t1 et t2. Nous appellons cette contrainte, **contrainte d'unification**.

L'ensemble de rejet d'une contrainte C sur UH^n est l'ensemble des valeurs de

UH^n dont la valeur d'interprétation de C est "faux".

L'ensemble d'agrément de C sur UH^n est donc l'ensemble des valeurs de UH^n dont la valeur d'interprétation de C est "vrai".

e) Solution. Satisfaction de réseaux et méthodes associées

La résolution d'un réseau de contraintes se base sur la résolution de Robinson. Le réseau se construit dynamiquement.

Rappelons le principe de la résolution en P.L. :

Soit P un programme, un ensemble de règles Rr, et R_0 la requête initiale. La résolution en P.L. est une machine abstraite. Elle est définie par une seule instruction que je note "->" :

état initial : G_i = <W1, :- t0, ..., tn.>

condition : $\exists$ Rj $\in$ P avec Rj $\equiv$ (L :- L1, ..., Lm.) et L $=_u$ t0 satisfait dans le contexte W1.

état final : G_{i+1} = <W2, :- L1, ..., Lm, t1, ..., tn.>

W1 est l'ensemble des variables à l'état initial. W2 est celui à l'état final avec le renommage et les substitutions induites par la contrainte d'unification.

Nous associons un espace de recherche à W1. Cet espace est réduit par la contrainte d'unification.

Soit W0 les variables de la requête R_0. Pour répondre à la requête R_0, la machine part de l'état initial <W0, R_0>. Cette machine passe par tous les états qu'elle peut atteindre en répétant l'opération de base "->" et en utilisant le programme P.

Nous appelons **dérivation** une séquence G_i. Le passage de deux requêtes consécutives se fait par l'opération de base : G_i -> G_{i+1}.

La résolution de la P.L. consiste à rechercher toutes les dérivations possibles à partir d'une requête.

Les réseaux sont induits par la résolution du programme P et de la requête R_0. Ainsi ***la satisfaction des réseaux*** se fait par la résolution de la P.L. et l'unification à travers ses dérivations. La satisfaction de chaque réseau correspond à chaque dérivation possible.

Deux cas d'arrêt sont détectés pour une dérivation :

- quand G_i -> <Wτ, ø>, la dérivation courante est **en succès**,

- quand $\forall$ Rj $\in$ P, Rj $\equiv$ (L :- L1, ..., Lm.), G_i = <Wi, :- t0, ..., tn.>
L $=_u$ t0 insatisfait avec Wi, la dérivation courante est **en échec.**

Les solutions d'une dérivation en succès est le produit cartésien des domaines des

variables de sa dernière requête.

Si une dérivation est en échec, la dérivation n'a pas de solution.

Les ***solutions*** du problème (P, R_0) sont l'union des solutions de toutes les dérivations en succès trouvées.

Si toutes les dérivations trouvées sont en échec, le problème initial est en échec, sinon c'est un succès.

B.2.2 Quelques caractéristiques de Prolog

a) Contrôle de la résolution

Reprenons les caractéristiques que doit avoir le langage de programmation par contraintes.

Premièrement, les règles du programme peuvent évoluer. Il existe des prédicats spécifiques permettant l'ajout, et la suppression de règles. Les règles sont exprimées d'une façon déclarative.

Le mécanisme de résolution du problème par la P.L. est par nature dynamique. Il permet de construire des réseaux de contraintes d'une façon évolutive.

Le backtrack permet de gérer l'indéterminisme.

Cependant deux inconvénients figent la stratégie de résolution.

Le premier est le mécanisme de profondeur d'abord. Le deuxième inconvénient provient de la chronologie du backtrack.

Des extensions à ce langage ont été introduites afin de mieux contrôler la résolution. Elles permettent de trouver les solutions d'une manière plus intelligente et essaient donc de pallier à ces inconvénients.

En général, ce contrôle se fait par une notion de coroutinage. Celle-ci permet au programmeur d'introduire des prédicats dans la résolution. Il le fait à des moments choisis, en fonction de certains critères. Les principales approches développées sont :

- Les prédicats "geler" et "dif" de Prolog-II [COL 83].

Ainsi "geler" permet de retarder l'évaluation d'un prédicat donné par rapport à une variable. Ces deux objets, variable et prédicat sont des paramètres de "geler".

Exemple : geler (*a, toto (*a)) permet de retarder l'évaluation de toto (*a), tant que *a n'est pas connu.

"dif" permet d'exprimer que deux termes inconnus ou non, resteront différents. Si ces termes sont structurés, la différence se fait aussi sur chaque élément de la structure de manière disjonctive.

- La déclaration "wait" dans MU-Prolog est attachée à une définition de prédicat. Elle permet de spécifier des schémas d'activation efficaces pour ce prédicat. Si l'appel

courant correspond à ce schéma, il est activé sinon il est retardé [NAI 83].

• Dans METALOG [DIN 84], l'adjonction de métaclauses permet d'optimiser le comportement du programme. Il a les contrôles par "gel" ou "activation". METALOG permet aussi de modifier partiellement la stratégie de "profondeur d'abord" de Prolog.

Ainsi, les échecs de chaque dérivation peuvent être détectés au plus tôt. La résolution est guidée par le programme décrit. Il ne suit pas une unique stratégie.

b) Lien avec le problème d'emploi du temps

Le Kang [LEK 90] présente un exemple de résolution d'emploi du temps à l'aide d'un langage de la P.L.. A ce langage, Le Kang intégre des méçanismes de "glissement" (cf. A.3.4 a). Il les utilise quand le backtrack devient trop coûteux. Il étend les formes des règles en ajoutant des quantificateurs ($\forall$, $\exists$) et des conditions (si ... alors).

Son problème d'emploi du temps est exprimé par un modèle à cinq variables :

- la variable cours. Son domaine est l'ensemble des cours à ordonner,

- la variable temps. Le domaine est l'ensemble des périodes,

- la variable salle. Le domaine est l'ensemble des salles existantes,

- la variable professeur. Le domaine est l'ensemble des professeurs et

- la variable programme d'enseignement. Le domaine est l'ensemble des cours requis et recommandés pour les différents programmes proposés aux étudiants.

Les contraintes (ou clauses) expriment des propriétés similaires à (P0), (P1), (P2), (P6) de l'exemple illustratif A.2.3. Ils s'écrivent dans les règles étendues de son langage.

Le Kang résoud ce problème d'une façon efficace ($\cong$ 1427 secondes, soit 23 minutes et 47 secondes) sur un grand volume ($\cong$ 2000 cours). Tous les cours de l'établissement traité ne sont pas placés.

Son langage est un dérivé de la P.L..

Son application a l'avantage d'utiliser un langage déclaratif.

L'inconvénient provient d'une utilisation simple de ces prédicats. Il n'utilise pas les extensions de la P.L.. Sa méthode de satisfaction ressemble alors à une recherche dans un arbre par un "générer et tester". De plus, il utilise un glissement des cours quand il ne peut plus placer les cours. Cette méthode de glissement n'est pas exacte et peut faire manquer la solution.

B.3 La Programmation en Logique avec Contraintes (P.L.C.)

Le fondement de ce langage est le même que celui de la P.L.. Il s'ouvre à des domaines de calcul autres que l'univers d'Herbrand. Nous reprenons les diverses notions des problèmes de satisfaction de contraintes. Puis nous introduisons des travaux faits pour étendre la résolution à une recherche de solutions approchées au problème posé.

B.3.1 Lien avec le problème de satisfaction de contraintes

Les termes et les variables de la P.L.C. sont les mêmes que ceux de la P.L..

a) Problème de satisfaction de contraintes et réseaux de contraintes

Des symboles relationnels Q sont introduits sur un univers $\mathcal{D}$. Nous nous donnons alors une réalisation [HOU 81]. Ces symboles relationnels Q sont des contraintes sur $\mathcal{D}$.

Un ***problème de satisfaction de contraintes*** en P.L.C. se compose d'un programme constitué de règles et d'une requête. Les règles utilisent des termes et des symboles relationnels Q.

La résolution de la P.L.C. construit des ***réseaux de contraintes*** à satisfaire.

b) Domaine, variables sous domaine et Espace de recherche

Le ***domaine*** de chaque variable est l'univers d'Herbrand. Soit une variable instanciée à un terme. Supposons que ce terme a une interprétation dans $\mathcal{D}$. Nous associons cette interprétation à une valeur possible pour la variable concernée.

Une ***variable sous domaine*** est une variable attachée à un domaine dont toutes les valeurs ont une interprétation dans $\mathcal{D}$.

Ainsi l'***espace de recherche*** est le produit cartésien des domaines des variables mises en jeu dans la résolution.

c) Contraintes, satisfaction d'une contrainte et domaine de calcul

Le ***domaine de calcul***, correspond à l'ensemble $\mathcal{D}$ ajouté. Ainsi, nous avons comme domaine de calcul, l'univers d'Herbrand UH et $\mathcal{D}$. Nous distinguons deux sortes de ***contraintes***.

Les premières sont les contraintes d'unification $=_U$ sur l'univers d'Herbrand. Les deuxièmes sont celles sur $\mathcal{D}$. Elles sont représentées par un symbole relationnel.

Les prédicats et clauses sont des contraintes à base de contraintes d'unification. Ainsi le programme P.L.C. utilise des règles de la forme :

Préd.$_1$:- préd.$_2$, ..., préd.$_m$, Q_1, ..., Q_v. où

Préd.$_1$ à préd.$_m$ sont des termes sur UH, Q_1 à Q_v des symboles relationnels sur $\mathcal{D}$.

La ***satisfaction d'une contrainte*** est l'interprétation de cette contrainte dans l'ensemble {vrai, faux}.

C est ***satisfaite*** au point s si la valeur d'interprétation de cette contrainte en ce point est la valeur "vrai". Elle est ***insatisfaite*** au point s si la valeur d'interprétation est "faux".

d) Méthode de satisfaction de contrainte. Ensemble de rejet. Ensemble d'agrément

Les différents types de connaissance sur $\mathcal{D}$ peuvent être : des propriétés algébriques de $\mathcal{D}$, des techniques de simplification d'expressions, des techniques de résolution de systèmes d'équations et d'inéquations dans $\mathcal{D}$ (Simplex par exemple), des techniques de vérification de cohérence par "forward checking", ...

La ***méthode de satisfaction*** se définit dans l'univers d'Herbrand par l'unification. Cette méthode se définit dans $\mathcal{D}$, par la connaissance que nous possèdons de ce domaine.

L'ensemble de rejet d'une contrainte C sur S est l'ensemble des valeurs de S dont la valeur d'interprétation de C est "faux" sur ces valeurs.

L'ensemble d'agrément de C sur S est donc l'ensemble des valeurs de S dont la valeur d'interprétation de C est "vrai" sur ces valeurs.

e) Solution. Satisfaction de réseaux et méthodes associées

i) Terminologie de méthodes en P.L.C.

Nous allons introduire quelques termes de la P.L.C.. Ils s'associent aux méthodes passées en revue dans B.1.2.

En règle générale, l'unification est étendue au domaine $\mathcal{D}$. La méthode globale de satisfaction correspond à la définition de **réduction** vue dans [LEP 90].

Le Pape et Ranson [LEP 90] déterminent deux sortes de réduction :

- la réduction passive. Une procédure de test est attribuée à chaque contrainte. Cette procédure est appelée quand toutes les variables de la contrainte sont instanciées. C'est une méthode "générer et tester",

- la réduction active. Elle se définit par la propagation locale de valeurs et la **saturation.**

La propagation locale de valeurs donne un schéma local de déclenchement à la contrainte. Si ce schéma est satisfait, une procédure locale déduit des valeurs pour les variables concernées (voir B.1.2). Sinon la contrainte est "gelée".

Elle permet de détecter les incohérences d'un réseau de contraintes. Elle s'apparente au "forward checking". Si elle ne détecte que partiellement ces incohérences, la connaissance intégrée pour $\mathcal{D}$ est alors incomplète. On dit que la saturation est **partielle.**

ii) Méthode de satisfaction de réseaux et solutions

Des réseaux sont induits par la résolution du programme P et de la requête initiale R_0. ***La satisfaction des réseaux*** se fait par la résolution de la P.L.C. et par la méthode de réduction choisie, donc à travers ses dérivations. La satisfaction de chaque réseau passe par chaque dérivation possible. Le principe de résolution des langages de la P.L.C. est :

Soit P un programme, un ensemble de règles Rr et R_0 la requête. La résolution en P.L.C. [BEN 90] transforme l'instruction "->" en :

état initial : G_i = <W1, :- t0, ..., tn., $\mathcal{C}$> où
W1 est l'ensemble des variables,
t0, ..., tn des termes de la P.L.,
$\mathcal{C}$ des contraintes sur {W1, $\mathcal{D}$}.

condition : $\exists$ Rj $\in$ P, Rj $\equiv$ (L :- L1, ..., Lm, Q_1, ..., Q_v.) où
L, L1, ..., Lm sont des prédicats de la P.L.,
Q_1, ..., Q_v des contraintes sur $\mathcal{D}$,
et où $\mathcal{C} \cup \{Q_1, ..., Q_v, L =_u t0\}$ est satisfait par la méthode de réduction choisie.

état final : G_{i+1} =<W2, :- L1, ..., Lm, t1, ..., tn., réduction($\mathcal{C} \cup \{Q_1, ..., Q_v, L =_u t0\}$)>

La méthode de réduction permet de déterminer si l'ensemble de contraintes est satisfait dans S et fournit un nouvel ensemble réduit de contraintes.

Soit W0, les variables de la requête R_0. Pour répondre à la requête R_0, la machine part de l'état initial <W0, R_0, ø>. Cette machine passe par tous les états qu'elle peut atteindre en répétant l'opération de base "->".

La notion de dérivation reste la même qu'en P.L.. Deux cas d'arrêt sont détectés :

- quand G_i -> <Wτ, ø, $\mathcal{C}$>, la dérivation courante est alors en **succès**,

- quand $\forall$ Rj $\in$ P, Rj $\equiv$ (L :- L1, ..., Lm, Q_1, ..., Q_v.),
et G_i = <Wi, :- t0, ..., tn., $\mathcal{C}$>
et $\mathcal{C} \cup \{Q_1, ..., Q_v, L =_u t0\}$ insatisfait par la méthode de réduction choisie,

la dérivation courante est en **échec**.

Soit une dérivation en succès, se terminant par <Wτ, ø, $\mathcal{C}$>. S est le produit cartésien des domaines des variables de Wτ. Nous appelons *solution* les points de S satisfaisant $\mathcal{C}$.

Les ***solutions*** du problème (P, R_0) sont toutes celles des dérivations en succès.

iii) Quelques langages

Les principaux langages de la P.L.C. sont les suivants :

- Prolog III qui utilise des équations linéaires sur un domaine rationnel et des équations booléennes [COL 86, 87],

- CLP(R) (Constraint Logic Programming) qui intégre des équations arithmétiques linéaires sur R [JAF 86, 87, 88],

- CHIP (Constraint Handling In Prolog) qui a une résolution sur un domaine de calcul entier fini, sur le domaine de l'algèbre de Bool et sur l'arithmétique rationnelle linéaire [DIN 87a, 87b, 88a, 88b, 88c, 88d], [VAN 89],

- CLEF v1 (Contraintes, Logique, Equations, Fonctions) qui possède une résolution de problème sur un domaine de calcul discret et fini de constantes [LEP 90].

B.3.2 Extension à la recherche de solutions approchées

Pour les problèmes d'emploi du temps et toute une classe de problèmes, il est utopique de vouloir obtenir des solutions exactes. C'est-à-dire des solutions satisfaisant toutes les contraintes. Dans la majorité des cas, la recherche de solutions est trop coûteuse ou les solutions sont inexistantes. La P.L.C. n'offre que des résolutions donnant des solutions exactes au problème posé. La résolution prouve qu'il n'existe pas de solution ou les donne toutes.

Des extensions à la P.L.C. sont apparues. Ils permettent de trouver de "bonnes" solutions au problème voulu. Leur méthode se fait par approche successives. Une solution est "bonne" quand elle satisfait un utilisateur.

La recherche de telles solutions passe par plusieurs techniques. La première est la modification de requêtes. La seconde compare des solutions d'un ensemble de requêtes possibles. Elle trouve donc la "meilleure" solution.

• La modification de requêtes consiste à changer un ensemble de réseaux. Nous ajoutons ou supprimons des contraintes.

• La comparaison des solutions porte sur celles obtenues dans des réseaux. Ces derniers sont créés par combinaisons possibles de poses de contraintes.

Nous qualifions ces deux techniques, de techniques de tâtonnement. En effet, il s'agit de chercher à tâton une solution satisfaisant un utilisateur. Dans la modification de requêtes, c'est l'utilisateur qui cherche. Dans le cas de comparaison de solutions, ce tâtonnement est automatique.

a) Modification de requête : tâtonnement par l'utilisateur

Cette technique s'apparente à une modification du problème à résoudre. Il se fait en cours de résolution.

Cela permet de modifier un ensemble de réseaux par la modification d'une des arêtes.

Poser une requête R_0 avec le programme P fournit un ensemble de solutions (éventuellement vide). Cet ensemble peut ne pas satisfaire l'utilisateur. Ainsi, des travaux sont apparus pour pouvoir adapter les solutions trouvées. Le principe consiste à ajouter ou supprimer des informations. Etape par étape, nous approchons de la solution désirée.

Ses travaux proposent de modifier en cours d'exécution les requêtes. La résolution n'est pas recommencée à partir de zéro.

La recherche de la "bonne" solution se fait pas à pas. Ce mécanisme est nommé "incrémental" par Van Hentenryck [VAN 90] et par Maher et al. [MAH 89].

L'intérêt pour notre problème d'emploi du temps est d'ajouter ou de supprimer des contraintes sur $\mathcal{D}$.

i) Modification de requête par réexécution

Une première proposition utilise d'une façon naïve et exclusive le "backtrack". Van Hentenryck [VAN 90], Maher et al. [MAH 89] étudient ce type de proposition. Ils ont conclu que cette méthode n'était pas assez efficace.

Van Hentenryck utilise exclusivement le backtrack pour changer de points dans l'espace de recherche. Donc chaque alternative est associée à un sous-espace de recherche. C'est le cas quand nous énumérons les valeurs d'une variable par backtrack.

Van Hentenryck propose alors une notion de "**réexécution**" pour ajouter ou supprimer une contrainte. Le principe consiste à mémoriser tous les chemins de la résolution menant à une solution. Nous mémorisons donc le choix des règles sélectionnées pour obtenir les solutions. Ces chemins sont nommés Oracle.

La recherche d'une solution se fait pas à pas. Le problème $P \cup R_0$ est noté P_0. Il constitue le premier pas. Puis nous additionnons (ou soustrayons) des contraintes. Nous passons alors au problème P_1. Le problème P_i correspond à $P \cup R_i$. R_i est égal à R_0 plus une union de contraintes {C}. Il s'associe à l'oracle O_i des chemins menant aux solutions.

L'ajout d'une contrainte C au problème P_i consiste à dérouler l'oracle O_i correspondant. Nous déroulons donc les chemins. Puis nous continuons la résolution normalement avec la contrainte C ajoutée. Ceci correspond au problème P_{i+1}. Cela revient à appliquer la contrainte C à l'espace de recherche réduit par P_i.

La suppression d'une contrainte C consiste à revenir au problème P_k. Nous reprenons donc l'espace de recherche réduit par P_k. P_k ne contient pas C et P_{k+1} contient C. L'oracle O_k est redéroulé. La résolution continue sans la contrainte C.

La réexécution est avantageuse par rapport à la méthode basée sur le backtrack. Son atout est de repartir d'un espace de recherche déjà connu. Le backtrack, par contre, recherche toutes les solutions. La résolution revient au lieu où la contrainte a été introduite. Ceci équivaut à repartir de zéro.

Cependant, la complexité de ce relâchement dépend du lieu de la pose de la contrainte C. Si C apparaît à la première règle, il faut refaire toute la résolution.

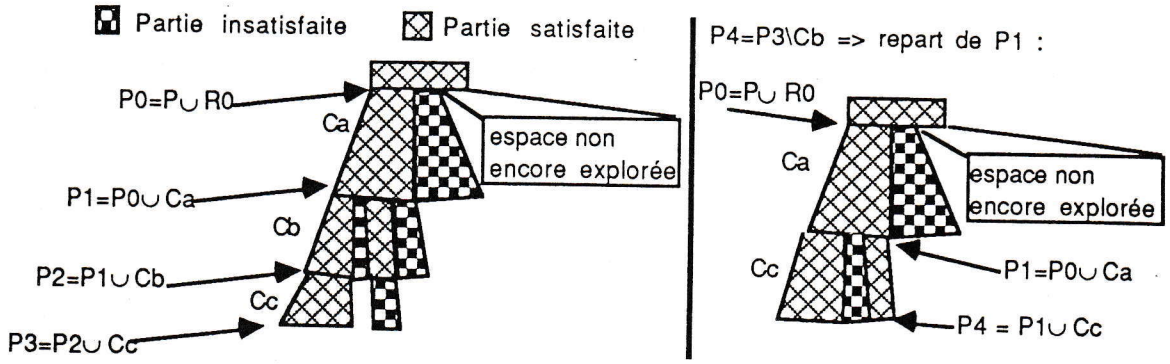


Figure 16. Exemple de réexécution

La figure 16 illustre cette réexécution. La partie gauche montre comment évolue l'espace de recherche au fur et à mesure des ajouts de contraintes. La partie droite exprime que le problème P4 est constitué du problème P3 moins la contrainte Cb. La résolution repart du problème P1. C'est-à-dire de l'espace de recherche réduit par Ca.

Van Hentenryck constate que les solutions du problème P_i sont inclues dans celles de $P_i \setminus C$. Il introduit alors une heuristique. Celle-ci consiste à continuer la résolution comme si la contrainte était relâchée. Mais nous gardons l'alternative pour revenir au problème P_k. Celui-ci est la dernière règle ne contenant pas C.

Ceci améliore la résolution mais ne garantit rien dans la complexité. Si le problème P_i n'a pas de "bonne" solution, il faut revenir à P_k.

ii) Modification de requête par ajout intelligent

Maher et al. [MAH 89] proposent une autre direction. Elle est basée sur le backtrack et une exécution intelligente de la nouvelle requête. Ils proposent à l'utilisateur la seule fonctionnalité d'ajout de prédicats dans une requête en cours d'exécution. L'utilisateur peut ainsi approcher sa "bonne" solution par tâtonnement.

Soit R_0 équivalent à ":- P, Q.". Nous désirons ajouter le prédicat I entre P et Q. La nouvelle requête équivaut à ":- P, I, Q.".

Le principe permet de changer sa requête en cours de résolution. Pour cela, nous distinguons les dérivations en succès et celles qui restent à explorer :

- pour celles en succès, l'incrément I est ajouté à la fin de leur résolution. Nous avons l'équivalent de la requête ":- P, Q, I.",

- pour celles à chercher, le backtrack a lieu jusqu'au point où l'incrément est à ajouter. Puis la résolution se poursuit avec l'incrément. C'est comme si nous avons la requête ":- P, I, Q.".

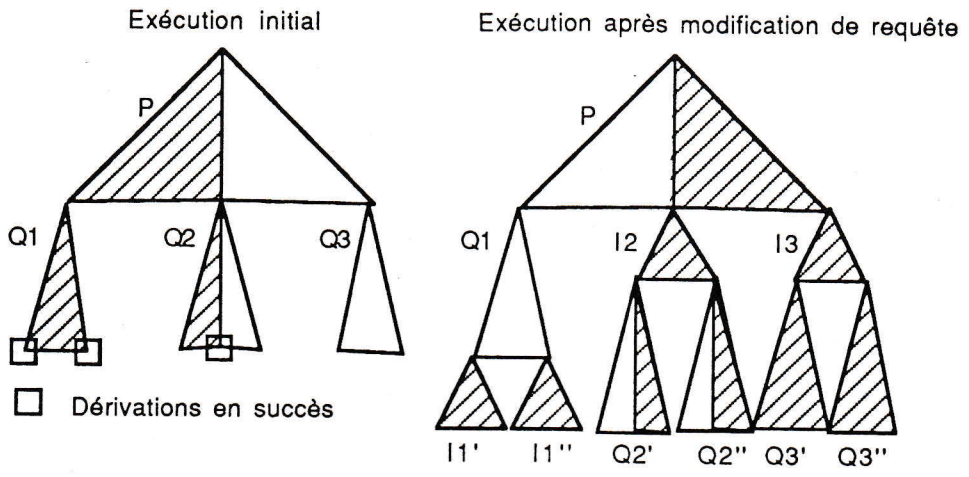


Figure 17. Ajout d'un incrément I

La figure 17 représente cette modification avant et après l'ajout de I. Les parties hachurées représentent les parties parcourues par la résolution.

Q1, Q2, Q3 sont les parties élaguées et dérivées de Q après application de P. Similairement, I1', I1'', I2 et I3 découlent de I. Q2', ..., Q3'' découlent de Q2 et Q3.

Cette méthode n'est pas plus efficace que celle de "réexécution". Si le problème est sans solution, l'utilisateur peut "s'épuiser" pour approcher sa "bonne" solution. La fonction de suppression est traitée d'une façon naïve par backtrack.

b) Meilleures solutions : tâtonnement automatique

Cette technique s'apparente à rechercher parmi un ensemble de réseaux celui qui donne la(es) solution(s) convenant le plus à un utilisateur. Pour faire cette comparaison un ordre est donné sur les sous-réseaux.

Le but revient à "surdimensionner" le problème initial. Nous ajoutons des contraintes dites "secondaires". La similitude avec les algorithmes d'optimisation de Recherche Opérationnelle est très forte. Le principe consiste à ajouter des coûts aux contraintes et une fonction "objectif" à optimiser.

Pour faire ce rapprochement, Maher et al [MAH 89], [BOR 89] proposent diverses notions :

- une correspondance entre les requêtes et un système de notation. La fonction de correspondance est appelée mesure.

- une possibilité de comparer ces requêtes à travers leur note et

- une hiérarchie de contraintes. Nous comparons les solutions à travers les contraintes mises en jeu dans les requêtes.

i) Mesures et comparateurs

Une **mesure** $m(G_i)$ fait correspondre une note v à une requête G_i.

Une **mesure élagante** est une mesure tel que :

$$\underline{si}\ G_i \rightarrow G_{i+1}\ \underline{alors}\ m(G_{i+1}) \leq m(G_i).$$

Supposons que nous désirons obtenir une meilleure solution à une requête Q. Nous avons une mesure m. Admettons que nous possèdons déjà une solution. Elle a été obtenue par une requête de note M. La recherche de meilleures solutions passe par toute requête G_j où $m(G_j) \leq M$.

Par la suite, la note des solutions d'une requête correspond à la note de la requête.

Une mesure est **monotone** si $\forall\ G_i$, $G_i \rightarrow \ldots \rightarrow G_{i+n}$ alors $m(G_{i+n}) \leq m(G_i)$. Il est clair qu'une mesure élagante est monotone.

Une mesure est **basée sur les contraintes** si nous ne considèrons que les termes d'interprétation sur le domaine $\mathcal{D}$ ajouté. Une mesure monotone basée sur les contraintes est élagante.

Maher et al. [MAH 89], [BOR 89] appliquent ces notions sur un langage comme CLP(R). La méthode de satisfaction se base sur une variante de l'algorithme du Simplex. Ils précisent que la mesure fournit la note optimale d'une fonction. Cette dernière est similaire à une fonction "objectif". Elle est sujette à un ensemble particulier de contraintes.

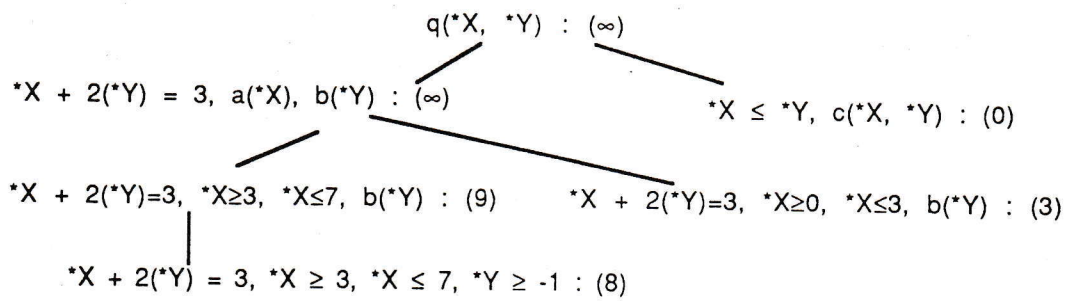
Exemple : Reprenons l'exemple de Maher [MAH 89].

Soit le programme CLP($\mathcal{R}$) suivant :

```
q(*X, *Y) :- *X + 2(*Y) = 3, a(*X), b(*Y).
q(*X, *Y) :- *X <= *Y, c(*X, *Y).
a(*X) :- *X >= 3, *X <= 7.
a(*X) :- *X >= 0, *X <= 3.
b(*Y) :- *Y >= -1.
c(*X, *Y) :- 3(*X) - *Y = 7, *Y >= 0.
```

avec la requête q(*X, *Y) et la mesure élagante $m_{CLP} = \max(*X - *Y)$.

La figure 18 présente cette résolution. Il n'est pas nécessaire de développer les branches les plus à droites. Car suivant m_{CLP}, elles ne sont pas meilleures que celles déjà développées.



Traversée de l'arbre de recherche pour la requête q(*X, *Y)

Figure 18. Mesure élagante

Nous nous rapprochons donc des résolutions par optimisation.

ii) Hiérarchie de contraintes

Cette notion de hiérarchie étend celle de contraintes. Ainsi les préférences entre contraintes expriment des préférences entre solutions.

Hiérarchisons les contraintes facultatives. Nous créons un ensemble de réseaux à satisfaire par la combinaison des contraintes. Chaque combinaison se différencie par la présence d'une contrainte falcultative. L'ensemble des réseaux créés est ordonné en respect de la hiérarchie et d'une mesure introduite.

Une **hiérarchie de contraintes** est un multi-ensemble de contraintes étiquetées d'un **poids**. Ces poids forment un ordre total. Ils correspondent à un entier entre 0 et N.

Soit $\mathcal{C}$ un ensemble de contraintes, soit $\mathbf{C_i} \in \mathcal{C}$, alors C_i est l'ensemble des contraintes de poids i.

Soit $\mathcal{C}$ un ensemble de contraintes mises en jeu, soit $C_i \in \mathcal{C}$, $C_j \in \mathcal{C}$ et $i < j$, alors C_i caractérise un ensemble de contraintes plus préférables à celles de C_j.

C_0 représente les contraintes qui doivent être respectées. Par la suite, le mot **absolu** est associé aux contraintes de C_0. Le mot **secondaire** est réservé à celles de $(C_1 \cup ... \cup C_N)$.

Une **pré-solution** ou **solution réalisable** θ d'une hiérarchie de contraintes HI est une valuation des variables présentes dans HI. Nous devons avoir : $\forall c \in C_0, C_0 \in HI, c$ est satisfait en θ. Si S est l'espace de recherche, nous notons S^{C_0} pour l'ensemble des solutions réalisables.

iii) Mesure associée aux hiérarchies

Une **pré-mesure** g est une fonction de correspondance. Les paramètres sont une solution réalisable et un ensemble de contraintes. La valeur rendue est un élément d'un multi-ensemble MV de notes :

$$\theta \in S^{C_0}, g(\theta , C) = mv \in MV.$$

Soit α et β, deux solutions réalisables, soit g_1, ..., g_N les pré-mesures de multi-ensembles MV_1, ..., MV_N alors α est **meilleur** que β si :

$$<g_1(\alpha, C_1), ..., g_N(\alpha, C_N)> \geq <g_1(\beta, C_1), ..., g_N(\beta, C_N)>$$

où pour $\geq$, nous prenons l'ordre lexico-graphique de $MV_1 \times ... \times MV_N$.

Le but est de chercher les solutions réalisables maximales sous cet ordre.

Soit MV le multi-ensemble de notes. C'est un ensemble ordonné par inclusion sur tous les ensembles de contraintes secondaires. Soit $g_i(\theta, C_i) = \{c \in C_i, c \text{ est satisfait en } \theta\}$. Nous obtenons la notion de **comparateur "locally predicate better"** définie dans [BOR 89].

Soit MV l'ensemble R des réels. Avec un choix approprié de pré-mesures, nous arrivons aux **comparateurs "globally-better"** définis dans [BOR 88]. Par exemple, une pré-mesure est le nombre de contraintes secondaires satisfaites. Nous pouvons avoir le nombre de contraintes secondaires insatisfaites.

Cette définition de comparateur peut être étendue pour comparer deux solutions de deux hiérarchies différentes, HI et HI' :

Soit α solution réalisable de HI et β solution réalisable de HI', alors α est meilleure que β si :

$$C_i \in HI, C'_i \in HI', <g_1(\alpha, C_1), ..., g_N(\alpha, C_N)> \geq <g_1(\beta, C'_1), ..., g_N(\beta, C'_N)>.$$

En réalité, ces mesures basées sur les contraintes sont à rapprocher de la notion de critères [ROY 88] ou de celle de fonction "objectif".

iv) Une implantation : HCLP(R)

Borning et al. [BOR 89] intègrent leur notion de hiérarchie de contraintes dans le langage CLP(R). Ceci forme le système HCLP(R). Ils intégrent la mesure "locally predicate better" entre solutions de même hiérarchie.

N.B. : *Syntaxiquement, un prédicat HCLP(R) définit des symboles. Ils correspondent aux différents niveaux de la hiérarchie. Chaque contrainte secondaire est précédée par le symbole définissant son niveaux. Chaque contrainte absolue garde sa syntaxe originelle. Aucun symbole ne précède sa syntaxe.*

L'algorithme intégré dans HCLP(R) passe par deux phases.

Une première phase cherche les solutions réalisables. En même temps, les contraintes secondaires sont stockées et triées.

Une deuxième phase s'applique sur les solutions réalisables trouvées. Les contraintes secondaires stockées sont posées par ordre décroissant.

Le comparateur introduit ne s'applique qu'à une même dérivation. Si des alternatives existent entre contraintes, la résolution de la P.L.C. backtracke. Les nouvelles solutions trouvées ne sont pas comparées aux anciennes. Donc les solutions

d'une dérivation ne sont pas comparées aux autres dérivations.

Il faut introduire soi-même un comparateur global au niveau du programme. Il compare les meilleures solutions de chaque dérivation.

c) Critiques

La première voie d'étude est de construire pas à pas une requête. Celle-ci correspond à la "bonne" solution. Elle implique une grande disponibilité du concepteur de l'emploi du temps. Il faut qu'il sache comment il obtient sa "bonne" solution. Or ce n'est pas toujours le cas. L'intérêt de l'informatisation du problème d'emploi du temps est d'aider le concepteur dans sa recherche de la bonne solution.

De plus, procéder par tâtonnement n'est pas optimum. Cela peut demander une longue recherche. C'est le cas dans le problème d'emploi du temps. En effet, il existe des cas rapides d'exécution. Souvent ce sont des cas où il y a trop ou trop peu de contraintes.

Mais il existe, en général, des cas où le temps d'exécution est long. Si le temps de résolution paraît infini, l'utilisateur ne va pas chercher à enlever des contraintes ou à en rajouter. De plus, il ne peut pas connaître la raison de cette trop longue durée du temps d'exécution.

Une meilleure approche est celle de la recherche de "meilleures solutions" par des mesures. La possibilité de hiérarchiser les contraintes est une aide à l'utilisateur. Il peut alors "surdimensionner" son problème d'emploi du temps.

Les idées et concepts des algorithmes d'optimisation vus précédemment pourraient être repris (Relaxation Lagrangienne, méthode du sous-gradient, ...).

Allier la méthode de meilleures solutions à celle de la modification de requêtes peut apporter un bénéfice à la résolution.

Cependant, plusieurs inconvénients s'en dégagent.

Le premier inconvénient est qu'il faut intégrer les mesures. Celles-ci doivent être explicites. Bien souvent l'utilisateur ne connaît pas tous ses critères et agit par expérience.

Le deuxième inconvénient dans HCLP(R) provient de l'intégration figée de la mesure. Supposons que les contraintes changent. C'est-à-dire qu'il y a un ajout ou une suppression dans la hiérarchie. Il faut tout refaire. La résolution recommence de zéro. Donc les inconvénients des méthodes d'optimisation restent. Nous perdons alors le côté dynamique et évolutif des données de la P.L..

De plus, il faut rechercher les meilleures solutions satisfaisant toutes les combinaisons de contraintes possibles. Cette méthode augmente la complexité de la résolution.

Aucune des deux méthodes ne peut indiquer à un utilisateur la raison des échecs des dérivations. Il ne connait pas la cause d'élimination de certains points de l'espace de recherche. L'utilisateur ne sait pas pourquoi une solution est meilleure qu'une autre. En lui indiquant ces raisons et causes, cela peut l'éclairer dans sa résolution.

En conclusion, il ne faut pas uniquement disposer d'un mécanisme de relâchement ou d'évaluation des solutions suivant un critère figé. Ces mécanismes semblent intéressants. L'idée est de "surdimensionner" le problème initial. Cependant, il faut pouvoir utiliser un jeu de mesures. L'expérience de l'utilisateur est une mesure particulière.

Ce "surdimensionnement" risque de provoquer un arrêt de la résolution. Cet arrêt s'exprime par des dérivations en échec. Il faut donc avoir un outil analysant l'arrêt. Certaines dérivations seront alors reprises et redémarrées.

Ce mécanisme doit diagnostiquer les réductions de l'espace de recherche. Il doit expliquer la cause des échecs des dérivations. Il faut alors avoir une résolution souple. Elle doit s'adapter aux évolutions des données.

Dans la seconde partie de cette thèse, je présente une méthode et une technique pour diagnostiquer les contextes liés aux étapes de dérivations. Nous indiquons alors, les contraintes réduisant effectivement l'espace de recherche. En cas d'échec(s) de dérivation, nous indiquons les contraintes à relâcher dans un sens nécessaire et suffisant. Les notions de "meilleures solutions", de hiérarchie de contraintes pourront être reprises pour aider à trouver le "meilleur relâchement".

B.4 Une intégration aux domaines discrets finis : CLEF v1

Nous présentons une intégration particulière d'un langage de P.L.C., CLEF v1. Ce langage a été développé au CNET à Lannion. Il intégre des notions de domaine fini. Il permet une programmation et une résolution intéressantes des problèmes d'emploi du temps.

Le concept de domaine fini a fait l'objet d'étude approfondie pour le langage CHIP [VAN 89]. CLEF v1 est un langage qui intègre cette notion de domaine discret et fini.

B.4.1 Lien avec le problème de satisfaction de contraintes

a) Problème de satisfaction de contraintes et réseaux de contraintes

CLEF v1 est un langage de la P.L.C.. Les règles ont la même expression qu'en P.L.C..

Un ***problème de satisfaction de contraintes*** est composé d'un programme constitué de règles de CLEF v1 et d'une requête.

La résolution CLEF v1 permet de construire des ***réseaux de contraintes***. Ces réseaux ont la forme R ($V_1 \cup ... \cup V_n$, $\mathcal{C}$). $\mathcal{C}$ est l'ensemble des contraintes présentes dans le réseau R. C'est-à-dire des contraintes d'unification ou sur $\mathcal{D}$. V_i est l'ensemble des instanciations de la variable $*v_i$ à chaque valeur de son domaine.

b) Domaine, variables sous domaine et Espace de recherche

Un domaine discret fini est un ensemble fini de constantes. Soit un sous-ensemble discret fini D_i du domaine de calcul $\mathcal{D}$. Un tel ensemble D_i est attribué à chaque variable $*v_i$. Il désigne le ***domaine*** de $*v_i$.

N.B. : *En CLEF v1, un prédicat associe en interne un domaine D_i à une variable $*v_i$. Sa syntaxe est "domaine ($*v_i$, D_i)".*

Si une variable *v1 est unifiée à une autre *v2 qui possède un domaine D2, alors nous avons deux cas. Si *v1n'a pas de domaine associé, cette variable hérite du domaine de *v2. Si *v1 a déjà un domaine D1, alors nous attribuons aux deux variables *v1 et *v2, l'intersection des deux domaines D1 et D2.

Si une variable n'a pas de domaine, elle appartient alors exclusivement à la P.L.. Son domaine est l'univers d'Herbrand.

L'***espace de recherche*** S d'un problème est le produit cartésien des domaines de chaque variable apparaissant dans le problème de la P.L.C..

c) Contraintes, satisfaction d'une contrainte et domaine de calcul

Le ***domaine de calcul*** $\mathcal{D}$ ajouté à CLEF v1 se constitue de l'ensemble discret fini.

Il existe dans CLEF v1 des symboles relationnels pré-définis. Ils portent sur l'univers de calcul introduit $\mathcal{D}$. Ces symboles sont des ***contraintes***.

Des schémas de déclenchement sont associés à chaque contrainte. Ils stipulent quand et comment a lieu le déclenchement de chaque contrainte. Une contrainte est "gelée" tant que tous ses schémas ne concordent pas avec le contexte courant de la résolution.

Le premier schéma est appelé de "**réveil**". Si les variables de la contrainte sont en concordance avec ce schéma, il permet de réveiller la contrainte. Ce schéma indique l'état des variables pour que le contrainte soit réveillée. Un premier état est que la variable est sous domaine. Un deuxième état est qu'elle est instanciée.

Un deuxième schéma est nommé d'"**activation**". Il permet d'activer la méthode locale de saturation. Quand la contrainte ne relie plus que k variables sous domaine, la méthode de saturation est activée.

Un troisième schéma interne est appelé de "**satisfaction**". Il indique quand la contrainte est satisfaite.

La contrainte est surveillée par rapport à ces schémas. Le schéma de saturation est appliqué à nouveau tant que la satisfaction de la contrainte n'est pas connue.

L'***insatisfaction*** d'une contrainte provoque l'échec de la résolution. La ***satisfaction*** implique la disparition de la contrainte du réseau et donc de la résolution.

Ainsi, une contrainte C est ***insatisfaite*** quand sa valeur d'interprétation est "faux". C'est le cas quand une des variables que relie la contrainte a son domaine vide.

Dans le cas contraire, la contrainte est ***satisfaite*** ou en attente de satisfaction.

Supposons que l'état de satisfaction d'une contrainte est indéterminé. C'est-à-dire ni satisfait, ni insatisfait. Alors la contrainte reste réveillée dans la résolution. Elle reste donc dans le réseau.

Le programmeur peut définir ces propres contraintes. Il dispose d'un prédicat

déclaratif de syntaxe :

"d-contrainte (schéma_d'activation, [nom_contrainte | schéma_de_réveil])"

Cette déclaration s'associe à une application "nom_contrainte". Cette dernière est définie avec son algorithme local de saturation et son schéma de satisfaction.

Le schéma de satisfaction d'une contrainte s'apparente à un test de cohérence.

Les contraintes pré-définis de CLEF v1 sont présentées ultérieurement.

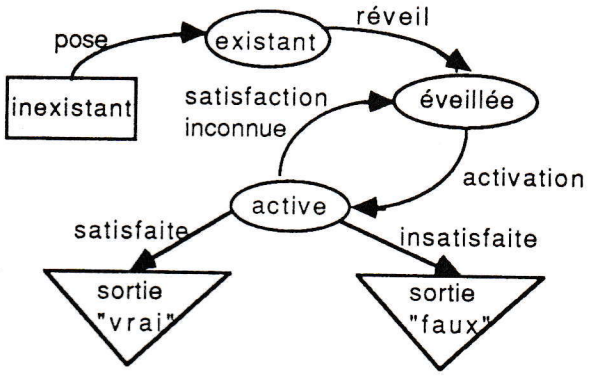


Figure 19. Etats d'une contrainte

d) Méthode de satisfaction, Ensemble de rejet, Ensemble d'agrément

La ***méthode de satisfaction*** est une **méthode de saturation partielle**. Un schéma local de saturation est associé à chaque contrainte. Il élimine les valeurs incohérentes des domaines des variables en relation. Cependant, toutes les incohérences ne sont pas éliminées.

Supposons qu'une contrainte C relie k variables $*v_i$. L'espace de recherche S est de la forme $D_1 \times ... \times D_k$ avant le déclenchement de la saturation. L'espace résultat d'une saturation locale est décrit par un produit cartésien.

La méthode m est toujours la même. Par simplification de notation, l'indice m est éliminé des notations futures. Ainsi nous notons S^C, les points de l'espace de recherche S satisfaisant une contrainte C. $S^{C'}$ représente les points rejetés par C.

CLEF v1 sait seulement calculer l'espace de rejet. Nous associerons le terme espace d'agrément à un espace candidat d'agrément.

L'***ensemble d'agrément*** S^C de C sur S, après saturation a la forme :

$$S^C = D_1{}^C \times ... \times D_i{}^C \times ... \times D_n{}^C \text{ tel que}$$

si C relie (*v1, ..., *vk) : $\forall$ *vi, i $\in$ [1..k], $\forall$ ai $\in D_i{}^C$, $\exists$ s $\in S^C$, s formé avec ai et C satisfait en s.

L'***ensemble de rejet*** $S^{C'}$ de C sur S, après saturation a la forme :

$\forall s \in S^{C'}$, C non satisfait en s.

La méthode de satisfaction est une réduction de l'espace de recherche. Nous prenons donc toujours des contraintes C tel que $D_i^C \subseteq D_i$.

e) Solution, Satisfaction de réseaux et méthodes associées

La résolution du programme P et de la requête R_0 construisent des réseaux. ***La satisfaction des réseaux*** se fait par la résolution de la P.L.C. et la méthode de saturation choisie à travers ses dérivations. La satisfaction de chaque réseau est l'union des satisfactions locales. Elle est similaire à la technique de "forward checking".

Les solutions sont constituées par l'union des solutions de chaque dérivation. Leur définition est similaire à celle faite pour la P.L.C..

Le but de CLEF v1 est de proposer une recherche intelligente des solutions. Ainsi, la notion d'"**étiqueteur**" est introduite. Elle permet d'énumérer les points de l'espace de recherche. Cette énumération se fait à travers les valeurs des variables sous domaine.

Le principe de l'énumération rassemble les variables qui nous intéressent. Puis chaque variable est instanciée à chaque valeur cohérente de son domaine.

Ces étiqueteurs proposent différentes façons de choisir les variables. Les trois principaux prédicats permettent différentes énumérations :

i) **d-valuer**

Sa syntaxe est : d-valuer (*terme*)

Le principe interne est le suivant :

• Si *terme* est une variable $*v_i$ de domaine D_i alors $*v_i =_u d_k$ tel que $d_k \in D_i$ et d_k cohérent.

Ce prédicat est associé au backtrack. Il balaye toutes les valeurs cohérentes du domaine de la variable $*v_i$. Chaque valeur est instanciée à la variable.

• Si *terme* est une constante alors succès inconditionnel,
• sinon échec.

ii) **pluscontraint**

Sa syntaxe : pluscontraint (*var terme liste*).

La variable la plus contrainte de *terme* est attribuée à *var*. La liste restante est associée au terme *liste*.

• Soit *terme* = $<*v_1, ..., *v_k>$. Soit Nb_i le nombre de contraintes associées à $*v_i$ et soit la fonction position avec position ($*v_i$, *terme*) = (k - i). $*v_i$ est plus contraint que $*v_j$ ($i \neq j$) si :

- $(1/|D_i|, Nb_i, \text{position } (*v_i, terme)) > (1/|D_j|, Nb_j, \text{position } (*v_j, terme))$.
  où > est l'ordre lexico-graphique sur $Q \times N^2$.

N.B. : *Ce mécanisme ou stratégie est à associer à l'heuristique du "plus grand degré avec saturation d'abord" (c.f. A.3.4 d).*

iii) **étiqueter-pluscontraint**

Sa syntaxe : étiqueter-pluscontraint (*l)

Le principe de ce prédicat est défini par la stratégie suivante :

- étiqueter-pluscontraint (*l) :-
  pluscontraint (*var, *l, *l_restante),
  d_valuer(*var),
  étiqueter-pluscontraint (*l_restante).

- étiqueter-pluscontraint (*l).

B.4.2 Contraintes pré-définies

La syntaxe des contraintes pré-définies est préfixée d'un accent circonflexe. La notation "d" indique qu'une variable est sous domaine. La notation "f" indique que la variable est instanciée.

Soit la contrainte C(a1, a2, a3). Associons le schéma de réveil (d f d). Au réveil de C, a1 et a3 doivent être sous domaine et a2 instanciée.

Rappel : D_1 est le domaine de la variable *v1. $D_1{}^C$ est le domaine réduit par C.

a) ^diff

Syntaxe : ^diff (*a1*, *a2*).

Schéma de réveil : (d d)

Schéma d'activation : au plus une variable sous domaine

Schéma local de saturation :
Trois cas se produisent :

1) Si *a1* est une variable sous domaine et *a2* une constante alors :

$$Da1^{\wedge diff} = Da1 \setminus \{a2\}$$

Si $Da1^{\wedge diff} \neq \emptyset$ alors ^diff satisfaite sinon ^diff insatisfaite.

2) Le cas est symétrique si *a1* est une constante et *a2* une variable sous domaine.

3) Si *a1* et *a2* sont des constantes, nous avons :
si $a1 \neq a2$ alors ^diff est satisfaite
sinon ^diff est insatisfaite.

Si l'un des domaines se réduit à un singleton, la contrainte d'unification est générée. L'unification se fait entre la variable concernée et l'unique valeur de son domaine.

b) ^element

Syntaxe : ^element (*n*, *l*, *v*).

Schéma de réveil : (d f d)

Schéma d'activation : au plus deux variables sous domaine

Cette contrainte exprime que *v* est le $n^{ième}$ élément de *l*.

Schéma local de saturation :

La saturation purge les domaines D_v et D_n tel que :

$$Dn^{\text{^element}} = \{\lambda / \lambda \in Dn, \exists \mu \in Dv^{\text{^element}}, \mu\ \lambda^{ième} \text{ élément de } l\} \text{ et}$$

$$Dv^{\text{^element}} = \{\mu / \mu \in Dv, \exists \lambda \in Dn^{\text{^element}}, \mu\ \lambda^{ième} \text{ élément de } l\}.$$

La purge des domaines ne garantit pas la satisfaction de la contrainte. La satisfaction peut passer par plusieurs purges faites au fur et à mesure de la résolution.

c) ^au-plus-n=k

Syntaxe : ^au-plus-n=k (*n*, *l*, *k*).

Schéma de réveil : (f d f)

Au réveil, *n* et *k* sont des constantes,
l est une liste de variables sous domaine.

Schéma d'activation : quelque soit le nombre de variables sous domaine

Schéma local de saturation :

Soit VIK = $\{*v_i / *v_i \in l$ et $*v_i$ instanciée à la constante $k\}$,

- si |VIK| = *n* alors ^au-plus-n=k satisfaite et

$\forall\ *v_j \notin$ VIK, $*v_j$ sous domaine $\in l$: $D_{*vi}^{\text{^au-plus-n=k}} = D_{*vi} \setminus \{k\}$

- si |VIK| > *n* alors ^au-plus-n=k insatisfaite

Si |VIK| < *n*, la contrainte reste active.

d) ^sigma-idemx

Syntaxe : ^sigma-idemx (*lcoef*, *l*, *seuil*).

Schéma de réveil : (f d f)

Au réveil, *lcoef* est une liste de coefficients numériques c_i
seuil est un nombre et
l est une liste de variables sous domaine $*v_i$, $i \in [1 .. n]$.
$*v_i$ a le même rang dans *l* que son coefficent c_i dans *lcoef*.

Schéma d'activation : quelque soit le nombre de variables sous domaine

Schéma local de saturation :

Le principe est le même que celui de la contrainte précédente. Le test à effectuer est :

$$\sum c_i \leq seuil$$
$$c_i \in lcoef$$
$$*v_i \in l \text{ et instanciée à } k$$

e) Contraintes numériques

Ces contraintes portent sur des nombres entiers. Elles ne prennent en paramètre que des expressions numériques.

Une expression numérique est composée d'un monôme ou d'une somme de monômes. Un monôme est une constante, une variable ou une structure. La structure a la forme (* *constante variable*).

Les constantes sont des entiers. Les variables ont des domaines de ces constantes.

Syntaxes : - ^= (*exp1, exp2*),
- ^< (*exp1, exp2*),
- ^> (*exp1, exp2*),
- ^<= (*exp1, exp2*),
- ^>= (*exp1, exp2*),
- ^# (*exp1, exp2*).

Schéma de réveil : (d d)

Schéma d'activation : quelque soit le nombre de variables sous domaine sauf pour ^#. Ce dernier doit avoir au réveil, au plus une variable sous domaine.

Leur schéma local de saturation teste les égalités ou inégalités correspondantes. Ces contraintes éliminent les valeurs incohérentes dans le domaine des variables concernées.

La contrainte ^# a le même comportement que ^diff.

f) Une mesure élagante : minimiser-maximum

Syntaxe : minimiser-maximum (*pb, terme, borne-inf*)

où *pb* est un problème CLEF v1 à résoudre,
terme est un terme CLEF v1 et

borne-inf, facultatif, réfère une valeur numérique.

Ce prédicat cherche les solutions de coût minimal pour *pb*. Les variables dans *terme* sont assimilées à des variables de coût. Le coût d'une solution est le maximum de ces variables.

Si *borne-inf* est spécifié, ceci permet de partir des solutions de coût inférieur ou égal à *borne-inf*.

Le prédicat "minimiser-maximum" met en place une méta-contrainte. Elle est activée dès que toutes les variables de *terme* sont sous domaine. Elle énumére intelligemment tous les points du sous-espace induit par ces variables. Pour ceci, elle intègre des techniques de Branch & Bound.

Ce prédicat "minimiser-maximum" peut être associée à une mesure élagante. Cette dernière notion est définie dans la division B.3.2.

B.5 Conclusion

Dans ce chapitre, nous nous sommes aperçu que la P.L.C. est issu des mondes de la Programmation en Logique et de la Programmation par Contraintes.

La P.L.C. hérite des résolveurs numériques utiles et efficaces de la Programmation par Contraintes. Les contraintes peuvent exprimer des relations à maintenir entre variables. Les variables et les contraintes forment alors un réseau de contraintes à satisfaire dans un espace de recherche. Les problèmes sont donc facilement et naturellement exprimés.

La P.L.C. hérite de Prolog la capacité à pouvoir gérer l'indéterminisme et donc la combinatoire. Elle hérite aussi de la nature déclaratif de Prolog.

Ainsi nous avons réunis tous les "ingrédients" pour résoudre au mieux les problèmes d'Emploi du Temps.

De plus, le langage particulier, CLEF v1, se restreint à un espace de recherche discret et fini. Ce langage possède donc de sérieux atouts pour bien résoudre les problèmes d'Emploi du Temps.

CONCLUSION

Dans le chapitre A, nous avons rencontré une première méthode de résolution se basant sur la coloration dans les graphes. Celle-ci est efficace si la matrice d'incidence du graphe est totalement unimodulaire. Cependant, dans le cas général, le problème de la recherche du nombre minimum de couleurs est NP-complet.

Une deuxième méthode de résolution a été introduite et est celle de flot et de couplage. Il s'agit de rechercher plusieurs flots sur un réseau de transport. Tous les types de contraintes souhaitées ne sont pas toujours possibles à intégrer. De plus, quand l'intégration est faite, celle-ci est figée. La modification de contraintes revient alors à refaire une analyse du problème.

Une troisième méthode de résolution abordée est la programmation par variables entières. L'inconvénient majeur est celui du volume des données. Une autre limitation est

qu'il faut être expert pour trouver les équations adéquates. Cette méthode ne permet pas de prendre en compte facilement l'évolution des données.

Ensuite, nous avons présenté des techniques cherchant des solutions moins exactes. Ils permettent une certaine souplesse dans la description des données.

Cependant, des inconvénients persistent. Le nombre de composants doit être limité. Les matrices d'incidence doivent être totalement unimodulaires. La durée des cours doit être unitaire. Il existe plusieurs problèmes : de stockage, de gestion des données, de formulation précise, de prise en compte de l'aspect évolutif des données, ...

En bref, les "points noirs" du problème d'emploi du temps sont :

- Des données trop diversifiées (en particulier les contraintes),

- La nature combinatoire du problème,

- L'aspect grande dimension et fortement contraint du problème,

- L'utilisation de critères flous, mal définis,

- Les changements intempestifs.

Les résolutions particulières pour chacun de ces points noirs doivent être synthétisées dans un système "programmable" et déclaratif. Ce système doit permettre d'exprimer toutes sortes de propriétés d'emploi du temps et différentes manières de résoudre les problèmes d'emploi du temps. C'est donc un langage.

Ainsi le chapitre B présente le paradigme et les langages de la Programmation par Contraintes.

Nous rendons compte que Prolog est un langage de Programmation par Contraintes. Son univers de calcul est celui d'Herbrand. Il permet une prise en compte dynamique des données. Il est par nature déclaratif. Cependant, la résolution est restreinte au monde symbolique. Ceci ne permet pas une résolution efficace pour les problèmes d'emploi du temps.

D'autres langages ont étendu leur univers de calcul au monde numérique. Ainsi les langages de Programmation en Logique avec Contraintes possédent l'univers d'Herbrand et un autre univers numérique. Ces langages intègrent un système aisé et souple de manipulation de contraintes symboliques comme numériques. Ils permettent de gèrer les combinaisons de données associées. L'aide à la décision est facile. Les données sont prises en compte dynamiquement. Ces langages sont par nature déclaratifs.

Plus particulièrement un langage, CLEF v1, a été présenté pour bien montrer que ses caractéristiques peuvent nous aider à mieux résoudre les problèmes d'emploi du temps.

Partie II

Partie II

Introduction

Cette partie présente exclusivement mes travaux.

Le premier travail effectué constitue le premier chapitre de cette deuxième partie. Nous y expliquons une manière de résoudre les problèmes d'emploi du temps par la P.L.C..

Il s'agit d'utiliser une coloration multi-dimensionnelle. Celle-ci est définie dans le premier chapitre. Les limites et les avantages sont alors dégagés.

Nous pouvons comparer le problème d'emploi du temps en différents termes.

En terme de problème d'emploi du temps, cela correspond à chercher une fonction g telle que g : $\mathcal{F} \rightarrow$ I, respecte des propriétés *P*.

En terme de satisfaction de contraintes, il s'agit de la satisfaction du réseau R($\mathcal{F}$,*P*) sur S= I^n pour n compositions.

En terme de coloration d'hyperarêtes, il faut colorer les hyperarêtes $\mathcal{F}$ de H[E, $\mathcal{F}$]. Les propriétés *P* doivent être respectées. La palette I utilisée est un ensemble de couleurs multi-dimensionnelles. Un ensemble d'affectations d'un type particulier correspond à chaque dimension.

Le premier chapitre établit la corrélation entre coloration multi-dimensionnelle, satisfaction de réseau par la P.L.C. et résolution de problème d'emploi du temps.

Dans la première partie de ce rapport, nous avons passé en revue les principales résolutions existantes sur un même exemple. Ici, nous reprenons cet exemple que nous étendons. Celui-ci illustre l'application proposée. Cet exemple est plus général que ceux pris pour les autres résolutions présentées au chapitre A. Ceci nous aide à comparer les

inconvénients et les avantages de cette application par rapport aux autres résolutions.

Nous en déduirons que l'application avec la P.L.C. apporte beaucoup. Mais la résolution par la P.L.C. possède aussi ses inconvénients.

Un premier inconvénient intervient dans le cas où la résolution ne trouve aucune solution. Une deuxième limitation est quand ces solutions ne sont pas obtenues dans un temps raisonnable pour l'utilisateur. Dans ces cas, ce dernier n'a aucune solution même incomplète. Une solution incomplète signifie des placements qui ne satisfont pas toutes les contraintes en place.

Le deuxième chapitre de cette partie présente ma proposition pour résoudre le premier inconvénient. Celui-ci se caractérise par une trop grande restriction de l'espace de recherche par les contraintes. Aucune solution n'est obtenue, même incomplète.

Après une critique de l'existant sur les résolutions approchées, nous en concluons qu'elles ne sont pas utilisables pour une approche d'implantation en P.L.C..

En effet, les langages de P.L.C. ne permettent pas de déterminer aisément les contraintes à relâcher lors d'un échec.

Une extension à la P.L.C. est proposée. Celle-ci permet de trouver des ensembles de contraintes nécessaires à relâcher. De plus, le relâchement est suffisant pour dégager au moins un point de l'espace de recherche.

Différents types possibles de relâchement sont proposés. Puis, nous utilisons cette extension pour obtenir une "meilleure" solution, au sens de l'utilisateur. Nous constatons que la "meilleure" solution est obtenue par un choix de mesures. L'utilisateur est aussi considéré comme mesure.

CHAPITRE C

Application de la Programmation en Logique avec Contraintes au problème d'Emploi du Temps

C.1 Introduction

Dans le chapitre A, nous avons introduit les principales résolutions existantes pour le problème d'emploi du temps : coloration dans un hypergraphe, flot et programmation linéaire entière. Un même exemple nous a permis de montrer leurs limitations. Nous avons pu dégager alors les besoins pour le problème d'emploi du temps.

Dans le chapitre B, la P.L.C. a été présenté pour bien la comprendre. En comparant ces caractéristiques avec les besoins dégagés au chapitre A pour les problèmes d'emploi du temps, nous en avons conclu que la P.L.C. paraît apte à résoudre efficacement ces problèmes.

C'est la raison pour laquelle ce chapitre C présente l'application de la P.L.C. au problème d'emploi du temps.

Dans un premier temps, nous définissons un nouveau modèle pour le problème d'emploi du temps. Ce modèle est basé sur une nouvelle coloration : la coloration multi-dimensionnelle. Nous présentons alors des propriétés de cette coloration, utiles

pour la résolution des problèmes d'emploi du temps.

Dans un deuxième temps, nous définissons la manière d'appliquer la P.L.C. au problème d'emploi du temps avec ce modèle de coloration multi-dimensionnelle. Nous introduisons, alors quelques contraintes.

Dans un troisième temps, nous présentons une implantation particulière à l'aide du langage CLEF v1. Cette implantation porte sur un exemple simple, restreint mais suffisant pour comparer les apports de la P.L.C. par rapport aux résolutions présentées au chapitre A.

C.2 Emploi du Temps : Théorie des graphes et P.L.C.

Ce chapitre présente tout d'abord le modèle et la méthode choisis. Il s'agit d'une modélisation et de résultats personnels. Une première étape définit la coloration multi-dimensionnelle. Puis nous donnons un théorème pour nous faciliter la coloration sur les problèmes d'emploi du temps.

Prenons un réseau de contraintes R($\mathcal{F}$, P) et un espace de recherche égal à I^n. Puis, prenons une coloration des cours $\mathcal{F}$. Celle-ci utilise une palette de couleurs associée à I. Elle respecte les propriétés P. Nous montrons alors l'équivalence entre la satisfaction du réseau R par la technique de la P.L.C. et cette coloration.

Ainsi, nous présentons le lien entre coloration multi-dimensionnelle d'hyperarêtes, problème d'emploi du temps et problème de satisfaction d'un réseau de contraintes.

C.2.1 Coloration multi-dimensionnelle d'hyperarêtes

Nous définissons la coloration multi-dimensionnelle. Un théorème est dégagé pour bien l'utiliser. Nous abordons l'application de cette coloration dans le problème d'emploi du temps.

Nous avons vu qu'un problème d'emploi du temps est représenté par un hypergraphe H et un ensemble de propriétés. Dès à présent, nous utilisons un modèle et une méthode différentes du chapitre A.

N.B. : ***La modélisation est différente de celle présentée au chapitre A. Nous généralisons l'approche vue dans le chapitre A.***

• Le modèle de recherche d'un emploi du temps est une recherche de coloration multi-dimensionnelle. Nous nous basons sur un hypergraphe défini de la même façon qu'avant avec E et $\mathcal{F}$. E est l'ensemble des composants (connues) et $\mathcal{F}$ est celui des compositions. Cette fois, la couleur donnée à une composition est multi-dimensionnelle. Cette couleur se décompose en un spectre où chaque élément du spectre représente un élément d'une dimension.

Dans le cas le plus général du problème d'emploi du temps, le seul composant connu de chaque composition est la durée. Pour chaque composition, les autres composants ne sont pas connues (par exemple, la salle, le professeur, ...). Ces composants sont alors regroupés par type de composant et forment ces dimensions. La couleur multi-dimensionnelle est donc formée d'un élément de chaque dimension.

• La méthode s'appuie sur un hypergraphe Ha particulier. Appliquer cette méthode consiste à faire une coloration multi-dimensionnelle. L'**hypergraphe Ha est construit avec toutes les combinaisons de compositions possibles**. Cet hypergraphe Ha [Ea, $\mathcal{F}a$] sera nommé complet. L'ensemble Ea est l'ensemble de tous les composants et de toutes les couleurs. Une hyperarête de $\mathcal{F}a$ est un ensemble d'éléments de Ea et s'associe à une combinaison de coloration multi-dimensionnelle. A chaque composition, nous représentons toutes les hyperarêtes possibles.

Les propriétés *P* déterminent à partir de Ha un ensemble d'hypergraphes valides. Si une combinaison de coloration est impossible, l'hyperarête correspondante est éliminée.

La fonction g permet de fournir chaque hypergraphe valide. Un hypergraphe obtenu par g représente un Emploi du Temps. Un hypergraphe valide donne alors pour chaque composition une couleur multi-dimensionnelle valide. Le but est de chercher la fonction g donnant un Emploi du Temps et donc une coloration valide.

• La technique que nous utilisons est une technique de cohérence et d'énumération par la P.L.C. sur des réseaux de contraintes. Nous abordons cette technique dans une autre division de ce chapitre.

a) Coloration avec le temps comme palette

Dans notre cas, la fonction qui nous intéresse dans la coloration est celle qui à partir d'une certaine valeur nous rend un intervalle de temps.

Supposons que nous n'ayons que le temps comme palette de coloration.

Notons que, dans ce cas, les compositions sont prédéfinies (au niveau professeur, classe, ...) donc "précolorées". Seule l'affectation de la date est à traiter. Un ensemble de propriétés s'associe à chaque composition prédéfinie. Ces propriétés éliminent pour la composition toutes les couleurs qui ne lui sont pas préaffectées.

Soit DT, un ensemble de valeurs, chaque valeur exprime une durée. L'hypergraphe H est construit à partir de E et de DT : H = (DT $\cup$ E, $\mathcal{F}$). Chaque composition Fk de $\mathcal{F}$ a une unique valeur de DT.

En représentant le temps par T, la fonction g devient :

$$g: \mathcal{F} \rightarrow T$$

$$Fk \rightarrow g(Fk) \cong g(dt/dt \in Fk) = t0 \text{ tel que } [t0, t0+dt[\text{ soit réservé à } Fk$$

Par exemple, soit un cours composé d'une durée (dt) de 2 heures, ce cours peut être affecté par g à la valeur "8 heures" de T. L'intervalle de temps réservé pour ce cours est : 8 heures à 10 heures.

Si deux compositions différentes ont deux valeurs identiques de DT, ces deux compositions n'ont pas obligatoirement le même intervalle de temps.

N.B. : *Il faut noter que DT est différent de T et que l'ensemble temps T est discrétisé. Nous remarquons que T a les mêmes propriétés que l'ensemble I de couleurs.*

Par la suite, quand nous prendrons T, cela implique que tout ensemble ayant les mêmes caractéristiques que T peut être substitué à T. T représente un ensemble de référence.

b) Coloration multi-dimensionnelle

Nous savons que la coloration utilise un "ensemble discret" appelé "palette de couleurs". La coloration multi-dimensionnelle utilise, quand-à-elle, le produit cartésien d'ensembles discrets disjoints.

Définition 1 : Une coloration multi-dimensionnelle utilise une palette PG de couleurs multi-dimensionnelles. Cette palette est le produit cartésien d'ensembles de couleurs uni-dimensionnelles σi. Chaque ensemble est appelé dimension. Nous avons donc, pour m dimensions :

$$PG = \sigma 1 \times ... \times \sigma m$$

Définition 2 : Une projection d'un ensemble EP de points colorés sur une dimension σk correspond à l'ensemble des couleurs utilisées de la dimension σk. Nous notons la projection par le symbole "|". Nous avons :

$$EP|_{\sigma k} \subseteq \sigma k$$

En appliquant le modèle, nous avons l'hypergraphe H [E, $\mathcal{F}$] et I où :

- E représente les composants connus (avec les durées) et déterminés pour chaque composition Fi,
- Fi $\in \mathcal{F}$ est un ensemble d'éléments de E,
- et $I = T \times \sigma 1 \times ... \times \sigma m$

En appliquant la méthode, nous utilisons l'hypergraphe Ha [Ea, $\mathcal{F}a$] où :

- Ea regroupe l'ensemble E et les couleurs de I,
- toute composition Fk $\in \mathcal{F}a$ est un ensemble d'éléments de Ea.

c) Coloration multi-dimensionnelle et Emploi du Temps : Propriétés

Avec l'hypergraphe H [E, $\mathcal{F}$], la palette I et un ensemble de propriétés *P*, il faut chercher une fonction g de $\mathcal{F}$ dans I suivant *P*. La fonction g représente la coloration. La palette de couleurs est I. Nous colorons alors les hyperarêtes $\mathcal{F}$ de H [E, $\mathcal{F}$].

N'oublions pas que l'ensemble I est multi-dimensionnel : $I = T \times \sigma 1 \times ... \times \sigma m$. Ainsi le modèle de coloration consiste à affecter à chaque composition une couleur de chaque dimension.

En résumé, nous avons la fonction de coloration g : $\mathcal{F} \rightarrow I = T \times \sigma 1 \times ... \times \sigma m$ sous les propriétés *P*.

A partir de maintenant, les propriétés de *P* peuvent agir sur plusieurs dimensions. Ainsi toute propriété de *P* entre deux compositions Fi et Fj astreint les valeurs de g(Fi) et de g(Fj) qui sont multi-dimensionnelles.

Nous avons alors : g : $\mathcal{F} \rightarrow I = T \times \sigma 1 \times ... \times \sigma m$

(dt,p,s,...) → (t0, pk,sk, ...) sous *P*

Nous avons introduit la coloration multi-dimensionnelle car elle permet de mieux exprimer les différentes caractéristiques d'un problème d'emploi du temps.

Effectivement, cette coloration exprime le problème désiré au niveau le plus basique. Nous pouvons poser des propriétés non seulement sur les compositions (cours, ...) mais aussi sur les composants (professeur, classe, ...). Nous avons donc une meilleure expression et nous sommes plus à même de satisfaire un utilisateur.

Cependant, si le problème est mieux exprimé, sa résolution est plus difficile. La fonction "g" de placement passe pour chaque composition, d'une unique coloration à une coloration multi-dimensionnelle. La complexité s'en trouve augmentée.

C'est pourquoi, nous nous sommes posés la question si dans certains cas particuliers, le problème formulé avec plusieurs dimensions ne peut pas se ramener à une recherche de colorations "uni-dimensionnelles".

Prenons une fonction γ de coloration dans le temps et des fonctions gi de coloration uni-dimensionnelle dont la palette de couleurs de gi est la dimension σi. Si la fonction "g" de coloration multi-dimensionnelle s'exprime avec la fonction γ et plusieurs fonctions gi, il est intéressant de reporter la recherche sur ces fonctions γ et gi pour constituer cette fonction "g".

Il est logique que toute fonction "g" ne peut pas se décomposer simplement en plusieurs fonctions. Cela dépend principalement des propriétés de "g". Nous allons donc étudier la fonction "g" par ses propriétés et dégager un **cas** particulier pour la problématique de l'emploi du temps.

Par la suite, nous représentons "g" de la manière suivante :

$g = (\gamma, g1, ..., gm) \rightarrow I = T \times \sigma 1 \times ... \times \sigma m$ en respect des propriétés *P*

avec $g(Fk)=g(\{dt,s, ..., r\})=(\gamma(Fk),g1(Fk), ..., gm(Fk))\cong([t0,t0+dt[,sk, ..., rk)$ et $\gamma(Fk)=t0$.

Chaque propriété (P0) de contrainte "générale" est concernée par cette extension à plusieurs dimensions. Cette propriété exprime une relation de non-ubiquité. C'est-à-dire qu'un même composant ne peut pas faire partie de deux compositions au même moment.

Exemple : Si deux cours ont lieu dans un intervalle de temps commun, ils ne peuvent avoir ni le même professeur, ni le même classe, ni la même salle.

Nous allons exprimer la propriété (P0) étendue aux multi-dimensions. Représentons l'ensemble des heures comme T, chaque ensemble de composants de même type par les dimensions σi.

Supposons que nous attribuons un intervalle de *dk* couleurs consécutives de T à une composition Fk. Nous notons cet intervalle par la notation 'Interv' :

$Interv(\gamma(Fk), dk) = [\gamma(Fk) .. \gamma(Fk) + dk[$

La propriété (P0) étendue est alors :

$\forall$ Fk $\in \mathcal{F}$, $\forall$ Fk' $\in \mathcal{F}$, k$\neq$k', dk $\in$ Fk, dk' $\in$ Fk',
Interv(γ(Fk), dk) $\cap$ Interv(γ(Fk'), dk') $\neq \emptyset \Rightarrow$
g1(Fk) $\neq$ g1(Fk') et
g2(Fk) $\neq$ g2(Fk') et
. . .
gm(Fk) $\neq$ gm(Fk').

Ceci est équivalent à la suite de propriétés ($P0^i$):

Interv(γ(Fk), dk) $\cap$ Interv(γ(Fk'), dk') $\neq \emptyset \Rightarrow$ g1(Fk) $\neq$ g1(Fk') et

Interv(γ(Fk), dk) $\cap$ Interv(γ(Fk'), dk') $\neq \emptyset \Rightarrow$ g2(Fk) $\neq$ g2(Fk') et

. . .

Interv(γ(Fk), dk) $\cap$ Interv(γ(Fk'), dk') $\neq \emptyset \Rightarrow$ gm(Fk) $\neq$ gm(Fk').

La propriété (P0) est donc une conjonction de propriétés ($P0^i$).

Pour simplifier l'écriture, nous prenons la propriété "chevauche (Fk, Fj)" suivante :

chevauche (Fk, Fj) : dk $\in$ Fk, dj $\in$ Fj, Interv(γ(Fk), dk) $\cap$ Interv(γ(Fj), dj) $\neq \emptyset$.

Chaque propriété ($P0^i$) peut être formalisé par la propriété A(γ, gi) suivante :

☞ A (γ, gi) : $\forall$ Fk $\in \mathcal{F}$, $\forall$ Fk' $\in \mathcal{F}$, k$\neq$k', chevauche(Fk, Fk') $\Rightarrow$ gi(Fk) $\neq$ gi(Fk')

Nous allons montrer que si γ existe en respect de certaines propriétés que nous allons déterminer, alors gi existe et la propriété A est respectée.

Le problème de recherche de "g", coloration multi-dimensionnelle sous une propriété A se ramènera à une recherche de γ sous d'autres propriétés.

Présentons deux propriétés que nous nommons B et B'.

☞ La propriété B' indique que l'ensemble des compositions $\mathcal{F}$ est isomorphe à celui représentant la dimension σ. Chacun des deux ensembles est partitionné de la même manière en groupes.

Nous utilisons Party(σ) pour exprimer une partition Party de l'ensemble σ.

B' (σ, Part1(σ), Part2($\mathcal{F}$), θ) informe qu'il existe une partition Part1 sur σ, une partition Part2 sur $\mathcal{F}$ et un isomorphisme θ de Part2($\mathcal{F}$) sur Part1(σ).

Cette propriété nous indique que si nous avons un isomorphisme θ et une partition de σ en entrée, alors il en découle une partition sur $\mathcal{F}$.

La figure suivante montre la relation induite entre ces deux ensembles. Un ensemble de compositions ne s'associe qu'à un unique groupe de σ et vice-versa.

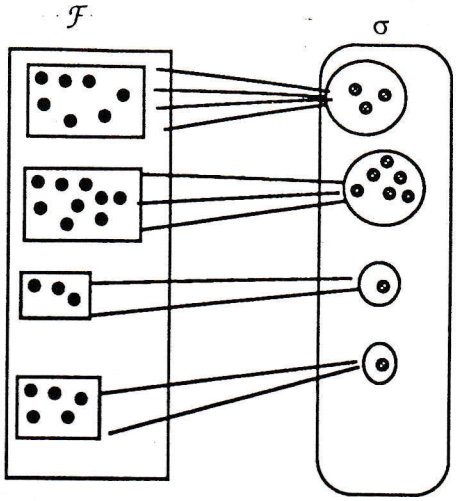


Figure 20. Isomorphisme θ

☞ La propriété B est une contrainte de limitation. Le nombre de compositions appartenant au même élément de partition P de Part2($\mathcal{F}$) et se chevauchant est limité par le nombre de couleurs de l'élément associé de partition de Part1(σ).

Nous utilisons la terminologie Rec(P) pour signifier un recouvrement de P. Rappelons qu'un recouvrement d'un ensemble P est une famille des parties de P tel que l'union des éléments de ce recouvrement est l'ensemble P.

Chaque ensemble G, élément d'un recouvrement, est associé à une composition F1 d'un élément P de partition. Cet ensemble G est formé de toutes les compositions de P qui chevauchent F1. Il y a autant d'ensembles G que de compositions dans P.

La propriété s'exprime de la manière suivante :

B (γ, σ) :

- ∃ σ, ∃ Part1(σ) et B' (σ, Part1(σ), Part2($\mathcal{F}$), θ) respectée
- ∀ P ∈ Part2($\mathcal{F}$), ∃ Rec3(P) tel que ∀ G ∈ Rec3(P),
  - ∃ F1 ∈ G, ∀ F2 ∈ G, chevauche (F1,F2) et
  - ∀ F3 ∈ P, F3 ∉ G, chevauche (F1,F3) est faux,
  - Card(G) ≤ Card(θ(P)) (θ(P) ⊆ Part1(σ))

Le but est de restreindre le nombre de sommets pouvant se dérouler avec une même couleur de T. La limite à ne pas dépasser est le nombre d'éléments du groupe de couleurs de σ.

Ainsi si un groupe de σ a η couleurs, nous limitons à une même couleur de T, η sommets de l'élément de partition de $\mathcal{F}$ associé. Nous avons donc la figure d'attribution suivante :

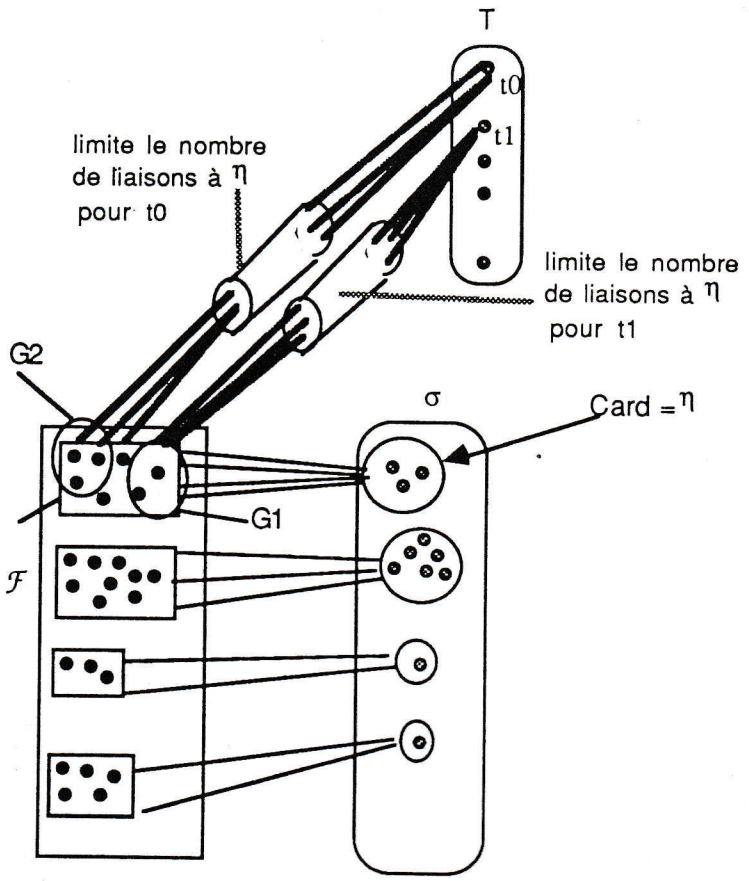


Figure 21. Limitation du nombre de couleurs de σ

☞ Prenons les hypothèses qu'il existe une fonction γ avec B (γ, σi) satisfaite, existe-t-il toujours une fonction gi telle que la propriété A (γ, gi) soit satisfaite?

Le théorème suivant répond à cette question par l'affirmative.

Théorème 1 :
|Soit A, B définies ci-avant, alors :
$|\forall \gamma : \mathcal{F} \rightarrow T, \forall \sigma i, B(\gamma, \sigma i) \Rightarrow \exists gi: \mathcal{F} \rightarrow \sigma i, A(\gamma, gi)$

Preuve :
Prenons une fonction gi de la manière suivante :

Si la propriété B est respectée, nous savons qu'il existe une partition Part2($\mathcal{F}$) et que $\forall P \in$ Part2($\mathcal{F}$), $\exists$ Rec3(P). Prenons gi tel que :

1) $\forall P \in$ Part2($\mathcal{F}$), $\exists$ Rec3(P), $\forall G \in$ Rec3(P), $\forall Fk \in G$, $gi(Fk) \in \theta(P)$.
2) pour chaque G, gi est bijectif entre G et un ensemble L avec $L \subseteq \theta(P)$ et Card(L) = Card(G).

N.B. : ***Ces deux conditions sur gi nous aideront par la suite à trouver cette fonction gi.***

• La fonction gi, ainsi définie, existe-t-elle?

1) gi(Fk) $\in$ θ(P)?

Par construction des partitions en respect de B', nous savons que pour tout cours Fk, Fk appartient à une partition P de $\mathcal{F}$. Nous savons aussi qu'il existe un élément de partition associé (par θ) dans σ qui est θ(P). Donc gi(Fk) $\in$ θ(P) est possible et existe.

2) gi bijectif?

Supposons qu'à chaque élément P, nous lui associons un graphe. Les sommets sont les compositions de P et les arêtes représentent des compositions qui se chevauchent. Chaque ensemble G correspond à toutes les compositions qui chevauchent une composition particulière Fk. Remarquons que le degré de Fk dans P plus une unité (Fk lui-même) est égal au cardinal de G.

Supposons que nous voulons colorer fortement chaque composition, un résultat de la théorie des graphes [BER 87] indique que le nombre chromatique γ(P) est inférieure ou égale au degré maximum plus un. Dans notre cas, cela signifie que $\gamma(P) \leq Card(G_{max})$ pour l'ensemble G correspondant à la composition de degré maximum.

De plus, nous savons que pour chaque ensemble G, Card(G) $\leq$ Card(θ(P)). Donc $\gamma(P) \leq Card(G_{max}) \leq Card(\theta(P))$. Nous avons donc assez de couleurs dans θ(P) pour colorer fortement toutes les compositions de P.

Comme les ensembles P forment une partition de $\mathcal{F}$ et qu'il y a une partition de σ en correspondance par θ, gi existe et est restreint à chaque G, nous pouvons exprimer qu'il est bijectif de G vers un ensemble L de θ(P) et de cardinalité Card(G).

▲ N.B. : *Notre cas est trop restrictif. Effectivement, deux compositions d'un même ensemble G peuvent ne pas se chevaucher et peuvent avoir une même couleur de σ. Ceci implique que la fonction gi définie ci-dessus est trop restrictive et ne donne pas le nombre minimal de couleurs. Cependant, pour trouver ce nombre minimal de couleurs, il faudrait se définir une base de propriétés dans σ.*

Par exemple, si la fonction gi place les salles à des cours, nous nous définissons une base de propriétés de différence et d'égalité. A chaque G, nous aurons un graphe dont les arêtes sont des propriétés de différence. Elles relieront les cours se chevauchant tous entre eux et les cours qui explicitement n'ont pas la même salle. Il faut fournir le nombre chromatique qui donne le nombre de couleurs minimum à avoir (il faut alors remplacer Card(G) dans B par ce nombre).

Nous n'avons pas chercher ces bases pour chaque σ et donc préférer garder la propriété B telle qu'elle est. La recherche de ces bases est un des travaux futurs.

• Il faut savoir que la recherche de γ avec une telle propriété B peut échouer. Cependant, il peut exister une fonction g telle que la propriété A soit satisfaite.

• A est-elle respectée?

Rappelons A : $\forall$ Fk $\in$ $\mathcal{F}$, $\forall$ Fk' $\in$ $\mathcal{F}$, k$\neq$k', chevauche(Fk, Fk') $\Rightarrow$ gi(Fk) $\neq$ gi(Fk')

Remplaçons $\forall$ Fk $\in$ $\mathcal{F}$, $\forall$ Fk' $\in$ $\mathcal{F}$ par une autre formule, en supposant B respectée :

$\forall$ P $\in$ Part2($\mathcal{F}$), $\exists$ Rec3(P), $\forall$ G $\in$ Rec3(P), $\forall$ Fk $\in$ G, ($\forall$ Fk' $\in$ G et $\forall$ Fk' $\notin$ G).

Nous divisons A en une conjonction de deux propriétés :

(a) : $\forall$ P $\in$ Part2($\mathcal{F}$), $\exists$ Part3(P), $\forall$ G $\in$ Rec3(P), $\forall$ Fk $\in$ G, $\forall$ Fk' $\in$ G, k$\neq$k',
chevauche(Fk, Fk') $\Rightarrow$ gi(Fk) $\neq$ gi(Fk'). et

(b) : $\forall$ P $\in$ Part2($\mathcal{F}$), $\exists$ Rec3(P), $\forall$ G $\in$ Rec3(P), $\forall$ Fk $\in$ G, $\forall$ Fk' $\notin$ G, k$\neq$k',
chevauche(Fk, Fk') $\Rightarrow$ gi(Fk) $\neq$ gi(Fk').

Montrons la véracité de (a) et de (b)

• <u>(a) est vraie.</u>

$\forall$ P $\in$ Part2($\mathcal{F}$), $\exists$ Rec3(P), $\forall$ G $\in$ Rec3(P), $\forall$ Fk $\in$ G, $\forall$ Fk' $\in$ G, k$\neq$k',
chevauche(Fk, Fk') $\Rightarrow$ gi(Fk) $\neq$ gi(Fk').

Pour chaque G, gi est bijective de G vers L (L $\subseteq$ θ(P)), cela implique que deux compositions différents de même G ont deux valeurs différentes attribuées par gi, dans σi.

Donc (a) est vraie.

• <u>(b) est vraie</u>

$\forall$ P $\in$ Part2($\mathcal{F}$), $\exists$ Rec3(P), $\forall$ G $\in$ Rec3(P), $\forall$ Fk $\in$ G, $\forall$ Fk' $\notin$ G, k$\neq$k',
chevauche(Fk, Fk') $\Rightarrow$ gi(Fk) $\neq$ gi(Fk').

Prenons un groupe Go précis et deux compositions F1 et F2. Soit F1 chevauche F2, soit F1 ne chevauche pas F2.

• F1 ne chevauche pas F2 implique que le prémisse de la condition est toujours fausse. Donc la condition est toujours vraie. Donc dans ce cas, (b) est vraie.

• F1 chevauche F2, alors il existe un ensemble G1 où par construction des G, F1 et F2 $\in$ G1. Or gi est bijective de tout G sur un L, sous-ensemble de σi. Donc G1 est aussi concerné, donc gi(F1) $\neq$ gi(F2). Donc (b) est vraie dans ce cas.

<u>FIN PREUVE</u>

Le problème d'emploi du temps est dans ce cadre.

Par exemple, il existe une propriété B' qui partitionne l'ensemble des salles par spécialité. Il faut alors trouver une fonction γ qui vérifie la propriété B.

La fonction "g" place des périodes, des salles, des professeurs, des classes aux cours. γ est la fonction de placement aux heures. g1 est la fonction de placement aux salles, ...

Les propriétés A sont des contraintes de non-ubiquité pour les professeurs, pour les classes, pour les salles, ...

Les propriétés B' regroupent les cours utilisant les mêmes salles (et même groupe de salle), les mêmes professeurs (et même groupe de professeur), ...

Les propriétés B limitent le nombre de cours se chevauchant par le nombre de salles disponibles de même spécialité, ...

C.2.2 Technique de coloration en P.L.C.

La technique de coloration multi-dimensionnelle d'hyperarêtes correspond à une recherche de solutions par la P.L.C. satisfaisant un réseau particulier de contraintes.

Les données de base sont constituées à partir de H [E, $\mathcal{F}$] et de I. Nous les décrivons par des clauses, des variables et des domaines.

Les compositions représentent des variables $\mathcal{V}$ de la P.L.C.. Les différentes possibilités de placement de ces compositions constituent le domaine de chaque variable dans chaque dimension.

Les propriétés P que la coloration doit respecter correspondent aux contraintes $\mathcal{C}$. Le placement ou coloration s'effectue par les étiqueteurs de la P.L.C.. Ceci constitue une énumération plus ou moins intelligente des points de l'espace de recherche.

Nous présentons différentes manières d'énumérer les valeurs aux variables. Les placements représentent les énumérations. Ils peuvent dépendre d'informations diverses comme les priorités, les propriétés (ou contraintes) mises en jeu, ...

a) Représentation du problème d'emploi du temps en P.L.C.

i) Hypergraphe en P.L.C. : diverses clauses

Nous décrivons, ici, l'hypergraphe H [E, $\mathcal{F}$]. L'ensemble E décrit les composants. Les hyperarêtes Fk de $\mathcal{F}$ représentent les compositions et donc toutes les combinaisons prédéfinies.

☞ L'ensemble E est formé en P.L.C. par une série de clauses ou de faits.

E est décomposé en sous-ensembles. Chacun d'eux représente un type de composant. Par exemple, nous avons un premier type "professeur", un autre "discipline".

L'ensemble $\mathcal{F}$ est alors composé avec les éléments de ces types de composant.

Des attributs peuvent être associés à chaque composant. Ces attributs sont spécifiés par une liste de valeurs qui caractérise les composants.

La forme d'une clause est alors : "composant (type, nom_composant, liste_d_attributs)".

Exemple : La clause générée pour le professeur Durand sans attribut est :
composant (professeur, Durand, []).

L'ensemble de toutes ces clauses forme l'ensemble E.

ii) Hyperarêtes à colorer : variables sous domaine

Les hyperarêtes à colorer ou compositions sont représentées par une liste de variables de la P.L.C.. Nous ne représentons pas les compositions Fk en soi, mais les places réservées pour la coloration.

Les placements sont spécifiés par $I = T \times \sigma 1 \times ... \times \sigma m$.

Les compositions correspondent aux variables du réseau de contraintes. Chaque dimension de placement est représenté dans un domaine de variable pour chaque composition.

Nous avons à chaque composition Fk, la liste des variables ${}^*v_{T,k}, ..., {}^*v_{\sigma m,k}$.

Nous regroupons les variables de même dimension. Nous avons alors l'ensemble $\mathcal{V}$ des variables de n compositions :

☞ $\mathcal{V} = \mathcal{V}_T \cup ... \cup \mathcal{V}_{\sigma m}$ où $\forall \varphi$, $\mathcal{V}_\varphi = \bigcup_{1 \le k \le n} {}^*v_{\varphi,k}$ pour $I = T \times \sigma 1 \times ... \times \sigma m$.

Une dimension est représentée par un domaine de constantes attribué à chaque variable (de même type).

Le domaine des variables *vi est noté D_i.

N.B. : *Chaque couleur est associée à une constante (ou un nombre entier).*

☞ Pour l'ensemble des variables mises en jeu $\mathcal{V} = \mathcal{V}_{\sigma 1} \cup ... \cup \mathcal{V}_{\sigma m}$ nous avons :
$\forall {}^*v_{\sigma i,k} \in \mathcal{V}_{\sigma i}$, domaine de ${}^*v_{\sigma i,k} = D_{\sigma i,k} = \sigma_i$.

☞ L'espace de recherche S est composé du produit cartésien des domaines de chaque variable, donc pour n compositions : $(D_{T,1} \times ... \times D_{\sigma m,1} \times ... \times D_{T,n} \times ... \times D_{\sigma m,n}) = S$ ou
$(T \times \sigma 1 \times ... \times \sigma m)^n = S$

N.B. : *Certaines variables de la P.L. ne nous intéressent pas. Elles interviennent lors de la résolution par la P.L.C. et ont comme domaine, l'univers d'Herbrand.*

Par la suite, l'espace de recherche S est assimilé au produit cartésien des domaines des seules variables "composition".

iii) Propriétés de coloration : les Contraintes

Si nous désirons prouver la généralité du modèle présenté ici, il faut tenir compte des propriétés déjà prises par les autres résolutions du chapitre A. Il faut aussi prendre des caractéristiques plus générales.

Nous avons défini les variables $\mathcal{V}$ et leur domaine associé. Nous avons alors les sommets du réseau de contraintes. Nous définissons maintenant les contraintes.

Contraintes à la "Prolog"

Pour représenter une partie des propriétés se rapportant à la préaffectation induite

par les compositions de $\mathcal{F}$, nous utilisons les contraintes d'unification sous forme de faits.

Nous allons restreindre l'ensemble $\mathcal{F}a$ à partir de propriétés sur les éléments de E. Avec ces propriétés, nous éliminons certains hypergraphes non valides en imposant certaines combinaisons de Fk de $\mathcal{F}a$.

Ces propriétés sont modélisées par un fait Prolog particulier. Les paramètres de ce fait sont des éléments de E. Ce fait est une relation qui restreint les domaines associés à une composition.

Nous introduisons ici la durée "dt". Pour nous, elle est assimilée à une contrainte de préaffectation.

La représentation d'un fait (ou d'une hyperarête Fi de $\mathcal{F}$) se réalise par une liste d'associations d'un type et de composants :

"Composition ([$type_1$, nom_composant1, ...], ..., [$type_m$, nom_composantm, ...])."

Exemple : pour un cours de physique de deux heures avec le professeur Durand, nous avons : Composition ([Professeur, Durand],[Durée, 2], [Discipline, Physique]).

N.B. : *Ces clauses n'ont pas exclusivement cette forme. Nous pouvons avoir des couplages entre Base de données et Base de connaissances. Les clauses sont alors gérées efficacement dans une Base de données.*

Nous pouvons avoir un schéma relationnel de Base de données associé à cet hypergraphe. Il prend en compte les clauses "composition" et "composant". Toutes les fonctionnalités associées à un S.G.B.D. peuvent être utilisées : tri, vérification de cohérence sur certaines données (existence d'au moins un composant dans chaque type), test sur ces données (nombre d'heures affectées à chaque composant), accès rapide aux informations voulues par des critères précis.

Ce problème de schéma relationnel est bien traité dans [HAN 88].

De plus, il existe des couplages entre Base de données et Base de connaissances. Ces travaux ont été exposés dans la première partie de ce rapport. Ils n'ont pas d'intérêt ici, dans le sens où seuls les problèmes de placement nous intéressent.

Cas 1 : Contraintes pré-définies

Certaines propriétés s'expriment par une ou plusieurs contraintes pré-définies. Ces dernières sont des relations entre des variables. Nous construisons donc le réseau R($\mathcal{V}$, $\mathcal{C}$).

La liste des contraintes pré-définies dans CLEF v1 est la suivante :

- ^diff (a1, a2), ^element (n, l, v), ^au-plus-n=k (n, l, k), ^sigma-idemx (lcoef, l, seuil) et ^= (exp1, exp2), ^< (exp1, exp2), ^> (exp1, exp2), ^<= (exp1, exp2), ^>= (exp1, exp2), ^# (exp1, exp2) et la contrainte implicite d'unification.

Par exemple, nous avons besoin pour la propriété B d'exprimer la disjonction de deux intervalles. Nous allons l'écrire avec le jeu de contraintes ci-dessus sous la forme suivante.

Soit les compositions Fk et Fk' de variables horaires $*v_T$ et $*v_{T'}$, de durée dk et dk'. Le chevauchement passe par deux alternatives : Fk est avant Fk' ou Fk' est avant Fk.

```
disjoint (*vT, *vT', dk, dk') :- ^<= ((+ *vT dk), *vT').
disjoint (*vT, *vT', dk, dk') :- ^<= ((+ *vT' dk'), *vT).
```

Ceci semble peu efficace. Avec n compositions, il faut n (n - 1)÷2 poses de telles clauses. Mais en plus, cela induit $2^{n(n-1)\div 2}$ combinaisons.

Les autres inconvénients concernent l'impossiblité d'exprimer correctement des propriétés d'équilibrage et de compacité.

Par exemple, une propriété d'équilibrage exprime qu'au plus et au moins une moyenne de M cours doit se faire par demi-journée.

Avec le jeu de contraintes disponibles, nous ne pouvons pas exprimer la notion de "au moins M compositions sont présentes". La notion "au plus M compositions sont présentes" peut s'exprimer mais pour des compositions de durée unitaire. Cependant, elle n'est pas possible pour des compositions de durée quelconque.

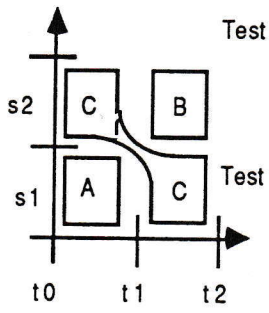
Exemple : Prenons le cas suivant où il existe trois compositions : C de durée 2 unités, A et B de durée une unité. Ces compositions sont à placer sur une plage de deux unités (t0-t1, t1-t2). Seules les deux salles s1 et s2 sont disponibles.

Il faut donc :

- qu'il n'y ait pas plus de deux compositions se chevauchant,
- respecter une propriété de limitation,
- tenir compte des durées et
- placer au maximum deux compositions dans une même salle.

N.B. : *Il ne suffit pas de tester le nombre de compositions présentes à chaque période de temps. Il faut tester ce nombre de compositions sur chaque intervalle horaire de composition.*

Regardons le schéma qui suit. Il nous donne une configuration valide pour le test sur chaque période. Cependant ceci impose que C doit changer de salle. Le test sur chaque intervalle horaire de composition invalide cette configuration.



Test sur chaque période : aucune détection que C doit changer de salle

Nombre de compositions en (t0-t1) : 2 : (A, C)

Nombre de compositions en (t1-t2) : 2 : (B, C)

Test sur chaque intervalle de compositions : détection d'invalidité sur le placement de C.

Nombre de compositions sur l'intervalle de A : 2 : (A, C)

Nombre de compositions sur l'intervalle de B : 2 : (B, C)

Nombre de compositions sur l'intervalle de C : 3 : (A, B, C)

Figure 22. Intégrité de salles

Nous nous apercevons que les contraintes pré-définies ne suffisent pas à exprimer toutes les propriétés.

Cas 2 : Nouvelles contraintes

Notre but est d'aller plus loin que les résolutions existantes. Nous voulons appliquer la P.L.C. à un problème d'emploi du temps assez général. Pour cela, cette partie définit deux nouvelles contraintes. Avec ces nouvelles contraintes et les autres pré-définies, nous allons pouvoir pallier aux inconvénients liés à la durée non unitaire des compositions et aux propriétés d'équilibrage et de compacité.

Ces contraintes se nomment : **^pas-plus-de-n** et **^pas-moins-de-n.**

Le réseau de contraintes R ($\mathcal{V}$, $\mathcal{C}$) est basé sur ces nouvelles contraintes et sur les contraintes pré-définies.

La première contrainte, ^pas-plus-de-n, est une contrainte limitant le nombre de ressources totales.

• **^pas-plus-de-n**

Syntaxe : ^pas-plus-de-n (*n*, *l1*, *l2*, *lcoef*)

n est une constante, *l1* une liste de variables sous domaine, *l2* et *lcoef* des listes de constantes.

Schéma de réveil : (g d g g)

Schéma d'activation : quelque soit le nombre de variables sous domaine.

Cette contrainte exprime qu'au plus *n* ressources sont utilisées.

Nous avons plusieurs objets. Chaque objet est utilisé durant un intervalle de temps. Cet intervalle est spécifié par une variable de début dans *l1* et une durée dans *l2*.

lcoef est le vecteur des ressources (ou besoins) locales à chaque objet.

Par exemple, nous avons deux cours de variables *v1 et *v2, de coefficient égal à 1 et de durée 2 pour chacun de ces cours. Nous voulons limiter le nombre de cours se chevauchant à 1, nous avons l'appel : ^pas-plus-de-n (1, [*v1, *v2], [2, 2], [1, 1]).

Schéma local de saturation :

l1 constitue la liste des débuts Fi d'intervalles,
l2 représente la durée d_i de ces intervalles,
lcoef est la liste des coefficients c_i attribués à chaque intervalle (donc à chaque objet). Elle indique le nombre de ressources utilisées pendant cet intervalle,
n est une limite supérieure de ressources à ne pas dépasser.

Le principe est similaire à celui de la contrainte "^au-plus-n=k". Elle doit maintenir la relation suivante (Le schéma interne s'appuie sur une variable *lrang* supplémentaire d'implantation) :

$$\forall\ Fi \in l1,\ i \in D_{lrang},\ d_i \in l2,\ \sum_{\substack{Fj \in l1,\ d_j \in l2,\ c_j \in lcoef\ /\\ [Fi..Fi+d_i-1] \cap [Fj..Fj+d_j-1] \neq \emptyset}} c_j \leq n$$

Nous utilisons le symbole "\" pour la différence ensembliste : $A \setminus B = \{x \in A, x \notin B\}$.

Algorithme local :
Boucle 1 : Pour tout Fi de *l1*, Fi instancié, $i \in D_{Irang}$, avec sa durée d_i dans *l2* et son coefficient c_i dans *lcoef* faire

Somme := Ø;
Boucle 2 : Pour tout Fj de *l1*, Fj instancié, avec d_j dans *l2* et c_j dans *lcoef* faire

Si $[Fi..Fi+d_i-1] \cap [Fj..Fj+d_j-1] \neq \emptyset$ alors
Somme := Somme + c_j;
Fin Si;

Condition 1: Si Somme > *n* alors
sortir de Boucle 1 par ^pas-plus-de-n insatisfaite;
Fin Si;
Fin Faire;

Condition 2 : Si Somme = *n* alors

Pour tout Fu de *l1*, Fu sous domaine, d_u dans *l2* faire

$D_{Fu}{}^{\text{pas-plus-de-n}} = D_{Fu} \setminus \{w \in D_{Fu} / [Fi..Fi+d_i-1] \cap [w..w+d_u-1] \neq \emptyset\}$
si $D_{Fu}{}^{\text{pas-plus-de-n}} = \emptyset$ alors ^pas-plus-de-n insatisfaite.
Fin Faire ;

Fin si;

$D_{Irang} := D_{Irang} \setminus i$;
Fin Faire ;

Condition 3 : Si tous Fi de *l1*, Fi instancié alors ^pas-plus-de-n satisfaite;
Fin si;

La boucle 2 somme les coefficients des intervalles chevauchant celui indexé par la boucle 1.

La condition 1 détecte les incohérences ($\Sigma c_j > n$).

La condition 2 maintient la cohérence. Elle interdit de faire chevaucher d'autres intervalles avec celui indexé.

La contrainte est satisfaite quand toutes les variables sont instanciées (Condition 3).

En résumé, nous sommes sûrs de détecter les incohérences (condition 1) et de déclarer dans ce cas, l'insatisfaction de cette contrainte.

L'algorithme se termine car le nombre de compositions est fini. La satisfaction et

l'insatisfaction de la contrainte sont toujours détectées.

La deuxième contrainte, ^pas-moins-de-n, est une contrainte imposant un nombre minimum de ressources.

• **^pas-moins-de-n**

Syntaxe : ^pas-moins-de-n (*n*, *l1*, *l2*, *lcoef*)

n représente une constante, *l1* une liste de variables sous domaine, *l2* et *lcoef* des listes de constantes.

Schéma de réveil : (g d g g)

Schéma d'activation : quelque soit le nombre de variables sous domaine.

Cette contrainte exprime qu'au moins *n* ressources sont utilisées.

Chaque objet est utilisé durant un intervalle de temps. Cet intervalle est spécifié par une variable de début dans *l1* et une durée dans *l2*.

De même que pour la contrainte précédente, *lcoef* est le vecteur de ressources (ou besoins) locales à chaque objet.

Par exemple, nous avons deux cours de variable *v1 et *v2, de coefficient égal à 1 et de durée 2 pour chacun de ces cours. Nous voulons avoir au moins deux cours se chevauchant, nous avons l'appel : ^pas-moins-de-n (2, [*v1, *v2], [2, 2], [1, 1]).

Schéma local de saturation :

l1 constitue la liste des débuts Fi d'intervalles,
l2 représente la durée d_i de ces intervalles,
lcoef est la liste des coefficients c_i attribués à chaque intervalle (ou objet). Elle indique le nombre de ressources utilisées pendant cet intervalle,
n correspond à une limite inférieure de ressources.

Le principe est le même que celui de la contrainte "^pas-plus-de-n" (Le schéma interne s'appuie sur une variable supplémentaire *lrang* d'implantation). La contrainte doit maintenir la relation suivante :

$$\forall\ Fi \in l1,\ i \in D_{lrang},\ d_i \in l2,\ \sum_{\substack{Fj \in l1,\ d_j \in l2,\ c_j \in lcoef\ /\\ [Fi..Fi+d_i-1] \cap [Fj..Fj+d_j-1] \neq \emptyset}} c_j \geq n$$

Nous utilisons le symbole "\" pour la différence ensembliste : $A \setminus B = \{x \in A, x \notin B\}$.

Algorithme local :

Boucle 1 : Pour tout Fi de *l1*, Fi instancié, $i \in D_{lrang}$, avec d_i dans *l2* et c_i dans *lcoef* faire

Somme_sûre := Ø;

Somme_possible := Ø;
Liste_de_variables := Ø;
Liste_de_durée := Ø;
Boucle 2 : Pour tout Fj de *l1* avec d_j dans *l2* et c_j dans *lcoef* faire

Si Fj instancié et $[Fi..Fi+d_i-1] \cap [Fj..Fj+d_j-1] \neq \emptyset$ alors
Somme_sûre := Somme_sûre + c_j;
Somme_possible := Somme_possible + c_j;
Fin Si;

Si Fj sous domaine et $\exists\, v \in D_{Fj}$ $[Fi..Fi+d_i-1] \cap [v..v+d_j-1] \neq \emptyset$ alors
Somme_possible := Somme_possible + c_j;
Liste_de_variables := Liste_de_variables $\cup$ Fj;
Liste_de_durée := Liste_de_durée $\cup$ d_j;
Fin Si;

Condition 1 : Si Somme_sûre $\geq$ *n* alors
sortir de Boucle 2;
Fin Si;

Fin Faire;

Condition 2 : Si Somme_possible < *n* alors
sortir de Boucle 1 par ^pas-plus-de-n insatisfaite;
Fin Si;

Condition 3 : Si Somme_possible = *n* alors

Pour tout Fu de Liste_de_variables, d_u dans Liste_de_durée faire

$D_{Fu}{}^{\text{^pas-plus-de-n}} = D_{Fu} \setminus \{w \in D_{Fu} / [Fi..Fi+d_i-1] \cap [w..w+d_u-1] = \emptyset\}$

Fin Faire ;

Fin si;
$D_{Irang} := D_{Irang} \setminus i$;
Fin Faire ;

Condition 4 : Si tous Fi de *l1*, Fi instancié alors ^pas-plus-de-n satisfaite;
Fin si;

La boucle 2 somme les coefficients des intervalles intersectant celui indexé par la boucle 1.

Deux sommes possibles sont faites. La première, la somme sûre, additionne les coefficients des intervalles déjà connus (variables instanciées). La seconde, la somme possible, additionne les coefficients des intervalles qui peuvent intersecter l'intervalle indexé par la boucle 1. Cette deuxième somme concerne les variables sous domaine.

La condition 1 teste si la somme sûre dépasse la limite inférieure. Dans ce cas, la cohérence et la satisfaction de cette contrainte pour l'intervalle indexé sont toujours vraies. Nous testons alors la contrainte sur un autre intervalle (l'index change).

La condition 2 détecte qu'il ne sera pas possible d'atteindre la borne inférieure.

La condition 3 élimine les débuts possibles de ces intervalles qui ne correspondent à aucun chevauchement valide. Ceci rejette donc les valeurs menant à une incohérence.

En résumé, nous sommes sûrs de détecter les incohérences (Condition 2). Dans ce cas, nous déclarons l'insatisfaction de cette contrainte.

La contrainte est satisfaite quand toutes les variables sont instanciées (condition 4).

Le nombre de compositions est fini. Donc l'algorithme se termine. La satisfaction et l'insatisfaction de la contrainte sont toujours détectées.

• **Apports**

• La contrainte "^pas-plus-de-n" permet de prendre en compte les durées et donc les chevauchements entre compositions.

• Chaque composition peut utiliser un nombre de ressources supérieur à un. La composition garde intégralement ces objets pendant tout l'intervalle où elle est placée.

Exemple : Les ressources peuvent représenter des élèves. Nous plaçons plusieurs compositions simultanément dans une salle de conférence. Cette salle n'a qu'un nombre limité de places. Un seul élève peut être attribué à une seule place.

Soit *Liste_de_compositions,* une liste de compositions Fi. Celles-ci sont composées d'un groupe de *Néi* élèves et doivent se dérouler dans la salle de conférence,
soit *Liste_des_durées*, la liste des durées respectives de chaque composition Fi,
soit *Liste_des_coefficients*, la liste des nombres d'élèves Néi pris par chaque composition,
soit *Nombre_de_places*, le nombre de places disponibles dans la salle de conférence,

la contrainte est utilisée comme suit :

^pas-plus-de-n (*Nombre_de_places*, *Liste_de_compositions*, *Liste_des_durées*, *Liste_des_coefficients*).

• La disjonction sur une liste de compositions s'exprime par une seule et unique contrainte "^pas-plus-de-n". Le vecteur des besoins locaux est un vecteur de 1. La limite supérieure, dans ce cas, est $n = 1$.

• La contrainte "^pas-plus-de-n", alliée à "^pas-moins-de-n" permet d'exprimer l'équilibrage. Nous posons conjointement ces deux contraintes. Leur limite (supérieure et inférieure) n est la même (à une unité près). Elle est égale à la moyenne voulue de compositions par demi-journée.

Rappelons que l'équilibrage nous sert pour le problème d'emploi du temps. Par exemple, nous voulons qu'un professeur fasse des cours équitablement répartis dans chaque demi-journée.

• La contrainte "^pas-moins-de-n" permet d'exprimer la compacité en liaison avec des cours fictifs associés à chaque cours réel. Dans le problème d'emploi du temps, la compacité permet de grouper les cours d'un professeur dans un nombre de jours

restreint. Cette propriété assure aussi que deux cours de ce professeur se suivent à une distance minimum.

Nous présentons ultérieurement leur utilisation dans un exemple plus concret.

b) Colorations : Enumérations par étiqueteurs

Nous avons représenter un problème d'emploi du temps en P.L.C. à partir d'un hypergraphe H [E, $\mathcal{F}$], des possibilités d'affectations I = T x σ1 x ... x σm et des propriétés *P*. L'évaluateur P.L.C. peut construire un réseau de contraintes R($\mathcal{V}$, $\mathcal{C}$). Nous connaissons l'espace de recherche S.

Il nous reste à chercher g tel que g : $\mathcal{F} \rightarrow$ I satisfasse *P*. Ceci revient à chercher une coloration pour les compositions Fk par la palette de couleurs I, en respect de *P*. Cette recherche de g permet de trouver les points de S satisfaisant le réseau R($\mathcal{V}$, $\mathcal{C}$).

D'un point de vue P.L.C., ceci revient à énumérer les points de S. L'énumération affecte à chaque variable de "composition" une valeur de son domaine.

Les étiqueteurs de CLEF v1 réalisent cette énumération.

Rappelons les divers prédicats d'"étiquetage" de CLEF v1 (pour plus de détail se reporter au chapitre B) :

- "d-valuer (*term*)" instancie une à une les valeurs de la variable *term* en utilisant le backtrack.

- "pluscontraint (*var, term, liste*)" détecte dans *liste* la variable *var* de plus petit domaine et ayant le plus de contraintes associées. *term* est la nouvelle liste *liste**var*.

- "étiqueter-pluscontraint (**l*)" associe les deux précédents prédicats. En premier, la variable la plus contrainte de **l* est instanciée. L'ordre des variables est tenu à jour dynamiquement. Le processus recommence jusqu'à ce que la liste soit vide.

A partir de ces prédicats, plusieurs stratégies peuvent être dégagées. La coloration des compositions Fk de $\mathcal{F}$ correspond à l'instanciation de leurs variables $\mathcal{V}$. Nous supposons que toutes les variables sont regroupées dans la liste *$\mathcal{V}$.

i) Stratégie 1 : Ordre d'apparition des variables

Cette stratégie place les compositions suivant l'ordre d'apparition de leurs variables. La stratégie utilise les clauses "apparition" ("apparition(*$\mathcal{V}$)"). Elles s'écrivent en CLEF v1 de la manière suivante :

```
apparition ([]).
apparition ([*vprem | *suite]) :-
          d-valuer (*vprem),
          apparition(*suite).
```

ii) Stratégie 2 : Ordre du "pluscontraint"

La stratégie utilise le prédicat "étiqueter-pluscontraint($^*\mathcal{V}$)".

Celle-ci répond de la manière la plus fidèle à l'expérience d'un concepteur.

Effectivement, les compositions les moins disponibles ont peu de possibilités de placement. Le domaine de ces compositions est réduit. Elles sont donc plus contraintes.

Exemple : c'est le cas des compositions d'un professeur peu disponible. Nous plaçons des contraintes d'indisponibilité. Le domaine des variables concernées est diminué des valeurs indisponibles. Ceci se traduit par un domaine plus restreint que les autres professeurs. Le placement est plus prioritaire.

D'autres stratégies existent se rapprochant de plus en plus du backtrack intelligent.

iii) Utilisation de la propriété sur le temps

Nous supposons qu'il existe des propriétés similaires à (P0) composées en propriétés A définies dans la division C.2.1. Nous utilisons le théorème 1 pour placer des propriétés équivalentes à B.

Les compositions Fk sont affectées en premier lieu sur T. En second lieu, elles sont placées sur les σi. Ceci revient à étiqueter toutes les variables $\mathcal{V}_T (= {}^*\mathcal{V}_{T,k})$ avant les variables $\mathcal{V}_{\sigma i} (= {}^*\mathcal{V}_{\sigma i,k})$.

Dans le cadre du théorème 1, la recherche de g passe par celle d'une fonction $\gamma : \mathcal{F} \rightarrow T$. Le programme est alors formé comme suit :

1) Placer les contraintes sur T,

2) Placement (par γ) des Fk de $\mathcal{F}$ sur T en étiquetant ($\forall$ k), ${}^*\mathcal{V}_{T,k}$, tel que les contraintes représentant les propriétés B soient respectées.

3) Poser les contraintes sur σi,

4) Placement (par gi) des Fk de $\mathcal{F}$ sur σi en étiquetant ($\forall$ k), ${}^*\mathcal{V}_{\sigma i,k}$ (les deux aides présentées dans la preuve du théorème 1 nous permet de trouver sûrement ces gi).

5) Les étapes 3) et 4) sont appliquées à chaque dimension σj.

N.B. : *Le placement se fait sur chaque dimension : T, $\sigma 1$, ..., σm. C'est au programmeur de faire le lien entre les dimensions, par les propriétés ou contraintes appropriées (exemple, B et B').*

C.3 Exemple récapitulatif

C.3.1 Présentation de l'exemple

Avec cet exemple, nous montrons que les problèmes d'emploi du temps sont mieux résolus par la P.L.C. que les autres résolutions existantes. Par ce biais, nous constatons plus facilement les apports de la P.L.C. et ses limitations.

Nous avons plusieurs extensions. Nous affectons des heures et des salles. Les durées des cours peuvent être d'une heure, d'une heure et demi ou de deux heures.

Le nombre de composants est quelconque. Une composition représente un cours. Il ne se restreint pas à deux composants. Un cours peut avoir des groupes d'élèves, des professeurs, des salles et des disciplines.

Le critère de "bonne" solution est uniquement constitué par le désir de chacun (exprimé par des indisponibilités, des préaffectations, ...).

Nous allons définir les composants (E). Puis nous présentons les compositions ($\mathcal{F}$), les places d'affectation (I) et enfin les diverses propriétés (P).

a) Les composants : (professeur, salle, etc.)

L'ensemble E des composants est partitionné en sous-ensembles décrivant chaque type de composant.

Nous supposons définis les ensembles suivants : $\mathcal{El}$ ensemble des élèves et $\mathcal{S}$ ensemble des salles existantes. Ainsi, E est constitué de la manière suivante :

• $E = E_p \cup E_{Gé} \cup E_{Gs} \cup E_{Dis} \cup E_{durée}$ où

- E_p est l'ensemble des professeurs,

- $E_{Gé}$ est un ensemble de groupes $Gé_i$ d'élèves tel que :
  $\forall Gé_i \in E_{Gé}, Gé_i \subseteq \mathcal{El}$

- E_{Gs} est un ensemble partitionné de groupes de salles de même spécialité,
  $\forall Gs_i \in E_{Gs}, Gs_i \subseteq \mathcal{S}$ et
  $\forall Gs_i \in E_{Gs}, \forall Gs_j \in E_{Gs}, (i \neq j), Gs_i \cap Gs_j = \emptyset$.

▲ Cet ensemble E_{Gs} constitue une partition de $\mathcal{S}$. En comparaison avec le théorème 1, $\mathcal{S}$ est un ensemble σ_i. E_{Gs} est la partition $Part1(\sigma_i)$.

- E_{Dis} est l'ensemble des disciplines. Celui-ci est divisé en deux sous-ensembles :
  $E_{Dislégère}$ pour les disciplines légères,
  $E_{Dislourde}$ pour les disciplines lourdes avec :
  $E_{Dislégère} \cap E_{Dislourde} = \emptyset$,
  $E_{Dislégère} \cup E_{Dislourde} = E_{Dis}$.

- $E_{durée}$ est l'ensemble des durées possibles par unité d'une demi-heure.

Tous les éléments de $E \setminus E_{Gé}$ sont indépendants deux à deux :

$\forall e1 \in E \setminus E_{Gé}, \forall e2 \in E \setminus E_{Gé}, e1 \neq e2$
où "$\neq$" peut exprimer la différence entre deux constantes ou l'intersection vide entre deux ensembles ($e1 \cap e2 = \emptyset$).

b) Les compositions : (cours, etc.)

• Une composition est un cours Fk tel que

$Fk = \{e_{pk}, e_{Gék}, e_{Gsk}, e_{Disk}, e_{duréek}\}$ avec

- $e_{pk} \neq \emptyset$ et $e_{pk} \subseteq E_p$ (plusieurs professeurs sont possibles pour un même cours),

- $e_{Gék} \neq \emptyset$ et $e_{Gék} \subseteq E_{Gé}$ (plusieurs groupes d'élèves possibles),

- $e_{Gsk} \neq \emptyset$ et $e_{Gsk} \in E_{Gs}$ (un seul groupe de salles possibles),

- $e_{Disk} \neq \emptyset$ et $e_{Disk} \in E_{Dis}$ (une seule discipline possible),

- $e_{duréek} \neq \emptyset$ et $e_{duréek} \in E_{durée}$ (une seule durée possible).

En résumé, un cours peut réunir plusieurs groupes d'élèves. Il n'a qu'un seul groupe de salles. Il peut être consitué de plusieurs professeurs. Ce cours ne peut avoir qu'une unique discipline et qu'une unique durée.

▲ • Dans notre cas, chaque cours a un groupe de salle associé. En comparaison avec le théorème 1, la préaffectation des cours à un groupe de salle représente la fonction θ. L'utilisateur doit assurer que θ est un isomorphisme entre $\mathcal{F}$ et $\mathcal{S}$.

• Avec la structure de E_{Gs} et des cours, nous avons constitué la propriété B'.

Tous les cours Fk, formés de cette manière constituent l'ensemble $\mathcal{F}$.

c) Les possibilités d'affectation

L'ensemble $\mathcal{I}$ est le produit cartésien de deux ensembles : $\mathcal{S}$ et $\mathcal{H}$ ($\mathcal{H}$ est l'ensemble T théorique).

• Rappelons que $\mathcal{S}$ est l'ensemble des salles existantes dans l'établissement.

• $\mathcal{H}$ est l'ensemble des demi-heures possibles du Lundi 8 heures au Samedi 12 heures. Cet ensemble a des particularités dont $\mathcal{I}$ hérite.

$\mathcal{H}$ est décomposé en une partition $\{\mathcal{H}_t\}$. L'indice t correspond à un jour de la semaine. Nous avons :

° $\forall \mathcal{H}_{ta} \in \mathcal{H}, \forall \mathcal{H}_{tb} \in \mathcal{H}, (ta \neq tb), \mathcal{H}_{ta} \cap \mathcal{H}_{tb} = \emptyset$,

° $\bigcup_{ti} \mathcal{H}_{ti} = \mathcal{H}$; ti $\in$ {Lundi, ..., Samedi}

° Nous supposons qu'il y a w jours.

° Il existe un ordre total, "<", sur $\mathcal{H}$ tel que Lundi 8h < Lundi 8h 30 < ... < Samedi 12h.

Chaque $\mathcal{H}_t$ est décomposé en deux sous-ensembles $\{\mathcal{H}_{t,1}, \mathcal{H}_{t,2}\}$. Chacun d'entre eux représente chaque demi-journée de ce jour t :

° $\mathcal{H}_{t,1} \cap \mathcal{H}_{t,2} = \emptyset$,
° $\mathcal{H}_{t,1} \cup \mathcal{H}_{t,2} = \mathcal{H}_t$.

Nous connaissons deux autres ensembles, Fermetures et Récréation. Ils s'associent aux heures de fermeture (12 heures et 18 heures chaque jour) et à celles de récréation (à 10 heures et 16 heures chaque jour) :

° Fermetures = {Lundi 12h, Lundi 18 h, ..., Samedi 12h},

° Récréation = {Lundi 10 h, Lundi 16 h, ..., Samedi 10 h}.

° Certaines heures n'existent pas dans $\mathcal{H}$: chaque jour, les heures entre 12 heures et 14 heures et celles entre 18 heures et 8 heures le lendemain.

L'ensemble I est décomposé en une partition avec les demi-journées $I_{t,i}$ ($i^{ième}$ demi-journée du jour t) :

• $I = \bigcup I_{t,i}$ pour t du Lundi au Samedi et i de la $1^{ère}$ à la $2^{ème}$ demi-journée,

• $\forall I_{t,i} \in I, \forall I_{r,j} \in I, (t \neq r)$ ou $(i \neq j), I_{t,i} \cap I_{r,j} = \emptyset$

d) Placement et astreintes

La fonction g place les cours dans une salle à une heure précise. Elle est la suivante :

$g : \mathcal{F} \rightarrow I = \mathcal{H} \times \mathcal{S}$
$Fk \rightarrow (heure_k, salle_k)$

Nous utilisons les notations qui suivent pour un cours Fk de durée dk :

° $g(Fk)_{|\mathcal{H}}$ est le temps de début du cours Fk,
° $g(Fk)_{|\mathcal{S}}$ est la salle affectée au cours Fk et
° Interv($g(Fk)_{|\mathcal{H}}$, dk) exprime l'intervalle de temps occupé par Fk.

La fonction doit respecter un ensemble de propriétés *P*. Nous allons maintenant les découvrir. Avant chaque propriété, une présentation informelle est donnée afin de mieux la comprendre.

i) Contraintes générales

■ Aucun cours ne doit chevaucher les heures de fermetures. Un cours fictif est associé à une heure de fermeture (heure de début et durée). Chaque cours réel est en disjonction avec ces cours fictifs.

Rappelons que "<" est l'ordre total sur $\mathcal{H}$ tel que Lundi 8h < Lundi 8h 30 < ... < Samedi 12h.

Les heures hu et hv nous servent uniquement pour calculer la durée d'une heure de fermeture hi. Les durées associées à un cours Fk se noteront toujours dk. Ainsi par la suite, nous les utiliserons sans expliciter leur appartenance.

(P0) : $\forall$ Fk $\in \mathcal{F}$, $\forall$ hi $\in$ Fermetures,

$\exists$ hu $\in \mathcal{H}$, hi < hu, $\forall$ hv $\in \mathcal{H}$, hv≠hu, hv≠hi, (hv < hu et hv < hi) ou (hu < hv et hi < hv) alors

$$\text{Interv } (g(Fk)_{|\mathcal{H}}, dk) \cap \text{Interv } (hi, (hu - hi)) = \emptyset.$$

■ La deuxième proposition concerne les groupes d'élèves. Les cours ayant des élèves en commun ne peuvent pas se dérouler en même temps.

(P0') : $\forall$ Fk $\in \mathcal{F}$, $\forall$ Fk' $\in \mathcal{F}$, (k≠k'), $\forall$ $Gé_k \in$ Fk, $\forall$ $Gé_{k'} \in$ Fk',

$$(Gé_k \cap Gé_{k'} \neq \emptyset) \Rightarrow (\text{Interv } (g(Fk)_{|\mathcal{H}}, dk) \cap \text{Interv } (g(Fk')_{|\mathcal{H}}, dk') = \emptyset)$$

■ Deux cours ayant des périodes de temps communes ne doivent pas avoir lieu dans la même salle.

▲ • Nous constatons que cette propriété est équivalente à la propriété A du théorème 1 (C.2.1).

(P0") : $\forall$ Fk $\in \mathcal{F}$, dk $\in$ Fk, $\forall$ Fk' $\in \mathcal{F}$, dk' $\in$ Fk', (k≠k'),

$$(\text{Interv}(g(Fk)_{|\mathcal{H}}, dk) \cap \text{Interv}(g(Fk')_{|\mathcal{H}}, dk') \neq \emptyset) \Rightarrow (g(Fk)_{|\mathcal{S}} \neq g(Fk')_{|\mathcal{S}})$$

■ Tout cours doit avoir lieu dans une salle de son groupe.

(P0"bis) : $\forall$ Fk $\in \mathcal{F}$, $Gs_k \in$ Fk, $Gs_k \in E_{Gs}$, $g(Fk)_{|\mathcal{S}} \in Gs_k$.

▲ • Avec la structure spécifique de E_{Gs}, cette propriété constitue une des deux aides pour trouver la fonction de coloration uni-dimensionnelle équivalente à $g_{|\mathcal{S}}$ (cf la preuve du théorème 1).

■ Deux cours, ayant au moins un professeur en commun, ne doivent pas se dérouler au même moment.

(P0"') : $\forall$ Fk $\in \mathcal{F}$, $\forall$ Fk' $\in \mathcal{F}$, (k≠k'), $\{P_k\} \in$ Fk, $\{P_{k'}\} \in$ Fk',

$$(\{P_k\} \cap \{P_{k'}\} \neq \emptyset) \Rightarrow \text{Interv } (g(Fk)_{|\mathcal{H}}, dk) \cap \text{Interv } (g(Fk')_{|\mathcal{H}}, dk') = \emptyset$$

ii) Contrainte de préaffectation

■ Un cours choisi ne doit se dérouler que pendant une après-midi $I_{t,2}$. Cette contrainte est facultative.

(P1) : Pour Fk $\in \mathcal{F}$, Fk choisi, Interv $(g(Fk)_{|\mathcal{H}}, dk) \subseteq I_{t,2}$

iii) Contrainte d'indisponibilité

■ Certains cours ne doivent pas se dérouler à une période précise de temps. Ici, aucun cours ne doit être interrompu par une récréation. L'ensemble des récréations a été défini avant.

(P2) : $\forall$ Fk $\in \mathcal{F}$, Interv $(g(Fk)_{|\mathcal{H}}, dk) \cap$ Récréation $= \emptyset$.

iv) Contrainte d'équilibrage

Pour chaque groupe d'élèves, les disciplines légères doivent être bien réparties par demi-journée. Supposons qu'il y a w journées, et 2 w -1 demi-journées.

■ Nous cherchons d'abord le nombre de cours pour chaque groupe d'élèves ayant une discipline légère (Card(J2)). Puis nous calculons la moyenne de cours à placer à chaque demi-journée. Nous obligeons le nombre de tels cours par demi-journée (Card(J1)) à être égal à cette moyenne.

(P3) : $\forall$ $Gé_u \in E_{Gé}$, $\forall$ $I_{t,i} \in I$,

$$J1 = \{Fk \in \mathcal{F},\ Gé_u \in Fk,\ disck \in E_{Dislégère},\ Interv\ (g(Fk)_{|\mathcal{H}}, dk) \subseteq I_{t,i}\},$$

$$J2 = \{Fk \in \mathcal{F},\ Gé_u \in Fk,\ disck \in E_{Dislégère},\ disck \in Fk\},$$

$$[Card(J2) \div (2\ w\ -1)] \leq Card(J1) \leq [Card(J2) \div (2\ w\ -1)]^*$$

v) Contrainte de compacité

■ Les cours de professeurs choisis sont placés d'une façon la plus compacte possible. Nous stipulons que deux cours consécutifs doivent avoir un "trou" minimum.

Un "trou" est l'espace temporel non occupé entre la fin du premier cours et le début du deuxième. Nous le désignons par sa durée dtrou.

(P5''') : Pour $P_u \in E_P$, P_u choisi,
soit Max le dernier des cours :
Max = F_{max} tel que Maximum $\{g(Fk)_{|\mathcal{H}}, P_u \in Fk\} = g(F_{max})_{|\mathcal{H}}$,

$\forall$ Fk $\in \mathcal{F}$, Fk $\neq$ Max, $P_u \in$ Fk, alors $\exists$ Fv $\in \mathcal{F}$, $P_u \in$ Fv tel que :
$dk \leq g(Fv)_{|\mathcal{H}} - g(Fk)_{|\mathcal{H}} \leq dk + dtrou$

vi) Contrainte de limitation

■ Certains cours de même groupe de salles peuvent se dérouler en même temps. Le nombre de ces cours ne doit pas dépasser le nombre de salles disponibles.

(P6) : $\forall Gs_k \in E_{Gs}$, $\forall Fk \in \mathcal{F}$, $Gs_k \in Fk$,

$J = \{Fu / Fu \in \mathcal{F}, Gs_k \in Fu,$
Interv $(g(Fk)_{|\mathcal{H}}, dk) \cap$ Interv $(g(Fu)_{|\mathcal{H}}, du) \neq \emptyset\}$ alors $Card(J) \leq Card(Gs_k)$

▲ • Cette propriété est équivalente à la propriété B du théorème 1 (C.2.1) :

- nous avons vu que la propriété B' est respectée (cf b) avec la partition E_{Gs} de $\mathcal{S}$.
- un élément P de partition de $\mathcal{F}$ est un groupe de cours ayant le même groupe Gs_k de salles.
- les groupes, nommés ici J, représentent les éléments G du recouvrement de P. Il y a un élément par composition Fk de P et tous les cours qui chevauchent Fk constitue J.
- l'inégalité $Card(G) \leq Card(\theta(P))$ est représentée par $Card(J) \leq Card(Gs_k)$.

Cet exemple présente différents types de contraintes. Elles englobent celles des diverses résolutions présentées dans la première partie de ce rapport.

Nous abordons maintenant l'application de la P.L.C sur cet exemple.

C.3.2 Implantation avec la P.L.C.

Nous allons appliquer la P.L.C. avec la technique décrite en C.2 sur cet exemple.

a) Représentation de l'exemple en P.L.C.

Nous introduisons dans cette partie, les variables. Elles constituent les sommets de notre réseau à satisfaire. Nous présentons d'abord les clauses s'associant à l'hypergraphe H[E, $\mathcal{F}$]. Puis nous définissons l'ensemble des variables correspondant aux cours. Enfin l'ensemble des affectations possibles est décrit par le domaine de chaque variable.

i) L'hypergraphe en P.L.C.

Nous décrivons les diverses clauses qui modélisent l'hypergraphe de notre exemple. Nous trouvons deux sortes de clauses. La première constitue l'ensemble E et donc les composants. La deuxième représente $\mathcal{F}$ ou les cours.

☞ Nous savons que l'ensemble $E = E_p \cup E_{Gé} \cup E_{Gs} \cup E_{Dis} \cup E_{durée}$.

Chaque composant appartient à un type précis : professeur, groupe d'élèves, ...

Rappelons la forme générale d'une clause pour E :

"composant (type_de_composant, nom_de_composant, liste_d_attributs)."

Pour chaque sous-ensemble de E, nous obtenons les cinq types suivants de clause :

- pour un professeur de nom 'nom_P1', sans attribut :

  "composant (professeur, nom_P1, [])"

- pour un groupe d'élèves 'nom_G1', avec les noms des élèves :

  "composant (groupe d'élèves, nom_G1, [élève1, élève2, ...])."

- pour un groupe de salles 'nom_S1', avec les salles de ce groupe :

  "composant (groupe de salles, nom_S1, [salle1, salle2, ...])."

- pour une discipline 'nom_D1' avec son poids (par exemple légère) :

  "composant (discipline, nom_D1, [légère])."

- pour une durée de valeur de 2 unités, sans attribut :

  "composant (durée, 2, [])."

Une union de telles clauses forme l'ensemble E de notre exemple.

☞ L'ensemble $\mathcal{F}$ est tel que :

$$\mathcal{F} = \{Fk / Fk = \{e_{pk}, e_{Gék}, e_{Gsk}, e_{Disk}, e_{duréek}\} \text{ avec } e_{pk} \neq \emptyset,\ e_{Gék} \neq \emptyset,\ e_{Gsk} \neq \emptyset,\ e_{Disk} \neq \emptyset,\ e_{duréek} \neq \emptyset,\ e_{pk} \subseteq E_p,\ e_{Gék} \subseteq E_{Gé},\ e_{Gsk} \in E_{Gs},\ e_{Disk} \in E_{Dis},\ e_{duréek} \in E_{durée}\}$$

Rappelons qu'une clause de composition, donc de cours, a la forme :

"composition ([type1, nom_composant1, ...], ...)."

Les vérifications sur la formation des cours (existence d'un composant de chaque type, existence du composant choisi), ne sont pas complexes. Elles ne nous intéressent pas. Elles ne sont pas présentées ici.

Ainsi les hyperarêtes Fk ont la forme suivante :

"composition ([professeur, Nom_P1, ...], [groupe d'élèves, Nom_G1, ...],
[groupe de salles, Nom_S1], [discipline, Nom_D1],
[durée, Une_durée])."

ii) <u>Hyperarêtes à colorer : variables sous domaine</u>

Dans le cas général, $I = T \times \sigma 1 \times ... \times \sigma m$, $\mathcal{V} = \mathcal{V}_T \cup ... \cup \mathcal{V}_{\sigma m}$ et $\mathcal{V}_\varphi = \bigcup_{Fk \in \mathcal{F}} {}^*v_{\varphi,k}$

Ici seules $\mathcal{H}$ et $\mathcal{S}$ nous intéressent. Les autres composants sont déjà préaffectés par les contraintes compositionnelles. Donc $I = \mathcal{H} \times \mathcal{S}$. L'ensemble T correspond à $\mathcal{H}$.

Nous associons donc, à chaque cours Fk deux variables principales :

☞ la variable $*v_{\mathcal{H}k}$ ou variable horaire et
$*v_{\mathcal{S}k}$ la variable de salle.

Rappelons que I se décompose en demi-journées. Nous ajoutons alors une seconde variable horaire de demi-journée :

☞ la variable de demi-journée $*v_{\mathcal{H}itk}$

Les trois variables $*v_{\mathcal{H}k}$, $*v_{\mathcal{S}k}$, $*v_{\mathcal{H}itk}$ représentent l'hyperarête Fk.

Donc $\mathcal{V} = (\mathcal{V}_{\mathcal{H}} \cup \mathcal{V}_{\mathcal{H}it}) \cup \mathcal{V}_{\mathcal{S}}$.

$\mathcal{H}$ représente les demi-heures possibles à affecter et $\mathcal{S}$ les salles disponibles.

Rappelons que "<" est un ordre total sur $\mathcal{H}$ tel que Lundi 8h < Lundi 8h 30 < ... < Samedi 12h.

☞ Le domaine des variables horaires ou de demi-journées est l'ensemble des rangs des valeurs possibles dans $\mathcal{H}$. Initialement, le domaine de $*v_{\mathcal{H}k}$ correspond à tous les rangs de chaque demi-heure : [1, .., 99]. Le domaine de $*v_{\mathcal{H}itk}$ représente les rangs de chaque demi-heure commençant chaque demi-journée : [1, 10, .., 91].

☞ Le domaine des variables de salles est l'ensemble des salles disponibles.

N.B. : *En pratique, des valeurs numériques sont associées aux salles ou aux demi-heures. Pour des raisons de lisibilité, le symbole associé est employé plutôt que la valeur numérique.*

• Ainsi nous avons :

$\forall\ *v_{\mathcal{H}k} \in \mathcal{V}_{\mathcal{H}}$, $D_{\mathcal{H}k}$ = {Lundi 8h, Lundi 8h30, ..., Samedi 12h},
$\forall\ *v_{\mathcal{H}itk} \in \mathcal{V}_{\mathcal{H}it}$, $D_{\mathcal{H}itk}$ = {Lundi 8h, Lundi 14h, ..., Samedi 8h},
$\forall\ *v_{\mathcal{S}k} \in \mathcal{V}_{\mathcal{S}}$, $D_{\mathcal{S}k}$ = {Salle1, ..., Sallez}.

Rappel : La déclaration du domaine d'une variable passe par le prédicat "domaine" de CLEF v1 : "domaine ($*v_{\mathcal{H}k}$, [Lundi_8h, ...])."

iii) Propriétés de coloration : les Contraintes

Nous abordons les propriétés *P*. Nous les transformons en contraintes de CLEF v1.

Nous nous basons sur un ensemble de contraintes qui sont :
-^diff,
-^= (exp1, exp2),
-^<= (exp1, exp2),
-^element (n, l, v),
-^pas-plus-de-n (n, l1, l2, lcoef),
-^pas-moins-de-n (n, l1, l2, lcoef) et
-la contrainte d'unification à travers la résolution CLEF v1.

• **Contraintes entre variables horaires et de demi-journée**

Revenons à l'univers des nombres. Le domaine des variables est un ensemble de valeurs numériques.

$^*v_{\mathcal{H}k}$ correspond aux rangs dans $\mathcal{H}$ ou aux heures possibles du cours Fk :

Lundi 8 h est associé à 1, Lundi 8 h 30 est associé à 2, ...

La valeur de chaque demi-heure est remplacée par celle de la première demi-heure de la demi-journée qui l'inclut. Nous obtenons :

La valeur 1 correspond à Lundi_8h. Nous la remplaçons par Lundi_8h ou la valeur 1. La valeur 2 correspond à Lundi_8h30. Nous la remplaçons par Lundi_8h ou 1, ...

Donc *DJ = {Lundi 8h, ..., Lundi8h, Lundi 14h, ..., Samedi 8h} a 9 fois la valeur Lundi 8h (ou 1) pour les valeurs des demi-heures comprises entre Lundi 8h et Lundi 12h. Elle a 9 fois la valeur Lundi 14h (ou 10) pour les valeurs des demi-heures du Lundi 14h au Lundi 18h. Et ainsi de suite.

Nous posons la contrainte qui suit pour passer d'une variable horaire à une variable de demi-journée :

- $^*v_{\mathcal{H}k}$, la variable horaire de Fk,
- $^*v_{\mathcal{H}itk}$, la variable de demi-journée de Fk,
- *DJ, la liste {Lundi 8h, ..., Lundi8h, Lundi 14h, ..., Samedi 8h}

et ^element ($^*v_{\mathcal{H}k}$,*DJ, $^*v_{\mathcal{H}itk}$).

donc $^*v_{\mathcal{H}itk}$ est le $^*v_{\mathcal{H}k}^{\text{ième}}$ élément de *DJ.

Si $^*v_{\mathcal{H}k}$=3, elle correspond à la troisième demi-heure de la semaine (ou Lundi_9h) donc $^*v_{\mathcal{H}itk}$ a la troisième valeur de *DJ ou Lundi_8h.

• **Contraintes générales**

• Première propriété (P0) : le placement de tout cours ne doit pas chevaucher les heures de fermetures, donc

(P0) : $\forall$ Fk $\in \mathcal{F}$, $\forall$ hi $\in$ Fermetures,

$\exists$ hu $\in \mathcal{H}$, hi < hu, $\forall$ hv $\in \mathcal{H}$, hv $\neq$ hi, hv$\neq$hu, (hv < hu et hv < hi) ou (hu < hv et hi < hv) alors

Interv $(g(Fk)_{|\mathcal{H}}, dk) \cap$ Interv (hi, (hu - hi)) = $\emptyset$.

Nous posons des disjonstions entre le cours Fk et les heures de fermetures.

Nous créons autant de cours fictifs que d'heures de fermeture. Leur durée ici est d'une unité.

Nous posons la contrainte "^pas-plus-de-n" entre chaque cours réel et ces cours fictifs. La limite supérieure à ne pas dépasser est 1. Nous avons par cours Fk :

- *lvk, une liste comprenant la variable horaire $*v_{\mathcal{H}k}$ et la liste des heures de fermeture,

- *ldk, une liste comprenant la durée de Fk, et la durée des fermetures,

- *lcoefk, une liste de coefficients tous égaux à 1,

- une limite supérieure *n* à ne pas dépasser, soit 1,

et la contrainte : ^pas-plus-de-n (1, *lvk, *ldk, *lcoefk).

N.B. : *Dès la déclaration de ces contraintes, nous avons une unique variable sous domaine et toutes les autres instanciées. La contrainte se déclenche donc immédiatement. La limite est déjà atteinte car les heures de fermeture sont connues. A la sortie de la contrainte, l'état de satisfaction est connu (satisfait ou insatisfait).*

• La deuxième propriété concerne la disjonction entre cours qui ont des élèves en commun :

(P0') : $\forall$ Fk $\in \mathcal{F}$, $\forall$ Fk' $\in \mathcal{F}$, (k≠k'), $\forall$ $Gé_k \in$ Fk, $\forall$ $Gé_{k'} \in$ Fk',
$(Gé_k \cap Gé_{k'} \neq \emptyset) \Rightarrow$ (Interv $(g(Fk)_{|\mathcal{H}}$, dk) $\cap$ Interv $(g(Fk')_{|\mathcal{H}}$, dk') = $\emptyset$)

La contrainte "^pas-plus-de-n" est utilisée. Nous sélectionnons les cours d'un groupe Gi d'élèves et les cours possédant un élève en commun avec le groupe Gi. Nous posons la contrainte sur les variables horaires de ces cours. La limite supérieure est égale à 1. Ce procédé est effectué pour chaque groupe d'élèves.

Nous avons :

- $*lv_{Gi}$, la liste des variables horaires $*v_{\mathcal{H}k}$ des cours dépendant par au moins un des élèves de Gi,

- $*ld_{Gi}$, une liste composée des durées respectives,

- $*lcoef_{Gi}$, une liste de coefficients 1,

- une limite supérieure *n* à ne pas dépasser, égale à 1,

et la contrainte : ^pas-plus-de-n (1, $*lv_{Gi}$, $*ld_{Gi}$, $*lcoef_{Gi}$).

• Une troisième propriété (P0'') : si deux cours se chevauchent dans le temps, ils ne doivent pas avoir lieu dans la même salle.

(P0'') : $\forall$ Fk $\in \mathcal{F}$, $\forall$ Fk' $\in \mathcal{F}$, (k≠k'),
(Interv$(g(Fk)_{|\mathcal{H}}$, dk) $\cap$ Interv$(g(Fk')_{|\mathcal{H}}$, dk') $\neq \emptyset$) $\Rightarrow$ $(g(Fk)_{|\mathcal{S}} \neq g(Fk')_{|\mathcal{S}})$

N.B. : *Cette propriété correspond à la propriété A du théorème 1. Nous allons voir plus loin que la propriété B est exprimée principalement par une contrainte de limitation.*

▲ La recherche de $g_{|\mathcal{S}}$ passe par celle de γ $(g_{|\mathcal{H}})$ avec la propriété B. Avec le théorème

-1, nous savons que si les cours sont placés dans le temps en respect de B, il existe une fonction $g_{|\mathcal{S}}$ telle que la propriété ci-dessus soit respectée.

Pour faciliter la recherche de cette fonction $g_{|\mathcal{S}}$, nous allons poser les clauses suivantes. Elles expriment d'une manière simplifiée la propriété A du théorème 1.

Nous posons les clauses suivantes sur les couples de cours (Fk, Fk') de même groupe de salles. Ils ont les variables horaires $*v_{\mathcal{H}k}$ et $*v_{\mathcal{H}k'}$, les variables de salles $*v_{\mathcal{S}k}$ et $*v_{\mathcal{S}k'}$ et les durées respectives *dk et *dk'. Les deux premières clauses expriment que le cours Fk' est placé intégralement, soit avant, soit après Fk. Si ces deux clauses échouent, les cours se chevauchent dans le temps. Dans ce cas, ils ont des salles différentes.

Propriété_P0_seconde ($*v_{\mathcal{H}k}$, $*v_{\mathcal{S}k}$, *dk, *dk', $*v_{\mathcal{H}k'}$, $*v_{\mathcal{S}k'}$) :-
    ^<= ((+ $*v_{\mathcal{H}k}$ *dk), $*v_{\mathcal{H}k'}$).
Propriété_P0_seconde ($*v_{\mathcal{H}k}$, $*v_{\mathcal{S}k}$, *dk, *dk', $*v_{\mathcal{H}k'}$, $*v_{\mathcal{S}k'}$) :-
    ^<= ((+ $*v_{\mathcal{H}k'}$ *dk'), $*v_{\mathcal{H}k}$).
Propriété_P0_seconde ($*v_{\mathcal{H}k}$, $*v_{\mathcal{S}k}$, *dk, *dk', $*v_{\mathcal{H}k'}$, $*v_{\mathcal{S}k'}$) :-
    ^diff ($*v_{\mathcal{S}k}$, $*v_{\mathcal{S}k'}$).

Cette contrainte n'est placée qu'après les affectations des cours dans le temps (si elles existent). Les variables horaires sont donc connues et instanciées.

• Quatrième propriété : (P0"bis) : la salle attribuée à un cours Fk doit appartenir à son groupe de salles.

(P0"bis) : $\forall$ Fk $\in \mathcal{F}$, $Gs_k \in$ Fk, $Gs_k \in E_{Gs}$, $g(Fk)_{|\mathcal{S}} \in Gs_k$.

Nous utilisons la contrainte d'unification par l'intermédiaire du prédicat "domaine". Affecter deux domaines différents à une même variable revient à attribuer à cette variable l'intersection de ces deux domaines.

Soit le cours Fk. Sa variable de salle est $*v_{\mathcal{S}k}$. Son groupe de salles G_{sk} est égal à la liste $*l_{Gsk}$. Cette liste est un paramètre de la clause "composant" avec G_{sk}. Nous posons :

- domaine ($*v_{\mathcal{S}k}$, $*l_{Gsk}$).

▲ Cette propriété constitue la première aide pour trouver $g_{|\mathcal{S}}$: $g_{|\mathcal{S}}(Fk) \in \theta(P) = G_{sk}$.

• La cinquième propriété indique que deux cours de même professeur ne se déroule pas simultanément.

(P0'") : $\forall$ Fk $\in \mathcal{F}$, $\forall$ Fk' $\in \mathcal{F}$, (k≠k'), $\{P_k\} \in$ Fk, $\{P_{k'}\} \in$ Fk',
$(\{P_k\} \cap \{P_{k'}\} \neq \emptyset) \Rightarrow$ Interv $(g(Fk)_{|\mathcal{H}}, dk) \cap$ Interv $(g(Fk')_{|\mathcal{H}}, dk') = \emptyset$

Nous sélectionnons tous les cours de chaque professeur Pu. Nous posons la contrainte "^pas-plus-de-n" sur ceux-ci. Nous avons :

-$*lv_{Pu}$, la liste des variables horaires $*v_{\mathcal{H}k}$ des cours Fk de Pu,

- $*ld_{Pu}$, la liste des durées de ces cours,

- $*lcoef_{Pu}$, la liste de coefficients tous égaux à 1,

- la limite supérieure *n* égale à 1,

et la contrainte : ^pas-plus-de-n (1, $*lv_{Pu}$, $*ld_{Pu}$, $*lcoef_{Pu}$).

• **Contrainte de préaffectation**

Un cours spécifique Fk ne se déroule que dans une après-midi particulière $I_{t,2}$.

(P1) : Pour Fk $\in \mathcal{F}$, Fk choisi, Interv $(g(Fk)_{|\mathcal{H}}, dk) \subseteq I_{t,2}$

Nous connaissons l'après-midi par son début $*v_{\mathcal{H}t2}$ et sa durée *dt2 (dans notre cas 8 demi-heures).

Des inéquations contraignent le cours Fk à se restreindre aux horaires de cet après-midi. Ainsi, nous posons :

- $*v_{\mathcal{H}k}$ est la variable horaire du cours Fk,
- *dk, sa durée,
- $*v_{\mathcal{H}t2}$ le début de l'après-midi concerné,
- *dt2, sa durée,

et les contraintes en conjonction, réunies dans la clause suivante :

```
Propriété_3_prime  (*v_Hk, *dk, *v_Ht2,*dt2) :-
        ^<= (*v_Ht2, *v_Hk),
        ^<= ( (+ *v_Hk *dk), (+ *v_Ht2 *dt2)).
```

Ainsi, le cours commence au plus tôt au début de l'après-midi (première contrainte). Il finit au plus tard à la fin de l'après-midi (seconde contrainte).

• **Contrainte d'indisponibilité**

Aucun cours ne doit chevaucher les récréations. Nous créons des cours fictifs associés aux récréations. Leur durée est nulle. Pour chaque cours Fk, nous posons une disjonction entre chaque cours réel et ces cours fictifs. Ceci se fait par la contrainte "^pas-plus-de-n".

(P2) : $\forall$ Fk $\in \mathcal{F}$, Interv $(g(Fk)_{|\mathcal{H}}, dk) \cap$ Récréation = $\emptyset$.

Nous avons donc :

- $*lv_{kr}$, une liste composée de la variable $*v_{\mathcal{H}k}$ et des heures correspondant aux récréations,

- $*ld_{kr}$, une liste dont le premier élément est la durée dk de Fk, et autant de 0 que de récréations,

- *lcoef$_{kr}$, une liste de 1,

- une limite supérieure n égale à 1,

et la contrainte : ^pas-plus-de-n (1, *lv$_{kr}$, *ld$_{kr}$, *lcoef$_{kr}$).

• **Contrainte d'équilibrage**

Pour chaque groupe d'élèves, les cours de discipline légère sont bien équilibrés.

Rappelons qu'il y a w journées, donc 2 w-1 demi-journées.

(P3) : $\forall$ Gé$_u \in E_{Gé}$, $\forall$ $I_{t,i} \in I$,

$$J1 = \{Fk \in \mathcal{F},\ Gé_u \in Fk,\ disck \in E_{Dislégère},\ Interv\ (g(Fk)_{|\mathcal{H}},\ dk) \subseteq I_{t,i}\},$$

$$J2 = \{Fk \in \mathcal{F},\ Gé_u \in Fk,\ disck \in E_{Dislégère}\},$$

$$[Card(J2) \div (2\ w\text{-}1)] \leq Card(J1) \leq [Card(J2) \div (2\ w\text{-}1)]^*$$

La propriété exprime le fait que nous devons placer en moyenne, autant de cours dits "légers" dans chaque demi-journée.

Remarquons que ce qui nous intéresse n'est pas la demi-journée mais le début de chaque cours.

L'équilibrage suppose une limitation supérieure du nombre de cours par demi-journée. Ceci peut s'exprimer par la contrainte "^pas-plus-de-n".

L'équilibrage suppose aussi une limitation inférieure du nombre de cours par demi-journée. Ceci peut s'exprimer par la contrainte "^pas-moins-de-n".

Les deux contraintes nous expriment le système d'inéquations de (P3). Elles portent conjointement sur les variables de demi-journée.

Nous utilisons la contrainte ^élément qui relie chaque variable de début de cours à son vis-à-vis, variable de demi-journée (cf au début de cette partie iii).

Nous sélectionnons la liste des cours de discipline légère de chaque groupe d'élèves Gi. Puis nous calculons le rapport du nombre de cours sur les 2 w-1 demi-journées. Les moyennes MD et ME représentent la partie entière par défaut et par excès de ce rapport.

Nous posons alors :

- *v$_{\mathcal{H}k}$, les variables horaires des cours Fk,
- *lv$_{gilégère}$, la liste des variables de demi-journées *v$_{\mathcal{H}itk}$ des cours du groupe Gi de discipline légère,
- *ld$_{gilégère}$, la liste de durées des demi-journées (8 demi-heures),
- *lcoef$_{gilégère}$, une liste de 1,

- *MD, la moyenne par défaut,
- *ME, la moyenne par excès,

et conjointement avec les contraintes ^élément ($*v_{\mathcal{H}k}$, *DJ, $*v_{\mathcal{H}itk}$), les contraintes par la clause suivante :

Propriété_3 ($*lv_{gilégère}$, $*ld_{gilégère}$, $*lcoef_{gilégère}$, *MD, *ME) :-
^pas-plus-de-n (*ME, $*lv_{gilégère}$, $*ld_{gilégère}$, $*lcoef_{gilégère}$),
^pas-moins-de-n(*MD, $*lv_{gilégère}$, $*ld_{gilégère}$, $*lcoef_{gilégère}$).

• **Contrainte de compacité**

Le placement de tous les cours d'un professeur choisi est le plus compact possible. Nous limitons le temps (ou trou) entre deux cours consécutifs.

Dans la propriété, la durée du trou autorisé est nommé *dtrou*.

(P5''') : Pour $P_u \in E_P$, P_u choisi,
soit Max le dernier de ses cours :
Max = F_{max} tel que Maximum $\{g(Fk)_{|\mathcal{H}}, P_u \in Fk\}$,

$\forall$ Fk $\in \mathcal{F}$, Fk $\neq$ Max, $P_u \in$ Fk, alors $\exists$ Fv $\in \mathcal{F}$, $P_u \in$ Fv tel que :
dk $\leq g(Fv)_{|\mathcal{H}} - g(Fk)_{|\mathcal{H}} \leq$ dk + dtrou

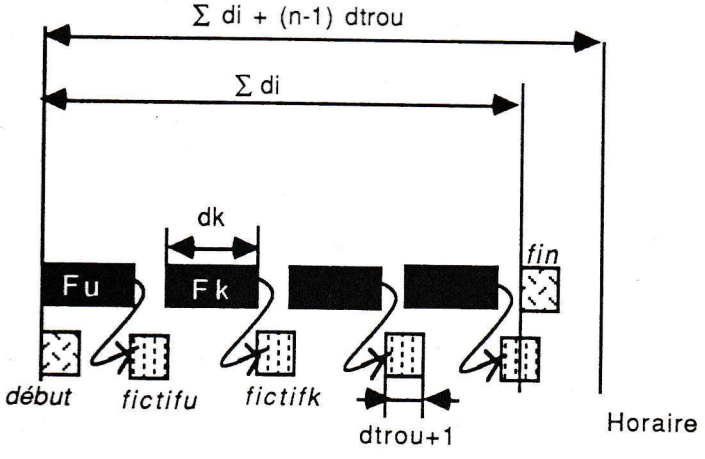


Figure 23. Compacité

Un trou sera schématisé par un cours fictif.

Deux cours fictifs particuliers expriment le début et la fin de tous les cours. Leurs variables horaires sont *début et *fin.

Nous créons, ensuite, autant de cours fictifs que de cours réels impliqués. Nous

-associons à chaque cours Fk de durée dk, un cours fictif *fictifk* de durée d*fictifk*.

Chaque début ($*v_{\mathcal{H}fictifk}$) est exactement la fin du cours Fk associé par une contrainte d'égalité ($*v_{\mathcal{H}fictifk} = *v_{\mathcal{H}k}$ + dk). Leur durée *d*fictifk* correspond à la durée du trou (+ 1) : *d*fictifk* = dtrou+1.

Au moins un cours Fk' doit chevaucher ce cours fictif $fictif_k$. Ainsi un cours Fk' doit commencer, au pire, dtrou après la fin de Fk. Ceci doit être garanti par la contrainte "^pas-moins-de-n".

Rappelons que nous avons exprimé auparavant la contrainte de disjonction entre les cours de ce professeur.

Pour plus d'efficacité, nous posons les contraintes redondantes que chaque cours est compris entre le début et la fin de tous les cours. Ceci est exprimé par les contraintes "<=".

De plus, la distance entre le début et la fin de tous les cours doit être comprise entre deux limites. La première somme les durées des cours réels. S'il y a n cours, la seconde est la même additionnée de (n-1) fois la durée des trous.

Pour les cours Fk du professeur Pu, nous avons :

- $*v_{\mathcal{H}k}$ variable horaire du cours Fk et *dk, sa durée,
- $*v_{\mathcal{H}fictifk}$ variable fictive associée à Fk avec *dfictifk sa durée,
- $*lv_{Pu}$, la liste des variables pour les cours (fictifs et réels) de Pu,
- $*ld_{Pu}$, la liste des durées respectives,
- $*lcoef_{Pu}$, une liste d'autant de 1 que de variables dans $*lv_{Pu}$,
- *début, la variable horaire du cours fictif "début de tous les cours" de durée 1,
- *fin, la variable horaire du cours fictif "fin de tous les cours" de durée 1,
- la constante SDmin, la somme des durées *dk,
- la constante SDmax, la somme des durées *dk plus (n -1) fois dtrou,

avec les contraintes entre le début, la fin et chaque cours réel Fk de durée *dk :

- ^= ($*v_{\mathcal{H}fictifk}$ (+ $*v_{\mathcal{H}k}$ *dk)),
- ^= (*dfictifk, (+ *dtrou 1)),
- ^<= (*début, $*v_{\mathcal{H}k}$),
- ^<= ((+ $*v_{\mathcal{H}k}$ *dk), *fin),
- ^<= (SDmin, (- *fin *début)) et
- ^<= ((- *fin, *début), SDmax).

les domaines des variables *début et *fin sont toutes les heures possibles de $\mathcal{H}$,

avec la contrainte entre les cours réels et les cours fictifs associés :

- ^pas-moins-de-n (2, $*lv_{Pu}$, $*ld_{Pu}$, $*lcoef_{Pu}$)

• **Contrainte de limitation**

Pour chaque groupe de salles, le nombre de cours se chevauchant dans le temps ne dépasse pas le nombre de salles de ce groupe.

(P6) : $\forall Gs_k \in E_{Gs}$, $\forall Fk \in \mathcal{F}$, $Gs_k \in Fk$,

$J = \{Fu/\ Fu \in \mathcal{F},\ Gs_k \in Fu,$
$\quad Interv\ (g(Fk)_{|\mathcal{H}},\ dk) \cap Interv\ (g(Fu)_{|\mathcal{H}},\ du) \neq \emptyset\}$ alors
$Card(J) \leq Card(Gs_k)$

▲ Nous pouvons nous rendre compte que (P6) n'est autre que la propriété B. L'ensemble J n'est autre qu'un ensemble G (cf C.3.1.d.vi).

Cette propriété se réalise aisément avec la contrainte "^pas-plus-de-n". A chaque groupe de salles Gs_k, nous prenons la liste des cours Fk de ce groupe. Nous posons alors :

- *lvGsk, la liste des variables horaires $*v_{\mathcal{H}k}$ des cours Fk de groupe Gs_k,
- *ldGsk, la liste des durées correspondantes,
- *lcoefGsk, une liste d'autant de constantes 1 que de variable dans *lvGsk,
- Nsk, le nombre de salles du groupe Gs_k,

et la contrainte : ^pas-plus-de-n (Nsk, *lvGsk, *ldGsk, *lcoefGsk)

b) Colorations : Enumérations par étiqueteurs

Nous utilisons les différentes stratégies déjà présentées. Dans notre problème, nous utilisons le théorème 1 défini ci-avant (C.2.1).

• Dans ce cas, la recherche et l'existence de g dépend de l'existence d'un γ respectant la propriété B. Ceci consiste à chercher une place horaire aux cours. Cette particularité va influencer la forme de notre programme.

Le programme consiste en la séquence suivante de contraintes et d'étiqueteurs :

1) Poser toutes les contraintes concernant $\mathcal{H}$. Dans notre cas, toutes les contraintes sont mises en place sauf celles découlant de la propriété (P0''). Soit le prédicat "pose_contraintes_H(*H)" qui pose toutes ces contraintes sur $\mathcal{H}$.

2) Etiquetage des variables horaires $*v_{\mathcal{H}k}$,

3) Poser toutes les contraintes sur $*v_{\mathcal{S}k}$ connaissant les $*v_{\mathcal{H}k}$. Cela consiste à appliquer les clauses Propriété_P0_seconde (P0''). Soit le prédicat "pose_contraintes_H_S(*H, *S)" qui pose toutes ces contraintes.

4) Etiquetage des variables de salles $*v_{\mathcal{S}k}$.

Nous supposons que le prédicat "étiqueter(*V)" étiquette les variables *V suivant la stratégie voulue. Nous avons alors le programme de la forme suivante :

```
Emploi_du_temps (*H, *S) :-
        pose_contraintes_H(*H),
        étiqueter(*H),
        pose_contraintes_H_S(*H, *S),
        étiqueter(*S).
```

Les divers étiquetages peuvent se faire en utilisant les différentes stratégies possibles :

- appliquer la clause "apparition". Les variables sont étiquetées suivant l'ordre d'apparition,

- appliquer la clause "étiqueter-pluscontraint" sur cette même liste,

...

C.4 Synthèse : Résultats et limites

Ce modèle d'application a été testé sur un exemple comportant 381 cours. L'établissement a 25 professeurs, 57 classes (ou groupes indépendants) et 25 salles avec deux groupes. Les cours ont une durée variant d'une heure à trois heures. L'unité est la demi-heure.

En complexité d'écriture nous avons posé environ 600 lignes dont 95 contraintes. Si nous ne posons que des contraintes générales en terme de disjonction entre cours (2 à 2), ceci implique 7210 paires de disjonction. Il y a donc un gain effectif.

Tout d'abord, nous avons posés des contraintes de pré-affectation, de limitation, d'indisponibilité.

L'affectation attribue des demi-heures et des salles. La première dimension correspond aux demi-heures d'une semaine de cinq jours et demi. La seconde dimension est l'ensemble des 25 salles.

L'application a été réalisée à l'aide de deux langages : CLEF v1 et CHIP, sur une SUN-360.

Les premiers essais sont effectués avec CLEF v1. Nous faisons uniquement le placement horaire. La première solution est obtenue en 300 secondes (5 minutes).

En CLEF v1, j'ai été amené à ajouter à l'interpréteur une simulation des contraintes "^pas-plus-de-n" et "^pas-moins-de-n". Le schéma de ces deux contraintes n'étant pas optimisées, elles se réveillent d'une manière intempestive. Dans la nouvelle version, des schémas de déclenchement adéquat ont été ajoutés.

Divers essais ont été faits avec CHIP. Le premier tient compte de propriétés générales, de préaffectation, d'indisponibilité et de limitation. Le placement est celui des demi-heures. Le temps d'exécution est de 4 secondes pour la première solution.

Un deuxième essai diminue la dimension horaire :

- entre zéro jour et quatre jours, aucune solution n'est trouvée. Il a fallu 6 secondes.

- pour cinq jours et cinq jours et demi, il faut 4 secondes pour une première solution.

- à quatre jours et demi, nous avons eu le temps de passer une bonne nuit. Il a fallu interrompre le programme. Malgré ça, nous ne savons pas s'il y a une solution ou non.

Le troisième essai fait une multi-coloration et place les salles et les demi-heures. Le temps d'exécution est de 120 secondes pour la première solution.

C.5 Conclusion

Nous avons montré comment modéliser le problème d'emploi du temps en P.L.C.. Nous avons utilisé une coloration multi-dimensionnelle. Pour chaque propriété, nous avons montré la correspondance avec la P.L.C..

Les apports avec l'application de la P.L.C. sur le problème d'emploi du temps interviennent à plusieurs niveaux :

- cette résolution prend en compte des problèmes plus généraux que les autres résolutions existantes. Les cours ont des durées quelconques. Il existe plusieurs types de contraintes. Les cours ne sont pas limités à deux composants, ...

- aucune analyse mathématique n'est nécessaire. Elle est plus intuitive et naturelle.

- le critère est implicitement fait par l'ensemble des contraintes posées.

- la stratégie est simple et générale. Elle permet de prendre en compte les effets des contraintes et même des priorités implicites.

- les changements intempestifs peuvent être partiellement pris en compte par le backtrack.

Cependant, plusieurs inconvénients persistent. Le plus important est d'expliciter les réponses :

- si une ou plusieurs solutions sont trouvées, nous ne savons pas comment.

- soit il n'y a aucune solution, soit le temps de résolution dure très longtemps. L'utilisateur ne sait pas pourquoi. Il n'a aucun moyen d'obtenir une solution même incomplète, c'est-à-dire un point ne satisfaisant pas toutes les contraintes en place.

De plus, les désirs, souhaits ou voeux sont pris en compte au même titre que les propriétés absolues.

Un autre inconvénient est celui des changements intempestifs. Le backtrack permet d'énumérer des points de l'espace de recherche. Mais, déplacer un cours signifie refaire l'ordonnancement des cours placés après lui.

Ceci est un inconvénient sur le temps d'exécution. Le backtrack permet, tout de même, de le faire. Dans ce rapport, le travail qui nous intéresse est celui de l'obtention d'une solution même incomplète.

Un troisième inconvénient est que notre propriété B exprimé par la contrainte de limitation est trop restrictive. Il serait intéressant de continuer ces travaux sur ce sujet. Il faut tenir compte de la remarque sur les cliques, exprimée dans la preuve du théorème 1. Ainsi les fonctions de coloration (autre que la coloration lié au temps) serait optimum.

CHAPITRE D

Relâchement de Contraintes en P.L.C.

D.1 Introduction

Dans le chapitre précédent, nous avons défini un modèle d'application de la P.L.C. au problème d'emploi du temps. Nous avons pu nous apercevoir que la P.L.C. apporte beaucoup. Cependant elle a un inconvénient majeur : l'utilisateur est aveugle vis à vis de la résolution de la P.L.C.. Il ne sait pas comment une solution est obtenue. Il ne sait pas non plus pourquoi certains points sont écartés.

Pour un problème d'emploi du temps, il y a de fortes chances que la résolution par la P.L.C. échoue ou dure trop longtemps. L'utilisateur n'a alors aucune solution.

Ce chapitre intervient à ce niveau. Il faut que l'utilisateur ait, de son point de vue, une "meilleure" solution. Celle-ci peut ne pas satisfaire toutes les contraintes. Elle est, dans ce sens, incomplète. Cependant, elle doit se rapprocher le plus possible du désir du concepteur de l'emploi du temps. La solution qui se rapproche alors le plus est nommée "meilleure" solution.

Pour rechercher une solution même incomplète, il faut pouvoir relâcher certaines contraintes indiquées par l'utilisateur. Puis il faut comparer les solutions incomplètes afin de trouver la "meilleure".

Cependant, plusieurs problèmes se posent lors de relâchement en P.L.C. avec des langages tels que CLEF v1 et CHIP. Nous allons survoler brièvement ces problèmes afin de justifier ce chapitre.

Nous voulons chercher des meilleures solutions. Pour un problème d'emploi du temps avec la P.L.C., cela signifie poser un ensemble de contraintes et chercher une solution. Dans cet ensemble, certaines contraintes sont moins importantes que d'autres. Si aucune configuration de placement de cours ne valide l'ensemble de toutes les contraintes en place, il faut alors analyser toutes les combinaisons de placements et détecter la raison de l'impossibilité de chaque placement. Si l'impossibilité d'un placement provient de contraintes de moindre importance (aux yeux de l'utilisateur), il faut alors la relâcher. Ainsi il faut chercher la configuration qui convient le "mieux" à l'utilisateur. Cette configuration est nommée ici "meilleure" solution.

Il faut donc un mécanisme de relâchement. Illustrons ce mécanisme sur un exemple. Quand nous plaçons un cours Fk, nous réservons un ensemble d'affectations possibles pour Fk. Certains cours ne doivent pas obtenir la possibilité d'avoir ces affectations. Ainsi les contraintes de ces cours ne sont pas testées en ces affectations.

Relâchons une contrainte sur un cours Fk. Cela signifie de rendre certaines places réservées uniquement par Fk à nouveau possibles pour les autres cours. Il faut donc tester les contraintes des autres cours sur ces places dégagées. Relâcher la contrainte sur un cours signifie de la rendre inopérante uniquement vis à vis de Fk, et opérante sur les autres cours qu'elle relie. Il ne faut pas que la contrainte se redéclenche pour le même placement de Fk. Si la contrainte est la cause principale de configurations impossibles, son relâchement implique d'introduire les configurations éliminées devenues possibles.

Ce chapitre se consacre à ces deux mécanismes : relâchement de contraintes et recherche de "meilleures" solutions.

Dans un premier temps, nous critiquons les travaux décrits au chapitre A sur les solutions approchées en P.L.C.. Il s'agit de la modification de requête par l'utilisateur et une recherche automatique de "meilleures" solutions. Ces travaux ne peuvent pas nous servir pour les problèmes d'emploi du temps avec la P.L.C.. C'est pourquoi, nous présentons une autre proposition.

Nous présentons d'abord un modèle pour mémoriser en P.L.C. les points insatisfaits et les contraintes associées. Ce modèle est un hypergraphe nommé hypergraphe de contraintes insatisfaites. A partir de ce dernier, nous sommes capables de détecter des contraintes à relâcher.

Ensuite, nous décrivons comment utiliser cet hypergraphe de contraintes insatisfaites dans des langages de P.L.C.. Ces langages sont conformes à certaines hypothèses (domaines discrets et finis, hiérarchie de contraintes, ...). Nous pouvons alors relâcher des contraintes et chercher des meilleures solutions.

Puis nous montrons comment implanter ce modèle de relâchement et de recherche de meilleures solutions à partir du langage CLEF v1. Ceci permet de montrer ce qu'il faut ajouter à tout langage de P.L.C. intégrant des domaines finis et discrets pour obtenir ces mécanismes.

Enfin, un exemple illustratif d'utilisation est décrit à l'aide de CLEF v1.

D.2 Critique des méthodes existantes

Cette division rappelle et critique les travaux présentés en B.3.2 : la méthode de requête par réexécution ou par ajout intelligent et la recherche de "meilleures" solutions. Les principes de ces travaux sont rappelés. Puis chacun des ces principes est critiqué. Nous déduisons que ces principes ne conviennent pas pour le langage CLEF v1 appliqué aux problèmes d'emploi du temps.

Nous terminons cette partie en présentant les "points épineux" du relâchement de contraintes et de la recherche de "meilleures" solutions pour la P.L.C..

D.2.1 Critiques du principe de modification de requête

Rappel

Actuellement, il existe deux principes : modifier une requête par réexécution ou par ajout intelligent (cf le chapitre B).

⇨ Rappelons le principe de modification de requête par réexécution. Celui-ci consiste à modifier des problèmes Pi en ajoutant ou en supprimant une contrainte.

Le problème initial est constitué d'une requête R_0 et d'un programme P. Pas à pas, le problème final est construit de telle sorte que les solutions finales satisfassent l'utilisateur.

A chaque problème Pi, les chemins (ou Oracle Oi) menant à une solution sont mémorisés.

Un pas est réalisé en additionnant ou supprimant une contrainte.

L'ajout d'une contrainte λ au problème Pi nous donne le problème Pi+1. Nous avons alors quatre phases :

- appel des chemins menant aux solutions de Pi,
- ajout de λ,
- recherche des solutions avec λ à partir des solutions de Pi,
- mémorisation de ces chemins.

La suppression d'une contrainte λ au problème Pi nous donne le problème Pi+1 et passe par trois phases :

- recherche du problème Pk tel que $\lambda \in$ Pk+1 et $\lambda \notin$ Pk,
- appel des chemins menant aux solutions de Pk,
- continuation de la résolution sans λ.

⇨ Rappelons le principe de modification de requête par ajout intelligent. Celui-ci permet d'ajouter un incrément I en cours de résolution entre deux prédicats P et Q.

Si Q' dérive de Q dans une alternative, il suffit de rajouter l'incrément I après Q'. En cas de backtrack sur une autre alternative Q" de Q, il faut appliquer I avant Q".

Critiques

Les critiques s'appliquent à un langage tel que CLEF v1 et à des problèmes d'emploi du temps. Nous pouvons dégager trois principales critiques à ces principes.

La première critique est le principe de modification de requête revient à faire du tâtonnement. L'utilisateur doit savoir quand et comment agissent ses contraintes. Or nous ne connaissons pas à quel moment les contraintes se déclenchent dans des langages tels que CLEF v1.

De plus, il existe une différence entre la pose d'une contrainte (prise en compte de son existence) et le moment où elle se joue un rôle effectif (par ses différents déclenchements). Ces déclenchements ne s'effectuent qu'en concordance avec des schémas précis. Une contrainte peut avoir été posée et jamais déclenchée.

Dans le principe de modification de requête, c'est l'utilisateur qui indique les contraintes à supprimer. Malheureusement, l'utilisateur ne connaît pas toujours les contraintes qui bloquent effectivement la dérivation. Il doit agir en aveugle.

La deuxième critique concerne l'ajout de contraintes. Avec le principe énoncé ci-dessus, nous rajoutons une contrainte et nous cherchons la faisabilité du nouveau problème. Or le problème de l'emploi du temps est par nature NP-Complet. Chercher une solution peut être extrémement long.

La troisième critique concerne le relâchement de contraintes. Suivant le lieu de pose dans le programme de la contrainte à relâcher, le relâchement provoque une résolution d'un problème plus ou moins déjà résolu. Si la contrainte à relâcher est la première posée, le relâchement peut provoquer une résolution du problème à partir de zéro. Si c'est la denière contrainte posée qui est à relâcher, il suffit de revenir au problème précédent. La complexité de résolution avec relâchement n'est donc pas linéaire en fonction du lieu de pose des contraintes.

De plus, les contraintes sont des relations n-aires. Le relâchement d'une contrainte porte donc sur n variables. Ceci signifie l'élimination de la relation sur ces n variables dans son passé mais aussi dans son futur.

Exemple : soit la contrainte "^pas-plus-de-n" de CLEF v1 exprimant une propriété générale. Elle stipule que le professeur Durand a tous ses cours à des horaires différents et qu'ils sont tous différents de cours fictifs de fermeture. Supposons qu'un cours de ce professeur ne peut pas se placer sur une heure de fermeture à cause de la contrainte ci-dessus et admettons qu'il reste des cours à placer. Le relâchement de cette contrainte implique que le cours bloqué peut être placé à cette heure de fermeture. Cela signifie aussi que les cours non encore placés pourront se chevaucher. Or, ce n'est pas l'effet voulu.

D.2.2 Critiques du principe de recherche de "meilleures" solutions

Rappel

Ce principe introduit différentes notions : celles de mesure et de hiérarchie de contraintes.

A un but G_i, nous appliquons une mesure $m(G_i)$ qui donne une **note** M appartenant à un ensemble MV. Ces mesures peuvent être monotone, (G_i -> ... $G_{i+n} \Rightarrow m(G_{i+n}) \leq m(G_i)$) et élagante ($\forall$ G_i, G_i -> $G_{i+1} \Rightarrow m(G_{i+1}) \leq m(G_i)$). Si nous ne considérons que les contraintes comme termes à interprétation sur $\mathcal{D}$, la mesure est basée sur les contraintes.

Une hiérarchie de contraintes est un multi-ensemble de contraintes étiquetées d'un poids. Ces poids forment un ordre total entre 0 et N.

Les contraintes de niveau 0 sont absolues. Elles doivent impérativement être respectées. Les autres contraintes sont falcultatives. Une contrainte de niveau i est préférable à toutes celles de niveau i+1.

Une solution réalisable est un point de l'espace de recherche satisfaisant toutes les contraintes absolues (C_0). Le but est de chercher à l'aide de mesures, les meilleures solutions réalisables.

Une pré-mesure g est une fonction de correspondance. Elle prend en paramètre une solution réalisable θ et un ensemble K de contraintes. Elle fournit une valeur d'un multi-ensemble MV de notes : $g(\theta , K) = mv \in MV$.

Soit α et β, deux solutions réalisables. Soit g_1, ..., g_N, les pré-mesures de multi-ensembles MV_1, ..., MV_N alors α est **meilleur** que β si :

$$<g_1(\alpha, C_1), ..., g_N(\alpha, C_N)> \geq <g_1(\beta, C_1), ..., g_N(\beta, C_N)>$$

où pour $\geq$, nous prenons l'ordre lexico-graphique de $MV_1 \times ... \times MV_N$.

Le but est de chercher les solutions réalisables maximales sous cet ordre.

Supposons que le multi-ensemble de notes MV est l'ensemble, ordonné par inclusion, de tous les ensembles de contraintes secondaires. Soit $g_i(\theta , C_i) = \{\lambda \in C_i, \lambda$ est satisfait en $\theta\}$, nous obtenons la notion de **comparateur "locally predicate better"**. Les "meilleures" solutions sont celles qui satisfont le plus de contraintes prioritaires.

Le principe consiste à développer toutes les combinaisons de contraintes. Puis nous évaluons les "meilleures" solutions issues de chaque combinaison. Le test peut être optimisé. Il n'est pas toujours nécessaire de développer entièrement une combinaison. Dans certains cas, la pose de quelques contraintes d'une combinaison suffit à déclarer moins bonne les solutions de cette combinaison.

Critiques

La première critique provient du nombre de combinaisons possibles.

Admettons que nous ayons N blocs de contraintes. Ils correspondent aux niveaux de hiérarchie. Pour chaque bloc, il existe deux possibilités : ce bloc est posé ou non.

Nous avons 2^N combinaisons à tester dans le cas général. Nous avons donc autant de mesures à faire.

Exemple : En CLEF v1, nous pouvons avoir une clause "pose_contrainte" dont la définition est la suivante :

```
pose_contrainte (*niveau, *bloc_contraintes) :- *bloc_contraintes.
pose_contrainte (*niveau, *bloc_contraintes).
```

La complexité de la résolution est multipliée par un facteur non négligeable (2^N).

Dans certains cas de mesures et de comparateurs, ceci peut paraître plus simple. C'est le cas du comparateur "locally predicate better" (cf B.3.2 b). Cependant, la complexité est toujours multipliée par un facteur non négligeable.

Exemple : Pour notre problème d'emploi du temps avec le comparateur "locally predicate better", nous pouvons ordonner la pose des blocs de contraintes suivant leur niveau.

Ainsi les prédicats "pose_contrainte" sont posés suivant l'ordre du bloc le plus prioritaire au moins prioritaire. La première solution suivant le comparateur "locally predicate better" est forcément la "meilleure".

Le backtrack relâche le dernier bloc posé. C'est le moins prioritaire. Le premier bloc de contraintes relâchées est donc le moins prioritaire parmi les blocs posés.

Relâcher la pose d'une contrainte dans le programme ne veut rien dire dans le cas du langage CLEF v1. Une contrainte posée ne signifie pas qu'elle est déclenchée.

Nous pouvons forcer ce déclenchement mais cela reviendrait à faire du "générer & tester". C'est donc une résolution peu efficace.

Un autre inconvénient vient du fait que nous travaillons sur un ensemble de contraintes connues à l'avance. Il faut pouvoir ajouter une contrainte ou en supprimer quand nous le voulons. Or, avec ce principe, nous ne pouvons pas le faire.

Nous voulons aussi avoir la possibilité de changer de hiérarchie. Ceci signifie changer l'importance entre contraintes secondaires.

Dans ce cas, il faut refaire la recherche de "meilleures" solutions à partir de zéro.

Un autre inconvénient provient de l'aspect intégral du relâchement. La contrainte est intégralement relâchée. La relation est donc intégralement enlevée.

D.2.3 Points épineux pour la P.L.C.

Ainsi pour le relâchement en P.L.C., nous avons les points épineux suivants :

- Dans une même dérivation, les déclenchements des contraintes forment un ordre. Une première contrainte π peut se déclencher. Elle élimine les points W. Les contraintes Q déclenchées après cette première contrainte π ne sont pas appliquées sur les points de W.

Un relâchement de la contrainte π implique de réintroduire les points de W, éliminés et appliquer à nouveau les contraintes Q sur ces points éliminés (W) par π.

- Le relâchement de contraintes peut impliquer la découverte de points particuliers dans certaines dérivations. Ces points ont la particularité d'insatisfaire uniquement ces contraintes relâchées et de satisfaire toutes les autres contraintes. Ainsi les dérivations qui sont dans ce cas, doivent être remise en cause.

- Une contrainte est une relation entière que l'on peut décomposer en plusieurs sous-relations. Relâcher une contrainte P.L.C. exprime le relâchement de la relation entière. Nous devons aussi pouvoir signifier le relâchement d'une de ses sous-relations. C'est-à-dire relâcher seulement une partie de cette contrainte P.L.C..

Exemple : La contrainte "^pas-plus-de-n" dans CLEF v1 exprime une relation entre cours. Nous voulons relâcher cette relation sur un cours Fu (une sous-relation vis-à-vis de Fu). La relation doit être maintenue entre les autres cours existants.

- En relâchant une partie d'une contrainte P.L.C., nous relâchons une des sous-relations la constituant. Cela ne signifie pas que la contrainte P.L.C. soit entièrement éliminée. Son comportement est uniquement diminué.

Dans ce cas, il faut que la contrainte P.L.C. inhibe dans le futur de la dérivation le comportement caractéristique de cette sous-relation.

En d'autres termes, associons une sous-relation à un déclenchement d'une contrainte dans une dérivation. Relâcher cette sous-relation implique que la contrainte P.L.C. ne doit plus refaire ce même déclenchement dans le futur de la dérivation. Chaque déclenchement se caractérise par une raison. Il faut donc que les raisons de déclenchements futurs de la contrainte P.L.C. soient différentes de la raison de déclenchement de la sous-relation relâchée.

- Le mécanisme de relâchement ne doit pas augmenter considérablement la complexité de résolution du problème en soi. Résolvons un problème Pi avec un ensemble C de contraintes. Puis relâchons un ensemble de contraintes π. La complexité pour obtenir des solutions avec ce processus doit être identique à la complexité de la résolution d'un problème Pj avec un ensemble $C \backslash \pi$ de contraintes.

Maintenant passons aux "points épineux" pour la P.L.C. et tout mécanisme de recherche de "meilleures" solutions :

- La résolution d'un problème en P.L.C. fournit un ensemble particulier de points. Dans cet ensemble, certains points satisfont toutes les contraintes en place. D'autres points ne satisfont que quelques contraintes. Nous voulons évaluer et comparer ces points en fonction des contraintes qu'ils satisfont.

Cependant, actuellement, nous ne savons pas quelles sont les contraintes satisfaites et insatisfaites par les divers points.

- Supposons que le relâchement existe et qu'il est sûr. La recherche de "meilleures" solutions approchées implique le relâchement d'un ensemble particulier de contraintes. Il faut pouvoir détecter ces contraintes telles que ce relâchement libère effectivement un ensemble de points. Ces contraintes seront nommées plus loin, contraintes de relâchement.

Actuellement, en P.L.C., nous n'avons pas cette possiblité. Lors de l'échec d'une dérivation, nous ne savons pas quel est l'ensemble minimum de contraintes à relâcher pour éviter l'échec.

Une dérivation en échec peut provoquer le relâchement d'un ensemble de contraintes. Pour obtenir un succès dans une dérivation précise, il faut parfois relâcher progressivement des ensembles de contraintes.

Le déroulement d'une dérivation se fait pas à pas. A un pas précis, l'ensemble des contraintes est insatisfait. Il faut analyser quelles sont les contraintes à relâcher pour pouvoir passer au pas suivant. Ainsi avant que la dérivation se termine par un succès, il faut avoir eu plusieurs relâchements à différents pas.

Quand nous allons évaluer les solutions de chaque dérivation, il faut tenir compte de tous les relâchements successifs qui ont été effectués sur chaque dérivation.

D.2.4 Conclusion

Pour le problème d'emploi du temps résolu avec un langage P.L.C., les travaux existants ne suffisent pas pour donner des "meilleures" solutions approchées.

Pour ces problèmes, il faut disposer des mécanismes de relâchement et de recherche de "meilleures" solutions permettant de :

- relâcher des contraintes sans altérer la complexité,

- changer la hiérarchie entre contraintes secondaires. Dans le problème d'emploi du temps, cela permet de changer l'importance de certaines propriétés,

- relâcher une contrainte dans une partie de ces effets passés. Dans le problème d'emploi du temps, une contrainte doit pouvoir être relâchée sur un unique cours. Tous les autres cours restent relier par cette contrainte.

- obtenir une meilleure solution à l'aide de différents comparateurs et mesures. La résolution peut être guidée. Les solutions peuvent être filtrées par des mesures particulières.

- contrôler les relâchements et se rendre compte des effets des contraintes sur la résolution. Ainsi le concepteur n'est plus aveugle.

D.3 Caractérisation de contraintes à relâcher

Nous avons introduit la notion d'hypergraphes valides dans le chapitre C. Ces hypergraphes représentent des combinaisons que valide chaque contrainte (ou propriété) posée.

Nous allons mettre en place un modèle d'hypergraphe qui représente les combinaisons rejetés. Ainsi ces hypergraphes représentent les complémentaires des hypergraphes valides dans l'hypergraphe de toutes les combinaisons possibles. Ces hypergraphes sont nommés Hypergraphe de Contraintes Insatisfaites (HCI).

Cette division D.3 présente une terminologie. Ceci nous permet de définir clairement un Hypergraphe de Contraintes Insatisfaites et certains cas particuliers d'HCI.

Dans le cas où toutes les combinaisons possibles sont rejetées, nous prouvons comment trouver les contraintes nécessaires à relâcher pour libérer une combinaison.

Rappel :

Un ensemble de rejet d'une contrainte C sur S est noté $S^{C'}$ et est tel que :

$\forall s \in S^{C'}$, s ne satisfait pas la contrainte C.

Un ensemble d'agrément de C sur S est noté S^{C} et est tel que :

$\forall s \in S^{C}$, s satisfait la contrainte C.

D_i^C est l'ensemble restreint de D_i. Nous introduirons, $D_i^{C'}$ son complémentaire.

D.3.1 Définitions et hypothèse générales

Cette sous-division présente plusieurs définitions que nous utiliserons par la suite. Effectivement, nous allons les utiliser non seulement pour la division D.3 mais aussi pour décrire les mécanismes de relâchement et de "meilleures" solutions en P.L.C..

a) Les solutions exactes, approchées, réalisables et partielles

Dans la P.L.C. appliqué au problème d'emploi du temps, il est nécessaire de différencier les combinaisons obtenues. Pour réaliser cette différence, nous supposons que nous avons une hiérarchie de contraintes. Dans cette proposition, nous prenons une hiérarchie à deux niveaux : absolue et secondaire. Seules les contraintes secondaires peuvent être relâchées.

Ainsi les combinaisons satisfaisant toutes les contraintes sont nommées solutions exactes. Les combinaisons ne satisfaisant que les contraintes absolues sont nommées solutions réalisables. Les solutions réalisables qui ne sont pas exactes, sont nommées approchées. Si toutes les contraintes ne sont pas placées, toute solution est partielle.

Définition 1 : Soit S un espace de recherche, $S \neq \emptyset$, $\mathcal{C}$ un ensemble hiérarchisé de contraintes. Soit $C_0 \in \mathcal{C}$, l'ensemble des contraintes absolues et $C_i \in \mathcal{C}$ ($i > 0$), l'ensemble des contraintes secondaires. Les points de S satisfaisant C_0 (S^{C_0}) sont des **solutions réalisables**.

Définition 2 : Soit S un espace de recherche, $\mathcal{C}$ un ensemble hiérarchisé de contraintes. Soit $C_0 \in \mathcal{C}$, l'ensemble des contraintes absolues et $C_i \in \mathcal{C}$ ($i > 0$), l'ensemble des

contraintes secondaires. Les points de S satisfaisant $\mathcal{C}$ ($S^{\mathcal{C}}$) sont des **solutions exactes.**

Définition 3 : Soit S un espace de recherche, $\mathcal{C}$ un ensemble hiérarchisé de contraintes Soit $C_0 \in \mathcal{C}$, l'ensemble des contraintes absolues et $C_i \in \mathcal{C}$ (i > 0), l'ensemble des contraintes secondaires. Les points θ de S satisfaisant C_0 mais pas $\mathcal{C}$ ($\theta \notin S^{\mathcal{C}}, \theta \in S^{C_0}$) sont nommés **solutions approchées.**

Définition 4 : Soit S un espace de recherche, $\mathcal{C}$ un ensemble hiérarchisé de contraintes et K, un sous-ensemble de $\mathcal{C}$. Les points de S satisfaisant K (S^K) sont des **solutions partielles.**

A chaque étape de dérivation G_i=<Wi, :-B..., K>, les points associés aux variables de Wi sont des solutions partielles. Ils satisfont les contraintes en place dans cette étape. Ces dernières ne constituent pas l'ensemble complet des contraintes du problème.

b) Classement de contraintes

Les contraintes de la P.L.C. correspondent à des relations à maintenir entre des variables. Ces contraintes peuvent être classées en deux sortes : élémentaires et compositionnelles.

N.B. : *La notation $\mathcal{C} \setminus \pi$ signifie l'ensemble suivant : $\{x / x \in C, x \notin \pi\}$.*

Définition 5 : Soit S un espace de recherche et $\mathcal{C}$ un ensemble de contraintes tel que $S^{\mathcal{C}} = \emptyset$. Nous appelons **contraintes de relâchement**, un sous-ensemble de contraintes $cr_k \subseteq \mathcal{C}$ tel que $S^{\mathcal{C} \setminus cr_k} \neq \emptyset$.

Définition 6 : Une **contrainte élémentaire** est une contrainte définie par la méthode locale associée à un déclenchement particulier d'une contrainte P.L.C.. et par la raison élémentaire du déclenchement de la contrainte P.L.C.. Ainsi une contrainte élémentaire est caractérisée par un ensemble de tests faits par la méthode à partir de la raison élémentaire de déclenchement.

Exemple : Pour les contraintes CLEF v1, une première raison élémentaire de déclenchement peut être l'instanciation d'une variable. Une deuxième raison peut être la réduction du domaine d'une variable ou encore l'affectation d'un domaine à une variable.

Définition 7 : Une **contrainte compositionnelle** est l'union de contraintes élémentaires.

Toute contrainte de CLEF v1 est une contrainte compositionnelle. A chaque déclenchement, des contraintes élémentaires sont appliquées. Il y a donc autant de contraintes élémentaires que de déclenchements causés par des raisons différentes.

Une contrainte qui est évaluée à nouveau pour une même raison (par exemple, une même instanciation de variable) est vue comme une même contrainte élémentaire.

Si deux contraintes élémentaires font des tests communs, elles sont nommées **dépendantes**. Dans le cas contraire, elles sont **indépendantes**.

Exemple : Soit une contrainte C(x1, ..., xn). Prenons les contraintes élémentaires associées à l'instanciation de chaque variable : C1 pour l'instanciation de x1, ..., Cn pour

celle de xn. Supposons que seule C1 est été appliquée (vis à vis de x1) et qu'elle consiste à faire tous les tests entre x1 et les autres variables. C2 doit être appliqué (vis à vis de x2). Si C2 ne fait pas de test entre x1 et x2, alors C1 et C2 sont indépendantes.

c) Ensemble de caractérisation

Définition 8 : Soit S, un espace de recherche tel que S soit le produit cartésien $D_1 \times ... \times D_n$. Un **ensemble de caractérisation d'insatisfaction** d'une contrainte C dans S est un sous-ensemble $\{xi\} \subseteq D_i$ tel que $D_1 \times ... \times \{xi\} \times ... \times D_n \subseteq S^{C'}$.

Par abus de langage, nous utilisons le terme abrégé : ensemble de caractérisation.

N.B. : *Pour une contrainte C et un ensemble S, il n'existe pas forcément d'ensemble de caractérisation de C. Cependant, il peut en exister plusieurs.*

Exemple : $S = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3)\} = \{1, 2\} \times \{1, 2, 3\} = \{(x, y)\} = D_x \times D_y$

1) $S^{(x=y)'} = \{(1, 2), (1, 3), (2, 1), (2, 3)\}$
$\forall \{xi\} \subseteq D_x, \{xi\} \times D_y \not\subset S^{(x=y)'}$ et
$\forall \{yi\} \subseteq D_y, D_x \times \{yi\} \not\subset S^{(x=y)'}$

Il n'existe pas d'ensemble de caractérisation.

2) $S^{(x<y)'} = \{(1, 1), (2, 1), (2, 2)\}$
$\exists \{1\} \subseteq D_y$ tel que $D_x \times \{1\} \subseteq S^{(x<y)'}$

{1} est un ensemble de caractérisation.

d) Hypothèse générale

Nous allons ici définir une hypothèse. Celle-ci est générale au reste du chapitre.

Hypothèse : Soit S un espace de recherche tel que $S = D_1 \times ... \times D_n$, soit C un ensemble de contraintes élémentaires, alors : $\forall c \in C$, $S^c = E1 \times ... \times En$, avec $Ei \subseteq D_i$, $\forall i \in [1 ..n]$.

Les Ei correspondent aux D_i^c, ensemble restreint de D_i ainsi : $S^c = D_1^c \times ... \times D_n^c$. Nous prenons aussi $S^{c'}$ comme le complémentaire de S^c.

Pour une contrainte $c \in C$, nous notons $D_i^{c'}$, l'ensemble complémentaire de D_i^c :
$D_i^{c'} \cap D_i^c = \emptyset$ et $D_i^{c'} \cup D_i^c = D_i$.

Pour un ensemble de contraintes C : $S^C = \bigcap_{c \in C} S^c$ et $D_i^C = \bigcap_{c \in C} D_i^c$

D.3.2 Hypergraphes de Contraintes Insatisfaites (HCI)

a) Définitions

Nous définissons trois hypergraphes de contraintes insatisfaites dans cette sous-division.

Le premier hypergraphe est celui d'insatisfaction. Ses sommets représentent les points rejetés. Ses hyperarêtes s'associent aux ensembles de rejet.

Dans le deuxième hypergraphe, nous supposons que les points rejetés peuvent s'écrire comme des produits cartésiens d'ensembles dans le cadre où l'hypothèse générale est vraie. Cet hypergraphe sert d'intermédiaire entre le premier et le suivant.

Le troisième hypergraphe est le même que le deuxième. Les hyperarêtes ne portent que sur des ensembles de caractérisation. C'est-à-dire que tous points composés avec des éléments de cet ensemble sont rejetés.

N.B. : Pour des raisons de lisibilité, tout produit cartésien impliquant un singleton sera plutôt écrit sans les accolades entourant l'élément du singleton :

$D_1 \times ... \times \{v\} \times ... \times D_n$ s'écrira $D_1 \times ... \times v \times ... \times D_n$

Définition 9 : Soit S un espace de recherche, C un ensemble de contraintes, un **hypergraphe $\mathcal{H}$ d'insatisfaction** est le couple <E, A> avec :

$E = \{ei / ei=s, s \in S \text{ et } \exists c \in C, ei \in S^{c'}\}$,
$A = \{a / \exists c \in C, a=\{ei, ei \in E\} = S^{c'}\}$.

Exemple : La figure 24 présente un tel hypergraphe. Nous supposons que nous ayons trois cours : F1, F2 et F3. Nous avons l'espace de recherche S = {(F1<-8h, F2<-8h, F3<-9h), (F1<-8h, F2<-9h, F3<-9h), (F1<-9h, F2<-8h, F3<-9h), (F1<-9h, F2<-9h, F3<-9h)}. Nous avons les contraintes suivantes : F1 différent de F3, F2 différent de F3 et F1 indisponible à 8h.

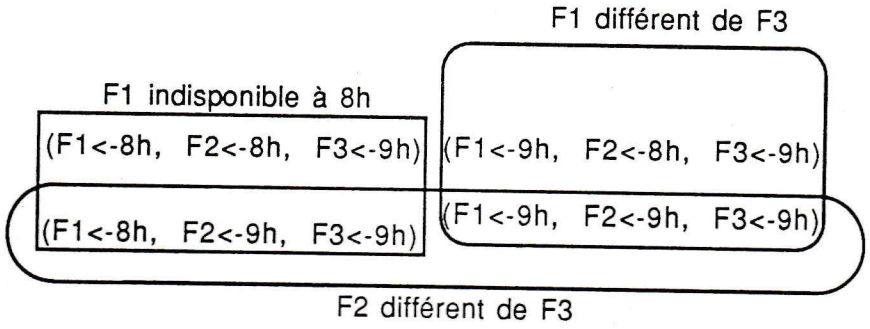


Figure 24. Hypergraphe $\mathcal{H}$ d'insatisfaction

Définition 10 : Soit S un espace de recherche, C un ensemble de contraintes. Si S et C satisfont l'hypothèse générale, alors un **hypergraphe H éclaté d'insatisfaction** est le couple <X, A'> avec :

$X = \bigcup_{1 \le i \le n} Xi$, Xi est un ensemble de plusieurs xi tel que
$Xi = \{xi / xi=\{v \in D_i\}$ et $\forall c \in C, \forall d1 \in D_1, ..., \forall dn \in D_n,$
$d1 \times ... \times xi \times ... \times dn \subseteq S^{c'}$ ou exclusivement
$d1 \times ... \times xi \times ... \times dn \subseteq S^{c}$ et

$\exists q \in C, \exists d1 \in D_1, ..., \exists dn \in D_n,$
$d1 \times ... \times xi \times ... \times dn \subseteq S^{q'}\}$,
tous les xi sont disjoints entre eux.

$$A' = \{a'=\{x1, ..., xn\} / x1 \in X1, ..., xn \in Xn, \exists c \in C, x1 \times ... \times xn \subseteq S^{c'}\}.$$

Nous désignerons par a'^S les points $x1 \times ... \times xn$.

Exemple : Reprenons l'exemple ci-avant et représentons le nouvel hypergraphe par la figure 25. Remarquons que l'ensemble S s'écrit {F1<-8h, F1<-9h} x {F2<-8h, F2<-9h} x {F3<-9h}.

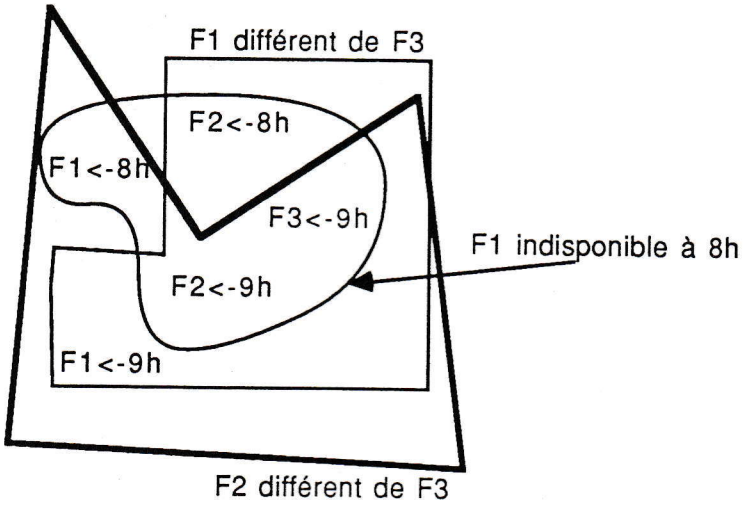


Figure 25. Hypergraphe H éclaté d'insatisfaction

Dans la figure 25, nous avons :

$X1 = \{x_{1,1} = \{F1 <- 8h\}, x_{1,2} = \{F1 <- 9h\}\}$
$X2 = \{x_{2,1} = \{F2 <- 8h\}, x_{2,2} = \{F2 <- 9h\}\}$ et
$X3 = \{x_{3,1} = \{F3 <- 9h\}\}$

Définition 11 : Soit S un espace de recherche, C un ensemble de contraintes. Si S et C satisfont l'hypothèse générale, un **hypergraphe HG de caractérisation** est le couple <X, HA> avec :

$X = \bigcup_{1 \le i \le n} Xi$, Xi est un ensemble de plusieurs xi tel que

$Xi = \{xi/ xi=\{v \in D_i\}$ et $\forall c \in C, \forall d1 \in D_1, ..., \forall dn \in D_n,$
$d1 \times ... \times xi \times ... \times dn \subseteq S^{c'}$ ou exclusivement
$d1 \times ... \times xi \times ... \times dn \subseteq S^{c}$ et

$\exists q \in C, \exists d1 \in D_1, ..., \exists dn \in D_n,$
$d1 \times ... \times xi \times ... \times dn \subseteq S^{q'}\},$

tous les xi sont disjoints entre eux.

$HA = \{ha/ ha=xi / xi \in Xi$ et $\exists c \in C, D_1 \times ... \times xi \times ... \times D_n \subseteq S^{c'}\}.$

Nous désignons par la notation $\mathbf{ha^S}$, l'ensemble de points : $D_1 \times ... \times ha \times ... \times D_n$.

Exemple : Reprenons toujours le même exemple. L'hypergraphe HG est schématisé de la façon suivante :

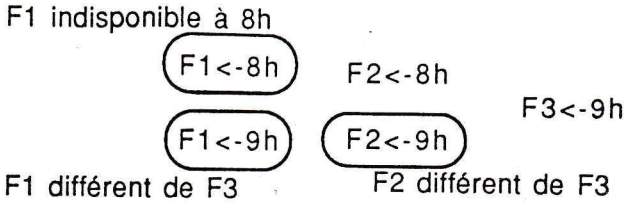


Figure 26. Hypergraphe HG de caractérisation

b) Caractérisation des contraintes insatisfaites

Prenons l'hypergraphe HG. Nous définissons les contraintes insatisfaites sur les points composés avec un des sommets de HG.

Définition 12 : Soit S un espace de recherche, C un ensemble de contraintes. Soit HG, l'hypergraphe de caractérisation avec le couple <X, HA>. Si S et C satisfont l'hypothèse générale, si le sommet e est un ensemble de caractérisation d'insastisfaction alors **mc**(e), e ∈ X, est un ensemble de contraintes insatisfaites aux points composés le sommet e :

$mc(e) \subseteq C$, $e \in X$ et
$mc(e) = \{c \,/\, \exists\, ha \in HA, e \in ha \text{ et } ha^S \subseteq S^{c'}\}$

Exemple : Sur notre exemple, l'ensemble mc(F2<-9h) est {F2 différent de F3}.

D.3.3 Equivalences entre Hypergraphes de Contraintes Insatisfaites

L'hypergraphe $\mathcal{H}$ ci-avant représente un hypergraphe d'invalidité. Cependant, nous ne pouvons pas travailler en P.L.C. directement avec lui. C'est pourquoi, nous avons introduit deux autres hypergraphes. C'est sur le dernier hypergraphe que la P.L.C. travaille. Nous voulons y détecter des contraintes à relâcher en cas de totale réduction de l'espace de recherche S. Il faut donc prouver que ces trois hypergraphes sont équivalents et sous quelle hypothèse. Ceci constitue le but de cette sous-division.

Nous allons d'abord démontrer ou présenter quelques théorèmes simples. Nous allons ensuite prouver l'équivalence de ces trois hypergraphes sous l'hypothèse générale.

a) Théorèmes

☞ Le premier théorème rappelle une formule de Morgan. Celle-ci exprime le complémentaire de l'intersection de deux ensembles.

Théorème 1 : Soit X et Y deux ensembles inclus dans un autre ensemble S. Soit la notation $Compl(A)_S$ pour l'ensemble complémentaire de A dans S. Nous avons alors :

$Compl(X \cap Y)_S = Compl(X)_S \cup Compl(Y)_S$
$Compl(X \cup Y)_S = Compl(X)_S \cap Compl(Y)_S$

☞ Le deuxième théorème exprime que l'intersection de deux produits cartésiens est égale au produit cartésien des intersections de chaque dimension.

Théorème 2 : Soit A, B, C, D quatre ensembles nous avons :

$$(A \times B) \cap (C \times D) = (A \cap C) \times (B \cap D)$$

Ces deux théorèmes ne sont pas prouvés. Le premier est connu et se retrouve dans tout livre concernant les ensembles. L'autre est logique en soi.

☞ Le troisième théorème montre la forme d'un ensemble de rejet d'une contrainte c ($S^{c'}$).

Théorème 3 : Soit S un espace de recherche, C un ensemble de contraintes. Soit D_i^c, ensemble restreint de D_i par c, et $D_i^{c'}$ son complémentaire dans D_i. Si S et C satisfont l'hypothèse générale ($S^c = D_1^c \times ... \times D_n^c$) alors : $S^{c'} = D_1^{c'} \times D_2 \times ... \times D_n \cup ... \cup D_1 \times D_2 \times ... \times D_n^{c'}$.

Preuve :

$\forall c \in C$, $S^{c'}$ est défini par $S^{c'} \cap S^c = \emptyset$ et $S^{c'} \cup S^c = S$.

Donc $S^{c'}$ est le complémentaire de S^c dans S : $S^{c'} = Compl(S^c)_S$

Or $S^c = D_1^c \times ... \times D_n^c \Rightarrow S^{c'} = Compl(D_1^c \times ... \times D_n^c)_S$

et $D_1^c \times ... \times D_n^c = D_1^c \times D_2 \times ... \times D_n \cap ... \cap D_1 \times D_2 \times ... \times D_n^c$ d'après le théorème 2 donc

$S^{c'} = Compl(D_1^c \times D_2 \times ... \times D_n \cap ... \cap D_1 \times D_2 \times ... \times D_n^c)_S$ (théorème 1) nous avons :

$S^{c'} = Compl(D_1^c \times D_2 \times ... \times D_n)_S \cup ... \cup Compl(D_1 \times D_2 \times ... \times D_n^c)_S$.

Or le complémentaire de $D_1^c \times D_2 \times ... \times D_n$ par rapport à S est $D_1^{c'} \times D_2 \times ... \times D_n$

donc $S^{c'} = D_1^{c'} \times ... \times D_n \cup ... \cup D_1 \times ... \times D_n^{c'}$

FIN DE PREUVE

☞ Prenons un produit cartésien de sommets de l'hypergraphe de caractérisation HG. Chaque sommet de ce produit cartésien est le seul à faire parti d'une dimension Xi de X. Le théorème suivant prouve que ce produit cartésien est exclusivement inclus dans un ensemble de rejet ou dans un ensemble d'agrément.

Théorème 4 : Soit S un espace de recherche, C un ensemble de contraintes et X l'ensemble de sommets de l'hypergraphe HG ou de l'hypergraphe H. Si S et C satisfont l'hypothèse générale, alors :

$$\forall x1 \in X1 \subseteq X, ..., \forall xn \in Xn \subseteq X, \forall c \in C, x1 \times ... \times xn \subseteq S^{c'} \text{ ou } S^c$$

Preuve :

Prouvons cette formule par récurrence. La loi de récurrence sur b est :

$\forall x1 \in X1 \subseteq X, ..., \forall xb \in Xb \subseteq X, \forall db+1 \in D_{b+1}, ..., \forall dn \in D_n, \forall c \in C,$

$x1 \times ... \times xb \times db+1 \times ... \times dn \subseteq S^{c'}$ ou S^c

❶ Pour b=1 Démontrons :

$\forall x1 \in X1 \subseteq X, \forall d2 \in D_2, ..., \forall dn \in D_n, \forall c \in C,$
$x1 \times d2 \times ... \times dn \subseteq S^{c'}$ ou S^c

Par construction des xi (Définition 11), cette proposition est vraie.
FIN ❶

❷ Pour b>1 Supposons la loi vraie à l'étape b, montrons qu'elle est vraie à l'étape b+1.

Si cette formule ξ est vraie (étape b) :

(ξ) $\forall x1 \in X1 \subseteq X, ..., \forall xb \in Xb \subseteq X, \forall db+1 \in D_{b+1}, ..., \forall dn \in D_n, \forall c \in C,$
$x1 \times ... \times xb \times db+1 \times ... \times dn \subseteq S^{c'}$ ou S^c

alors celle-ci aussi (étape b+1) :

$\forall x1 \in X1 \subseteq X, ..., \forall xb+1 \in Xb+1 \subseteq X, \forall db+2 \in D_{b+2}, ..., \forall dn \in D_n, \forall c \in C, x1 \times ... \times xb+1 \times db+2 \times ... \times dn \subseteq S^{c'}$ ou S^c

Montrons par l'absurde ce cas, il faut démontrer une incohérence avec l'hypothèse (Φ) suivante :

(Φ) $\exists x1 \in X1 \subseteq X, ..., \exists xb+1 \in Xb+1 \subseteq X, \exists db+2 \in D_{b+2}, ..., \exists dn \in D_n, \exists c \in C,$
$x1 \times ... \times xb+1 \times db+2 \times ... \times dn \not\subset S^{c'}$ et $\not\subset S^c$

Soit $a1 \in X1, ..., ab \in Xb, tb+2 \in D_{b+2}, ..., tn \in D_n$ et $c \in C$ répondant à cette formule. Par hypothèse ξ, la formule est vraie à l'étape b, nous avons pour toutes valeurs de xb+1 :

$a1 \in X1 \subseteq X, ..., \forall db+1 \in xb+1, tb+2 \in D_{b+2}, ..., tn \in D_n, c \in C,$
$a1 \times ... \times db+1 \times tb+2 \times ... \times tn \subseteq S^{c'}$ ou S^c

La seule possibilité pour que l'hypothèse (Φ) soit vraie, est qu'il existe deux valeurs différentes r1 et r2 de xb+1 où les points composés avec r1 sont rejetés et où les points avec r2 sont agréés :

$a1 \in X1 \subseteq X, ..., r1 \in xb+1, tb+2 \in D_{b+2}, ..., tn \in D_n, c \in C,$
$a1 \times ... \times r1 \times tb+2 \times ... \times tn \subseteq S^{c'}$

$a1 \in X1 \subseteq X, ..., r2 \in xb+1, tb+2 \in D_{b+2}, ..., tn \in D_n, c \in C,$
$a1 \times ... \times r2 \times tb+2 \times ... \times tn \subseteq S^c$

Ceci est vrai pour toutes valeurs t1 de a1, ..., tb de ab, nous avons donc pour l'ensemble {r1, r2} :

$\forall t1 \in a1, ..., \{r1, r2\} \subseteq xb+1, tb+2 \in D_{b+2}, ..., tn \in D_n, c \in C,$
$t1 \times ... \times \{r1, r2\} \times tb+2 \times ... \times tn \not\subset S^{c'}$ et $\not\subset S^c$

Donc il en est de même pour xb+1 :

$\forall t1 \in a1, ..., xb+1 \in Xb+1 \subseteq X, tb+2 \in D_{b+2}, ..., tn \in D_n, c \in C,$
$t1 \times ... \times xb+1 \times tb+2 \times ... \times tn \not\subset S^{c'}$ et $\not\subset S^c$

Or, c'est impossible par la construction des xi :

$\forall c \in C, \forall d1 \in D_1, ..., \forall dn \in D_n,$
$d1 \times ... \times xi \times ... \times dn \subseteq S^{c'}$ ou $d1 \times ... \times xi \times ... \times dn \subseteq S^{c}$

Il y a contradiction. La loi est donc vraie à l'étape b+1.
FIN ❷

La loi est vraie à toute étape. Le théorème est donc démontré.
FIN DE PREUVE

b) Equivalences

☞ Si l'hypothèse générale est vérifiée, le théorème prouve l'équivalence entre les trois hypergraphes. La définition de l'équivalence est que ces hypergraphes expriment les mêmes points d'insatisfaction et les mêmes contraintes insatisfaites.

Théorème 5 : Soit S un espace de recherche, C un ensemble de contraintes, S et C satisfaisant l'hypothèse générale, les trois hypergraphes $\mathcal{H}$, H et HG sont équivalents. Ils désignent les mêmes points de S ne satisfaisant pas C : $(\cup_{c \in C} S^{c'})$.

Preuve :

❶ $\mathcal{H}$ est équivalent à H avec $\mathcal{H}$=<E, A> et H =<X, A'>

① Equivalence sur E et X

• E désigne tous les points s de S tel que $\exists c \in C, s \in S^{c'}$
donc $E = \cup_{c \in C} S^{c'} \Rightarrow \cup_{c \in C} S^{c'} \subseteq E$

• Regardons s'il en est de même pour X de H.

① Soit la formule suivante à démontrer:

(Ξ) $\forall q \in C, \forall Xi \subseteq X, S^{q'} \subseteq D_1 \times ... \times \cup_{xi \in Xi} xi \times ... \times D_n$

Montrons sa véracité par l'absurde. Montrons une contradiction avec l'hypothèse :

(Ψ) $\exists q \in C, \exists Xi \subseteq X, S^{q'} \not\subset D_1 \times ... \times \cup_{xi \in Xi} xi \times ... \times D_n$

$\Rightarrow \exists s \in S^{q'}, s \notin D_1 \times ... \times \cup_{xi \in Xi} xi \times ... D_n$

$\Rightarrow \exists v \in D_i, \exists d1 \in D_1, ..., \exists dn \in D_n,$
$s=d1 \times ... \times v \times ... \times dn$ et $s \in S^{q'}$ et $s \notin D_1 \times ... \times \cup_{xi \in Xi} xi \times ... \times D_n$

$\Rightarrow \forall xi \in Xi, v \notin xi$ or la construction de Xi prend tous les xi tel que :
$\exists q \in C, d1 \times ... \times xi \times ... \times dn \subseteq S^{q'}$

En prenant un xi = {v}, nous vérifions que {v} correspond à un xi.

Il y a contradiction. L'hypothèse (Ψ) est donc fausse et la formule (Ξ) est vraie.
FIN ①

Donc $\forall q \in C, S^{q'} \subseteq \bigcup_{x1 \in X1} x1 \times ... \times D_n$ et ... et $D_1 \times ... \times \bigcup_{xn \in Xn} xn$.

Donc c'est aussi vraie pour l'intersection des produits cartésiens :

$$\forall q \in C, S^{q'} \subseteq \bigcup_{x1 \in X1} x1 \times ... \times D_n \cap ... \cap D_1 \times ... \times \bigcup_{xn \in Xn} xn$$

$$\Rightarrow (\Delta) \quad \forall q \in C, S^{q'} \subseteq \bigcup_{x1 \in X1} x1 \times ... \times \bigcup_{xn \in Xn} xn$$

$$\Rightarrow \quad \bigcup_{q \in C} S^{q'} \subseteq \bigcup_{x1 \in X1} x1 \times ... \times \bigcup_{xn \in Xn} xn$$

Donc tous les points de S rejetés par toute contrainte q de C peuvent être représentés par les sommets de X.

FIN ①

② Les hyperarêtes de A et de A' expriment les mêmes points : ceux rejetés.

• $A = \{a / \exists c \in C, a=\{ei, ei \in E\} = S^{c'}\}$

$\Rightarrow$ si $C = \{q1, ..., qm\}$, $A = \{S^{q1'}, ..., S^{qm'}\}$ ainsi $\bigcup_{ai \in A} ai = \bigcup_{c \in C} S^{c'}$

Montrons la même chose pour A' : $\bigcup_{ai' \in A} ai'^{S} = \bigcup_{c \in C} S^{c'}$

• $A' = \{a' = \{x1, ..., xn\} / x1 \in X1, ..., xn \in Xn, \exists q \in C, x1 \times ... \times xn \subseteq S^{q'}\}$

$a'_i{}^{S} = x1 \times ... \times xn$ est la signification en terme de point de $a'_i = \{x1, ..., xn\}$.

① Montrons $\bigcup_{a'_i \in A'} a'_i{}^{S} \subseteq \bigcup_{c \in C} S^{c'}$

Par définition de a'_i, $\forall a'_i \in A', \exists q \in C, a'_i{}^{S} \subseteq S^{q'}$,

$$\Rightarrow \quad \bigcup_{a'_i \in A'} a'_i{}^{S} \subseteq \bigcup_{c \in C} S^{c'}$$

FIN ①

② Montrons $\bigcup_{a'_i \in A'} a'_i{}^{S} \supseteq \bigcup_{c \in C} S^{c'}$

Soit RSC', l'ensemble des points rejetés par c et n'appartenant pas aux points désignés par les hyperarêtes de A' : $\forall c \in C$, $RSC' = S^{c'} \setminus \bigcup_{a'_i \in A'} a'_i{}^{S}$

Démontrons par l'absurde que RSC' = Ø.
Supposons que RSC' ≠ Ø.

Soit un point $s \in RSC'$, $s = \langle s1, ..., sn \rangle$,
comme $s \in S^{c'}$ et $\forall c \in C, S^{c'} \subseteq \bigcup_{x1 \in X1} x1 \times ... \times \bigcup_{xn \in Xn} xn$ (cf ①Δ)

alors $s \in \bigcup_{x1 \in X1} x1 \times ... \times \bigcup_{xn \in Xn} xn$

donc $\exists xs1 \in X1, ..., \exists xsn \in Xn, s \in xs1 \times ... \times xsn$.

or d'après le théorème 4 :
$\forall x1 \in X1 \subseteq X, ..., \forall xn \in Xn \subseteq X, \forall c \in C, x1 \times ... \times xn \subseteq S^{c'}$ ou S^{c}

et $s \in S^{c'} \Rightarrow xs1 \times ... \times xsn \subseteq S^{c'}$ et $\not\subset S^{c}$

Soit $a'_k = \{xs1, ..., xsn\}$, $s \in a'_k{}^S$, $a'_k{}^S = xs1 \times ... \times xsn$ et $\exists c \in C$, $a'_k{}^S \subseteq S^{c'}$
$\Rightarrow a'_k \in A'$

or $RSC' = S^{c'} \setminus \bigcup_{a'_i \in A'} a'_i{}^S \Rightarrow RSC' \cap (\bigcup_{a'_i \in A'} a'_i{}^S) = \emptyset$
or $s \in RSC'$ et $s \in a'_k{}^S$, $a'_k \in A'$, $A' \neq \emptyset$

$\Rightarrow RSC' = \emptyset$ donc contradiction avec l'hypothèse.
$\Rightarrow \bigcup_{a'_i \in A'} a'_i{}^S \supseteq \bigcup_{c \in C} S^{c'}$

FIN ②
FIN ②

$$\bigcup_{a'_i \in A'} a'_i{}^S \subseteq \bigcup_{c \in C} S^{c'} \text{ et } \bigcup_{a'_i \in A'} a'_i{}^S \supseteq \bigcup_{c \in C} S^{c'} \Rightarrow \bigcup_{a'_i \in A'} a'_i{}^S = \bigcup_{c \in C} S^{c'}$$

Donc $\mathcal{H}$ et H sont équivalents.
FIN ❶

❷ H est équivalent à HG avec H=<X, A'> et HG=<X, HA>

Les sommets étant les mêmes, nous ne vérifions que les hyperarêtes de HA :

• $HA = \{ha / ha=xi / xi \in Xi \text{ et } \exists c \in C, D_1 \times ... \times xi \times ... \times D_n \subseteq S^{c'}\}$.

En prenant la notation $ha_i{}^S$ pour les points $D_1 \times ... \times ha_i \times ... \times D_n$, montrons :
$\bigcup_{ha_i \in HA} ha_i{}^S = \bigcup_{c \in C} S^{c'}$

Rappelons nous que nous sommes dans l'hypothèse générale et donc
$S^c = D_1{}^c \times ... \times D_n{}^c$ et $S^{c'} = D_1{}^{c'} \times ... \times D_n \cup ... \cup D_1 \times ... \times D_n{}^{c'}$ (théorème 3).

① Prouvons que $\bigcup_{ha_i \in HA} ha_i{}^S \supseteq \bigcup_{c \in C} S^{c'}$

① Montrons que $\forall c \in C, \forall i \in [1..n], D_i{}^{c'} = \bigcup_{\substack{ha_i \in HA/ \\ ha_i \subseteq D_i{}^{c'}}} ha_i$

① $\forall c \in C, \forall i \in [1..n], D_i{}^{c'} \supseteq \bigcup_{\substack{ha_i \in HA/ \\ ha_i \subseteq D_i{}^{c'}}} ha_i$

Ceci est vraie par définition. $ha_i \subseteq D_i{}^{c'}$ donc l'union est aussi inclus dans $D_i{}^{c'}$.
FIN ①

② $\forall c \in C, \forall i \in [1..n], D_i^{c'} \subseteq \bigcup_{ha_i \in HA / ha_i \subseteq D_i^{c'}} ha_i$

Prouvons-le par l'absurde. L'hypothèse prise est :

$\exists c \in C, \exists i \in [1..n], D_i^{c'} \not\subset \bigcup_{ha_i \in HA / ha_i \subseteq D_i^{c'}} ha_i$

Ceci implique : $\exists v \in D_i^{c'}$ et $v \notin \bigcup_{ha_i \in HA / ha_i \subseteq D_i^{c'}} ha_i$

donc $v \in D_i^{c'}$ et $\forall ha_i \in HA, ha_i \subseteq D_i^{c'}, v \notin ha_i$

Or $D_1 \times ... \times D_i^{c'} \times ... \times D_n \subseteq S^{c'}$ donc $D_1 \times ... \times v \times ... \times D_n \subseteq S^{c'}$

Donc ceci implique que $\exists xi \in X$ tel que $v \in xi$ par construction des xi et donc par construction des ha_i, $\exists ha_i \in HA$ tel que $v \in ha_i$.

Or c'est une contradiction avec $v \notin ha_i$.

Donc $\forall c \in C, \forall i \in [1..n], D_i^{c'} \subseteq \bigcup_{ha_i \in HA / ha_i \subseteq D_i^{c'}} ha_i$ et

FIN ②

Donc avec ① et ② $\forall c \in C, \forall i \in [1..n], D_i^{c'} = \bigcup_{ha_i \in HA / ha_i \subseteq D_i^{c'}} ha_i$ et

FIN ①

or $\forall c \in C, S^{c'} = D_1^{c'} \times ... \times D_n \cup ... \cup D_1 \times ... \times D_n^{c'}$

donc $\forall c \in C, S^{c'} = (\bigcup_{ha_1 \in HA / ha_1 \subseteq D_1^{c'}} ha_1) \times ... \times D_n \cup ... \cup D_1 \times ... \times (\bigcup_{ha_n \in HA / ha_n \subseteq D_n^{c'}} ha_n)$

donc $\forall c \in C, S^{c'} = (\bigcup_{ha_1 \in HA / ha_1 \subseteq D_1^{c'}} ha_1^S) \cup ... \cup (\bigcup_{ha_n \in HA / ha_n \subseteq D_n^{c'}} ha_n^S)$

donc $\forall c \in C, S^{c'} = \bigcup_{ha_i \in HA / \forall i, ha_i \subseteq D_i^{c'}} ha_i^S$

donc $\forall c \in C, S^{c'} \subseteq (\bigcup_{ha_i \in HA} ha_i^S)$

donc $\bigcup_{c \in C} S^{c'} \subseteq (\bigcup_{ha_i \in HA} ha_i^S)$

FIN ①

② Prouvons que $\cup_{ha_i \in HA} ha_i^S \subseteq \cup_{c \in C} S^{c'}$

Par construction des ha_i :

$\forall ha_i \in HA, \exists c \in C, D_1 \times ... \times ha_i \times ... \times D_n \subseteq S^{c'}$

$\Rightarrow \forall ha_i \in HA, \exists c \in C, ha_i^S \subseteq S^{c'}$

$\Rightarrow \forall ha_i \in HA, ha_i^S \subseteq \cup_{c \in C} S^{c'}$

$\Rightarrow (\cup_{ha_i \in HA} ha_i^S) \subseteq \cup_{c \in C} S^{c'}$

FIN ②

① et ② $\Rightarrow (\cup_{ha_i \in HA} ha_i^S) = \cup_{c \in C} S^{c'}$

Nous avons montré l'équivalence entre H et HG.
FIN ❷

Comme $\mathcal{H}$ et H sont équivalents ainsi que H et HG alors $\mathcal{H}$ et HG sont équivalents.
FIN DE PREUVE

D.3.4 Détection des contraintes à relâcher

Dans cette sous-division, nous montrons comment trouver des contraintes de relâchement. Nous utilisons l'hypergraphe de caractérisation HG.

☞ Soit S un espace de recherche, produit cartésien de domaines. S'il est entièrement contraint, alors au moins un domaine est entièrement contraint. Le théorème suivant exprime cette information.

Exemple : Reprenons l'exemple présenté en D.3.2. Nous voyons que le domaine du cours F1 est entièrement contraint : toutes combinaisons avec (F1<-8h) est impossible. Il en est de même avec (F1<-9h). Ceci implique qu'il n'y a pas de solution.

Nous avons représenté en pointillé les D_i^C et en trait plein les complémentaires $D_i^{C'}$. Nous voyons bien qu'il n'y a pas de solution (S = Ø) et que $D_{F1}^C = Ø$ et que $D_{F1}^{C'} = D_{F1}$.

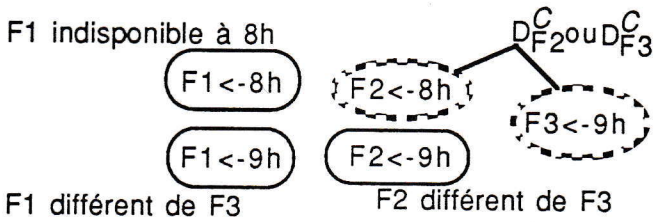


Figure 27. Domaines contraints

Théorème 6 : Soit S un espace de recherche, $S = D_1 \times ... \times D_n$, $S \neq Ø$ et C un ensemble de contraintes. Si S et C satisfont l'hypothèse générale alors ($S^C = Ø \Leftrightarrow \exists D_i \neq Ø, D_i^C = Ø$).

Preuve :

❶ $S^C = Ø \Leftarrow \exists D_i \neq Ø, D_i^C = Ø$

Ceci est évident car $S^C = D_1{}^C \times ... \times D_i{}^C \times ... \times D_n{}^C$
Si $D_i{}^C = \emptyset \Rightarrow D_1{}^C \times ... \times D_i{}^C \times ... \times D_n{}^C = \emptyset \Rightarrow S^C = \emptyset$
FIN ❶

❷ $S^C = \emptyset \Rightarrow \exists\ D_i \neq \emptyset,\ D_i{}^C = \emptyset$

Supposons que c'est faux et donc que $S^C = \emptyset$ et $\forall\ D_i \neq \emptyset,\ D_i{}^C \neq \emptyset$
$\Rightarrow D_1{}^C \times ... \times D_i{}^C \times ... \times D_n{}^C \neq \emptyset \Rightarrow S^C \neq \emptyset$

Or c'est en contradiction avec l'hypothèse $S^C = \emptyset$.

Donc l'implication est vraie.
FIN ❷

Les deux implications vraies prouvent le théorème 6.
FIN DE PREUVE

☞ Soit un domaine D_i entièrement contraint. Prenons les sommets x_l de l'hypergraphe HG, inclus dans ce domaine D_i. Alors, ils sont tous compris dans au moins une hyperarête. Au moins une contrainte est insatisfaite en chaque point issu de x_l. Le théorème 7 prouve ces affirmations.

Exemple : Dans notre figure 27, nous voyons que le domaine {F1<-8h, F1<-9h} est entiérement contraint. F1<-8h appartient à l'hyperarête "F1 différent de 8h". F1<-9h appartient à l'hyperarête "F1 différent de F3".

Théorème 7 : Soit S un espace de recherche, C un ensemble de contraintes et l'hypergraphe HG = <X, HA>. Si S et C satisfont l'hypothèse générale alors :

$$(D_i \neq \emptyset,\ D_i{}^C = \emptyset \Rightarrow \forall\ x_l \in Xi \subseteq X,\ x_l \neq \emptyset,\ mc(x_l) \neq \emptyset).$$

Preuve :

Rappelons que mc(e) correspond aux contraintes insatisfaites sur les points issus de e : mc(e) = {c / ∃ ha ∈ HA, e ∈ ha et $ha^S \subseteq S^{c'}$}. En remplaçant mc(e) dans la formule ci-dessus on a donc :

$D_i{}^C = \emptyset \Rightarrow \forall\ x_l \in Xi \subseteq X,\ x_l \neq \emptyset,\ \exists\ c \in C,\ D_1 \times ... \times x_l \times ... \times D_n = ha^S \subseteq S^{c'}$
Supposons que ce fait est faux. Montrons alors dans ce cas une contradiction.

Hypothèse: $\exists\ x_l \in Xi \subseteq X,\ x_l \neq \emptyset,\ \forall\ c \in C,\ D_1 \times ... \times x_l \times ... \times D_n \not\subset S^{c'}$

(Λ) donc $\exists\ x_l \in Xi \subseteq X,\ \forall\ \mathbf{c} \in C,\ \exists\ d1 \in D_1, ..., \exists\ d_l \in x_l,\ x_l \neq \emptyset, ..., \exists\ dn \in D_n,\ d1 \times ... \times d_l \times ... \times dn \notin S^{c'}$

Or par construction des x_l : $d1 \times ... \times x_l \times ... \times dn \subseteq S^{c'}$ ou $d1 \times ... \times x_l \times ... \times dn \subseteq S^c$
$\Rightarrow \exists\ x_l \in Xi,\ x_l \neq \emptyset,\ \forall\ c \in C,\ \exists\ d1 \in D_1, ..., \exists\ dn \in D_n,\ d1 \times ... \times x_l \times ... \times dn \subseteq S^c$

Supposons les deux cas : ❶ le produit cartésien est inclus dans S^C
❷ le produit cartésien n'est pas inclus dans S^C

❶ $\exists x_l \in Xi \subseteq X, x_l \neq \emptyset, \forall c \in C, \exists d1 \in D_1, ..., \exists dn \in D_n, d1 \times ... \times x_l \times ... \times dn \subseteq S^c$ et $\subseteq S^C$

Or $D_i^C = \emptyset \Rightarrow S^C$ est vide donc x_l est vide donc en contradiction avec $x_l \neq \emptyset$

FIN ❶

❷ $\exists x_l \in Xi \subseteq X, x_l \neq \emptyset, \forall c \in C, \exists d1 \in D_1, ..., \exists dn \in D_n, d1 \times ... \times x_l \times ... \times dn \subseteq S^c$ et $\not\subset S^C$ donc $\not\subset (\cap_{c \in C} S^c)$

Cela signifie $\exists q \in C, d1 \times ... \times x_l \times ... \times dn (\not\subset S^q) \subseteq S^{q'}$

or c'est une contradiction avec l'hypothèse (Λ).

Pour ($c = q$) : $d_1 \times ... \times x_l \times ... \times d_n \subseteq S^{q'}$ et $\subseteq S^q$ donc $x_l = \emptyset$.

Il y a donc contradiction.

FIN ❷

Donc l'implication est vraie et le théorème aussi.

FIN DE PREUVE

☞ Le théorème suivant donne les contraintes de relâchement. Elles sont obtenues à partir de l'hypergraphe de caractérisation HG. Elles constituent un ensemble CR.

Les éléments de CR sont des ensembles de contraintes à relâcher. Leur relâchement permet qu'un point de S ne soit plus contraint. Ces éléments sont des contraintes de relâchement.

Exemple : Reprenons l'exemple de la division D.3.2, nous voyons les contraintes de relâchement en gras dans la figure 28 :

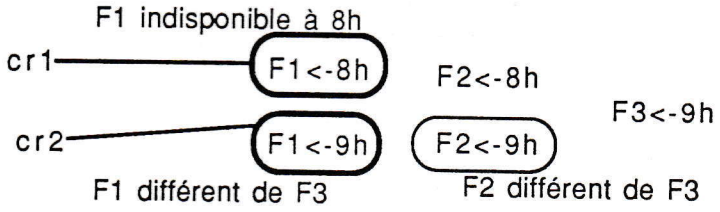


Figure 28. Contraintes à relâcher

Théorème 8 : Soit S un espace de recherche, C un ensemble de contraintes, S et C satisfaisant l'hypothèse générale et HG = <X, HA> alors :

$$S^C = \emptyset \Rightarrow CR = \{cr_k / \forall x1 \in X1, ..., \forall xn \in Xn, cr_k = \bigcup_{\substack{1 \leq i \leq n / \\ D_i^C = \emptyset}} mc(xi)\}$$

Preuve :

Nous savons que $S^C = \emptyset \Rightarrow \exists D_i^C = \emptyset$ (théorème 6)

$\Rightarrow \exists D_i^C = \emptyset, \forall x_l \in Xi \subseteq X, mc(x_l) \neq \emptyset$ (théorème 7)

❶ Soit D_i tel que $D_i^{\mathcal{C}} = \emptyset$,
comme $\forall x_l \in Xi$ et $x_l \subseteq D_i$ alors $D_i^{\mathcal{C}\setminus mc(x_l)} \neq \emptyset$.

Donc par construction de CR, $\forall cr_k \in CR$, $\forall D_i$, $D_i^{\mathcal{C}} = \emptyset$, $\exists x_l \in Xi$, $mc(x_l) \subseteq cr_k$

donc $\forall cr_k \in CR$, $\forall D_i$, $D_i^{\mathcal{C}} = \emptyset$, $\exists x_l \in Xi$, $mc(x_l) \subseteq cr_k$ et $D_i^{\mathcal{C}\setminus mc(x_l)} \neq \emptyset$

donc $\forall cr_k \in CR$, $\forall D_i$, $D_i^{\mathcal{C}} = \emptyset$, $D_i^{\mathcal{C}\setminus cr_k} \neq \emptyset$
FIN ❶

❷ Soit D_i tel que $D_i^{\mathcal{C}} \neq \emptyset$, il est évident que $D_i^{\mathcal{C}\setminus cr_k} \neq \emptyset$
FIN ❷

Nous pouvons conclure alors que $\forall cr_k \in CR$, $D_1^{\mathcal{C}\setminus cr_k} \times \ldots \times D_n^{\mathcal{C}\setminus cr_k} \neq \emptyset$
et donc $\forall cr_k \in CR$, $S^{\mathcal{C}\setminus cr_k} \neq \emptyset$ avec $S^{\mathcal{C}} = \emptyset$.

donc chaque cr_k constitue des contraintes de relâchement.
FIN DE PREUVE

☞ Nous allons nous apercevoir que relâcher ces contraintes est suffisant et nécessaire.

Corollaire : Soit S un espace de recherche, $\mathcal{C}$ un ensemble de contraintes, S et $\mathcal{C}$ satisfaisant l'hypothèse générale, soit HG = <X, HA> et CR l'ensemble de contraintes de relâchement alors :

si $S^{\mathcal{C}} = \emptyset \Rightarrow$ il est nécessaire et suffisant de relâcher $cr_k \in CR$ pour que $S^{\mathcal{C}\setminus cr_k} \neq \emptyset$.

Preuve :

❶ Suffisant
Les éléments de CR sont des contraintes de relâchement. Ce relâchement est donc suffisant pour libérer des points tel que : $S^{\mathcal{C}\setminus cr_k} \neq \emptyset$.
FIN ❶

❷ Nécessaire
Soit $p \in \mathcal{P}(\mathcal{C})$ (ensemble des parties de $\mathcal{C}$), il faut montrer que :
p est de relâchement $(S^{\mathcal{C}} = \emptyset \Rightarrow S^{\mathcal{C}\setminus p} \neq \emptyset) \Leftrightarrow (\exists cr_k \in CR, cr_k \subseteq p)$

Prouvons-le par la négation de la formule :
p n'est pas de relâchement $\Leftrightarrow (\forall cr_k \in CR, cr_k \not\subset p)$

① Le sens $\Rightarrow$ est évident : si p n'est pas de relâchement $cr_k \not\subset p$, sinon comme cr_k est de relâchement, p serait de relâchement.
FIN ①

② Prouvons le sens inverse ($\Leftarrow$) : $S^{\mathcal{C}\setminus p} \neq \emptyset \Leftarrow (\exists cr_k \in CR, cr_k \subseteq p)$

Montrons que $\forall cr_k \in CR$, $cr_k \not\subset p$, $p \neq \emptyset$, $cr_k \neq \emptyset \Rightarrow \forall cr_k \in CR$, $(cr_k \setminus p) \neq \emptyset$.

① Montrons que $S = \bigcup_{x1 \in X1} x1 \times \ldots \times \bigcup_{xn \in Xn} xn$

d'après le théorème 5 (❶, ①) : $\bigcup_{c \in C} S^{c'} \subseteq \bigcup_{x1 \in X1} x1 \times \ldots \times \bigcup_{xn \in Xn} xn$

et $\forall$ xi, xi $\subseteq D_i$ donc $\bigcup_{x1 \in X1} x1 \times \ldots \times \bigcup_{xn \in Xn} xn \subseteq S$ or $S^C = \emptyset \Rightarrow S = \bigcup_{c \in C} S^{c'}$

$\Rightarrow \bigcup_{c \in C} S^{c'} = S \subseteq \bigcup_{x1 \in X1} x1 \times \ldots \times \bigcup_{xn \in Xn} xn \subseteq S$ donc $S = \bigcup_{x1 \in X1} x1 \times \ldots \times \bigcup_{xn \in Xn} xn$

FIN ①

② cr_k est tel que : $\forall x1 \in X1, \ldots, \forall xn \in Xn, cr_k = \bigcup_{1 \le i \le n / D_i^C = \emptyset} mc(xi)$

Par hypothèse $\forall cr_k \in CR, cr_k \not\subset p \Rightarrow \forall x1 \in X1, \ldots, \forall xn \in Xn, \bigcup_{1 \le i \le n / D_i^C = \emptyset} mc(xi) \setminus p \neq \emptyset$

$\Rightarrow \forall x1 \in X1, \ldots, \forall xn \in Xn, \exists j/xj \in Xj, mc(xj) \neq \emptyset, mc(xj) \not\subset p$
$\Rightarrow \forall x1 \in X1, \ldots, \forall xn \in Xn, \exists j/xj \in Xj, \exists c \in mc(xj), c \notin p$ et donc $c \in C \setminus p$

et donc par définition des mc(xj) $\Rightarrow \exists c \in C \setminus p, D_1 \times \ldots \times xj \times \ldots \times D_n \subseteq S^{c'}$
donc $\forall x1 \in X1, \ldots, \forall xn \in Xn, \exists j/xj \in Xj, \exists c \in C \setminus p, D_1 \times \ldots \times xj \times \ldots \times D_n \subseteq S^{c'}$

soit $s = x1 \times \ldots \times xj \times \ldots \times xn$, $s \subseteq D_1 \times \ldots \times xj \times \ldots \times D_n$ donc
$\forall x1 \in X1, \ldots, \forall xn \in Xn, s = x1 \times \ldots \times xj \times \ldots \times xn, \exists j/xj \in Xj, \exists c \in C \setminus p, s \subseteq S^{c'}$

donc $\forall x1 \in X1, \ldots, \forall xn \in Xn, s = x1 \times \ldots \times xj \times \ldots \times xn, \exists c \in C \setminus p, s \subseteq S^{c'}$

donc $\forall x1 \in X1, \ldots, \forall xn \in Xn, s = x1 \times \ldots \times xj \times \ldots \times xn, s \subseteq \bigcup_{c \in C \setminus p} S^{c'}$

donc $\bigcup_{x1 \in X1} x1 \times \ldots \times \bigcup_{xn \in Xn} xn \subseteq \bigcup_{c \in C \setminus p} S^{c'}$

donc d'après le ① : $S = \bigcup_{x1 \in X1} x1 \times \ldots \times \bigcup_{xn \in Xn} xn$ donc $S \subseteq \bigcup_{c \in C \setminus p} S^{c'}$

Ceci implique que $S^{C \setminus p} = S \setminus \bigcup_{c \in C \setminus p} S^{c'} = \emptyset$.

$\Rightarrow$ p n'est pas de relâchement.
FIN ②

Nous avons donc démontré le corollaire.
FIN DE PREUVE

Nous allons donc utilisé ces hypergraphes en P.L.C.. Quand la résolution ne détectera aucune solution, nous pourrons alors détecter avec le théorème 8 les contraintes à relâcher.

D.4 HCI pour la P.L.C. avec hiérarchie et domaines finis

D.4.1 Introduction

Dans cette division, nous allons mettre en place plusieurs mécanismes. Ceux-ci sont basés sur un HCI et plus particulièrement sur un hypergraphe de caractérisation HG. Ils concernent des langages de la P.L.C intégrant des domaines finis et une hiérarchie de contraintes.

Nous décrivons comment est modélisé l'hypergraphe de caractérisation. Nous présentons donc d'abord ses sommets puis ses hyperarêtes. Nous montrons alors sa construction à travers la résolution du langage P.L.C. choisi. Enfin, nous présentons l'utilisation de tels hypergraphes dans les mécanismes de relâchement et de recherche de "meilleures" solutions.

Une dernière partie présente quelques recommendations. Certaines hypothèses doivent être prises si nous voulons obtenir des mécanismes "sûrs" de relâchement et de recherche de "meilleures" solutions.

D.4.2 HCI en P.L.C.

a) Classement et hiérarchie de contraintes de la P.L.C.

Dans les langages de la P.L.C., nous trouvons plusieurs sortes de contraintes. En premier lieu, nous supposons que l'ensemble des contraintes C est hiérarchisé :

- C_0 désigne l'ensemble des contraintes absolues,
- $C_{i,\ i>0}$ désigne l'ensemble des contraintes secondaires.

Les contraintes absolues ne peuvent pas être relâchées. Toutes les contraintes d'unification sont absolues.

Seules les contraintes secondaires peuvent être relâchées. Un niveau de hiérarchie est associé à chacune d'entre elles.

Par exemple, dans le problème d'emploi du temps nous avons des propriétés à tenir compte absoluement et d'autres propriétés exprimant des voeux ou des désirs.

Rappelons que les contraintes constituent des relations entre variables. Une méthode locale de satisfaction leur est associée.

En rapport avec les définitions introduites en (D.3.1 b), nous distinguons deux types de contraintes : les contraintes élémentaires et compositionnelles.

Les contraintes de la P.L.C. correspondent aux contraintes compositionnelles. Une application de leur méthode pour une raison particulière élémentaire représente une contrainte élémentaire.

Au fur et à mesure de la résolution, l'ensemble des déclenchements associés à une contrainte du programme pose toutes les contraintes élémentaires. Cet ensemble forme la contrainte compositionnelle associée.

En résumé, nous avons des contraintes élémentaires et compositionnelles qui peuvent être absolues ou secondaires.

b) Sommets de HCI ou valeurs "sûres" de la P.L.C.

Les sommets des hypergraphes HG sont construits à partir de valeurs rejetées. Ces rejets sont faits par des contraintes secondaires. Les valeurs rejetées par les contraintes absolues ne sont pas prises en compte.

Prenons C l'ensemble hiérarchisé de contraintes et Z l'espace de recherche satisfaisant les contraintes absolues C_0. Supposons que l'hypothèse générale est vérifiée. Chaque sommet xi représente alors une valeur "sûre" d'un domaine D_i. Elle n'est pas rejetée par les contraintes absolues mais rejetée par les contraintes secondaires. Nous construisons les sommets xi en tant que singleton tel que :

$$\forall\, i, \forall xi \in Xi,\ xi = \{v\},\ v \in D_i^{C_0} ;\ v \notin D_i^{C}$$

Exemple : Reprenons l'exemple de la division D.3. Rappelons que nous avons trois cours : F1, F2 et F3 et l'espace de recherche S = {(F1<-8h, F2<-8h, F3<-9h), (F1<-8h, F2<-9h, F3<-9h), (F1<-9h, F2<-8h, F3<-9h), (F1<-9h, F2<-9h, F3<-9h)}.

Nous posons les contraintes **secondaires** suivantes : F1 différent de F3, F2 différent de F3 et la contrainte **absolue** : F1 indisponible à 8h.

Nous pouvons constater que la valeur (F1<-8h) n'est pas une valeur "sûre". Ainsi les sommets de l'HCI est constitué par : {F1<-9h, F2 <-8h, F2 <-9h, F3<-9h}.

Il est facile alors de vérifier que les sommets sont bien construits (Définition 11) :

- Les sommets sont disjoints entre eux car les valeurs sont disjointes entre elles.

- Les points composés avec chaque xi sont exclusivement agréés ou rejetés. Car chaque composition est un unique point (xi est un singleton).

- Il existe une contrainte (secondaire) qui rejette chaque xi.

c) Hyperarêtes de HCI ou contraintes élémentaires secondaires

Nous avons abordés les divers types de contraintes. Les hyperarêtes sont associées alors aux contraintes élémentaires secondaires (c'est-à-dire une valeur et la raison de son rejet).

Ceci est dû à deux raisons. D'abord, l'union de contraintes élémentaires forment les contraintes compositionnelles. Ensuite, seules les contraintes secondaires peuvent être relâchées.

Supposons l'hypothèse générale vraie. Montrons que ces hyperarêtes peuvent exister (s'il existe une insatisfaction).

Soit c contrainte élémentaire secondaire, d'après l'hypothèse générale, $Z^{c'} = D_1^{c'} \times D_2 \times \ldots \times D_n \cup \ldots \cup D_1 \times D_2 \times \ldots \times D_n^{c'}$

Si $Z^{c'} \neq \emptyset$ alors $\exists$ xi = {v} $\subseteq D_i^{c'}$. Or $D_1 \times ... \times D_i^{c'} \times ... \times D_n \subseteq Z^{c'}$ donc $D_1 \times ... \times$ xi $\times ... \times D_n \subseteq Z^{c'}$ et xi est une hyperarête ha_i (définition 11).

Dans la suite, nous associerons hyperarêtes à une valeur et à la raison de son rejet par une contrainte élémentaire.

Exemple : Dans notre exemple, les hyperarêtes à ce sens sont composées des contraintes : "(F1<-9h) différent de (F3<-9h)", "(F2<-8h) différent de (F3<-9h)" et "(F2<-9h) différent de (F3<-9h)". Bien entendu, la contrainte "(F2<-8h) différent de (F3<-9h)" est toujours vraie et n'est pas à relâcher.

Représentons l'HCI ansi construit :

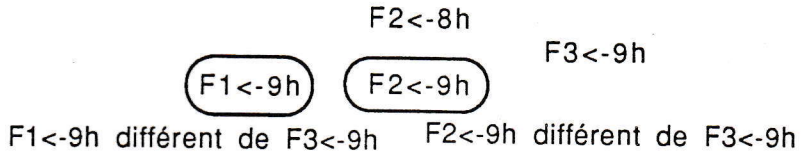


Figure 29. Exemple d'HCI

D.4.3 Construction d'un HCI en P.L.C.

a) Hypothèses sur les méthodes locales de satisfaction

i) Hypothèse sur les contraintes

Une contrainte élémentaire correspond à une application particulière de la méthode locale dans un déclenchement pour une raison particulière élémentaire. Il faut que deux déclenchements différents, dans une même dérivation, n'appliquent pas la même contrainte élémentaire et que toutes les contraintes élémentaires soient deux à deux indépendantes.

Ainsi, deux déclenchements successifs d'une même contrainte P.L.C. ne doivent pas avoir des raisons élémentaires communes de déclenchement et ne doivent pas refaire des tests communs. Ceci doit être pris en compte par l'algorithme local de satisfaction.

Supposons cette hypothèse respectée. Relâchons une contrainte élémentaire. La contrainte P.L.C. associée reste. L'hypothèse garantit que cette contrainte élémentaire, dans le futur de la résolution, ne sera pas à nouveau appliquer. Ainsi, les points libérés ne seront plus rejetés pour les mêmes raisons.

ii) Hypothèse sur les ensembles d'agrément

Ceci correspond à l'hypothèse générale décrite ci-avant. Si l'espace de recherche est un produit cartésien $S = D_1 \times ... \times D_n$ alors l'ensemble d'agrément d'une contrainte élémentaire secondaire c doit être aussi un produit cartésien : $S^c = D_1^c \times ... \times D_n^c$.

Nous allons maintenant différencier les conséquences des méthodes de satisfaction des contraintes absolues de celles des contraintes secondaires.

b) Les sommets de HCI

Au fur et à mesure de la résolution, nous ne gardons que les points satisfaisant l'ensemble des contraintes absolues. Les points rejetés par une méthode associée à une contrainte absolue sont **éliminés** de l'espace de recherche.

Ce mécanisme est celui déjà pratiqué dans les langages tels que CLEF v1 et CHIP.

Nous mémorisons dans l'hypergraphe, les points rejetés par les contraintes secondaires.

Le mécanisme du langage est ici changé. Il faut qu'il détecte la présence d'une contrainte secondaire. Dans ce cas, il mémorise les points rejetés. Ces points restent dans l'espace de recherche.

c) Les hyperarêtes contraintes à travers l'étape de dérivation

Nous prenons les contraintes secondaires insatisfaites pour exprimer les hyperarêtes de l'HCI. Cependant, l'hypergraphe évolue avec la résolution. L'étape de dérivation est donc changer. Nous rajoutons l'hypergraphe HG dans la description de l'étape. En fait, d'une étape à une autre, nous construisons un nouvel hypergraphe déduit du précédent. La nouvelle instruction "->" tient compte de cet hypergraphe.

N.B. : *Nous avons un hypergraphe HG (X, HA) qui prend ses sommets dans des domaines et ces hyperarêtes de HA dans des contraintes.*

Pour des raisons de simplicité de notation, nous confondrons X et HA avec les domaines Di et les contraintes C d'entrée.

L'hypergraphe HG d'une étape de dérivation sera noté suivant cette manière.

Soit P un programme, les règles Rr, R_0 une requête, l'instruction "->" est:

état initial : G_i = <Wi, :- t0, ..., tn., C, HG($\cup D_i^{C_0}$, $C_{u,u>0}$)> où
Wi est l'ensemble des variables,
t0, ..., tn des termes de la P.L.,
C des contraintes sur $\mathcal{D}$ avec C_0 l'ensemble des contraintes absolues, $C_{u,u>0}$ celui des contraintes secondaires.

condition : $\exists$ Rj $\in$ P, Rj $\equiv$ (L :- L1, ..., Lm, Q_1, ..., Q_v.) où
L, L1, ..., Lm sont des prédicats de la P.L.,
Q_1, ..., Q_v des contraintes sur $\mathcal{D}$, avec K_0 les contraintes absolues et $K_{u,u>0}$ les contraintes secondaires de Q_1, ..., Q_v.
et $C \cup \{Q_1, ..., Q_v, L =_u t0\}$ satisfait par la méthode de réduction choisie.

état final : G_{i+1} =<Wi+1, :- L1, ..., Lm, t1, ..., tn., réduction($C \cup \{Q_1, ..., Q_v, L =_u t0\}$), HG($\cup D_i[C_0 \cup K_0 \cup L=_u t0]$, $C_{u,u>0} \cup K_{u,u>0}$)>

L'espace de recherche de G_{i+1} est formé avec les points satisfaisant les contraintes absolues de G_i et de la règle Rj.

d) Construction d'un HCI par les arrêts d'une dérivation

Avec cette nouvelle instruction "->", nous trouvons deux arrêts différents. Le premier arrêt est sur un succès ($S^C \neq \emptyset$). Le deuxième est sur un échec ($S^C = \emptyset$).

Cependant, nous différencions deux cas d'échec différents. L'un des cas intervient quand $S^C = \emptyset$ et $S^{C_0} = \emptyset$. L'autre cas se produit quand $S^C = \emptyset$ et $S^{C_0} \neq \emptyset$.

Ce second échec est nommé **pseudo-échec**. Nous mémorisons, alors, l'étape de dérivation en arrêt. Car le relâchement d'une contrainte peut provoquer le redémarrage de la dérivation.

A la fin de cette dérivation, nous avons donc obtenu un HCI final. En cas de besoin, nous pourrons l'analyser.

i) Arrêt uniquement lié aux contraintes absolues

L'arrêt de la dérivation est en échec. Il se passe dans le cas suivant :

quand $\forall$ Rj $\in$ P, Rj $\equiv$ (L :- L1, ..., Lm, Q_1, ..., Q_V.),
et G_i = <Wi, :- t0, ..., tn., C, HG (X, HA)>
et $C_0 \cup K_0 \cup \{L =_u t0\}$ insatisfait par la méthode de réduction choisie, K_0 étant les contraintes absolues de Q_1, ..., Q_V.

Cet échec est absolu en opposition avec celui qui suit. Cette dérivation n'a aucune solution.

ii) Arrêt lié aux contraintes absolues et secondaires

Cet arrêt n'est pas définitif. Nous le nommons pseudo-échec.

Il faut noter que peut C être incomplet dû à des relâchements précédents et qu'il évolue suite à des relâchements successifs.

L'étape de dérivation est mémorisé. Ainsi il se passe dans le cas suivant :

quand G_i -> <Wτ, :- t0, ..., tn., C, HG (X, HA)>
et C_0 satisfait par la méthode de réduction choisie,
et C insatisfait par la méthode de réduction choisie.

Nous mémorisons la dérivation sous la forme suivante :
G_{i+1} = <Wτ, :- t0, ..., tn., $C \cup \{mc(xi), xi \in X\}$, HG (X, HA)>

Le redémarrage de cette dérivation se fait à partir de cette étape.

A une dérivation, nous pouvons associer plusieurs relâchements successifs (à des étapes différentes). Pour avoir une analyse saine de la cause de l'échec et un redémarrage sain de la dérivation, il faut reintroduire toutes les contraintes insatisfaites : mc(xi).

Le redémarrage d'une dérivation se fera avec un ensemble complet de contraintes

diminué des contraintes relâchées et avec un hypergraphe HG complet (avec les contraintes relâchées). Nous verrons plus loin comment redémarrer une dérivation (D.4.4 b i).

iii) Arrêt lié à un succès

Cet arrêt se fait dans le cas suivant :

quand G_i -> <Wτ, Ø, C, HG (X, HA)>, C satisfait par la méthode de réduction choisie.

Dans ce cas, les points satisfaisant C (S^C) sont des solutions exactes. Cependant, les points de S^C0 et n'appartenant pas à S^C sont des solutions approchées. Ils peuvent devenir des solutions exactes en cas de relâchement. Il faut donc mémoriser cette étape :

$$G_{i+1} = <W\tau, Ø, C \cup \{mc(xi), xi \in X\}, HG (X, HA)>$$

Là encore, la dérivation peut avoir été la conséquence de relâchements passés. Il faut mémoriser toutes les contraintes insatisfaites au cours de cette dérivation.

A chaque dérivation, nous avons un hypergraphe HCI. Ainsi à la fin de toutes les dérivations, nous avons un ensemble d'HCI.

D.4.4 Utilisation des HCI

a) Analyses d'insatisfaction dans une étape de dérivation

Cette sous-division indique comment trouver des insatisfactions dans une étape de dérivation. Ceci revient à trouver les points rejetés par un déclenchement d'une contrainte donc une contrainte élémentaire.

Nous indiquons ensuite la manière de trouver les contraintes de relâchement.

i) Détection d'insatisfaction d'une contrainte élémentaire

Nous voulons avoir les points insatisfaisant une contrainte élémentaire π, lors d'une étape de G_i à G_{i+1}.

Soit G_{i+1}=<Wi+1, :- t0, ..., tn., C, HG (X, HA)>. Nous voulons connaître les solutions insatisfaisant une contrainte élémentaire π de C. Cela consiste à récolter tous les xi de X, tel que π ∈ mc(xi). Les points désignés forment l'ensemble $D_1^{\pi'} \times D_2 \times ... \times D_n \cup ... \cup D_1 \times D_2 \times ... \times D_n^{\pi'}$.

ii) Contraintes à relâcher

Supposons une dérivation en pseudo-échec avec $S^C0 \neq Ø$ et $S^C = Ø$. Nous savons qu'il existe un domaine D_i tel que $D_i^C = Ø$ (théorème 6). Ce domaine appartient à une variable *vi. Il est entièrement restreint par les contraintes secondaires. Notons que les valeurs de D_i dans cette dérivation satisfont les contraintes absolues.

Les contraintes de relâchement sont une combinaison issue d'union de contraintes insatisfaites. Les points les insatisfaisant sont composés d'une valeur de chaque domaine

- entièrement restreint.

Par exemple, si quatre domaines D_a, D_b, D_c, D_d sont entièrement restreints, alors les combinaisons possibles cr_k sont :

$\forall$ da $\in$ D_a, xa={da} $\in$ Xa, $\forall$ db $\in$ D_b, xb={db} $\in$ Xb, $\forall$ dc $\in$ D_c, xc={dc} $\in$ Xc, $\forall$ dd $\in$ D_d, xd={dd} $\in$ Xd, cr_k= mc(xa) $\cup$ mc(xb) $\cup$ mc(xc) $\cup$ mc(xd)

b) Relâchement et "meilleures" solutions

i) Relâchement

☞ Les différents aspects et conséquences d'un relâchement : principe

Nous allons décrire le principe de relâchement à l'aide d'un petit exemple.

Pour plus de précision, nous complétons les notations des dérivations. Une dérivation est un chemin de la racine jusqu'à un point d'arrêt dans l'arbre de résolution. Les dérivations ont donc des parties communes. $G_{i,j}$ représente l'étape i de la dérivation j.

Soit P un programme, R_0 une requête, regardons la figure 30. Nous y voyons 3 dérivations. Elles ont toutes finies en pseudo-échec.

Les contraintes élémentaires $\pi 1$, $\pi 2$ et $\rho 1$ ont réduit l'espace de recherche S. Elles forment deux contraintes P.L.C. π et ρ : $\pi = \pi 1 \cup \pi 2$ et $\rho = \rho 1$.

Les contraintes de relâchement de $G_{2,2}$ sont {{$\pi 1$}}. Celles de $G_{3,3}$ sont {{$\pi 1$, $\pi 2$}} et celles de $G_{3,1}$ sont {{$\rho 1$}, {$\pi 1$}, {$\pi 2$}}.

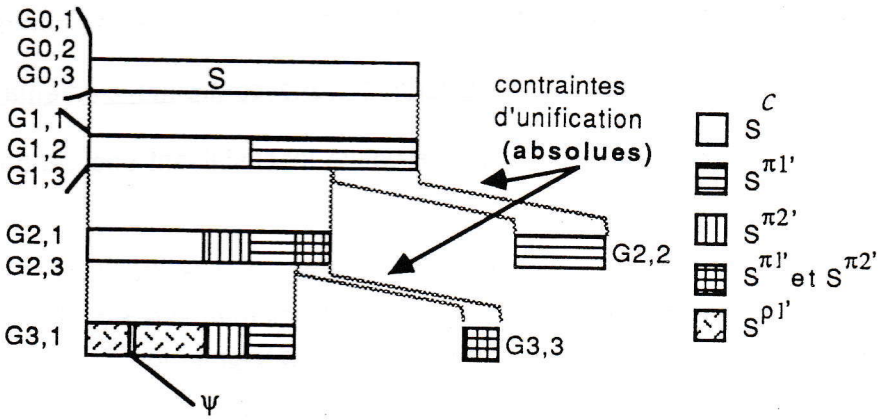


Figure 30. Contexte d'échec

1) Libérons le point particulier ψ. Quelles contraintes faut-il relâcher? Que cela implique-t-il ?

Nous voyons sur le figure 30 que ψ appartient à l'espace de recherche de $G_{3,1}$. Seule la contrainte $\rho 1$ est insatisfaite en ce point. Il suffit de la relâcher en ψ. L'espace de recherche de $G_{3,1}$ devient réduit à ψ. Cette dérivation redémarre.

2) Relâchons la contrainte élémentaire $\pi 2$. Cela implique de redémarrer la

dérivation 1 avec l'étape $G_{3,1}$. Les contraintes de relâchement des autres dérivations ne sont pas inclus dans {π2}.

3) Relâchons la contrainte P.L.C. π. Nous redémarrons les 3 dérivations. Nous repartons donc des étapes $G_{3,1}$, $G_{3,3}$, $G_{2,2}$.

4) Supposons que nous sommes dans une dérivation (1 par exemple). Admettons que les solutions de celle-ci ne satisfont pas un utilisateur.

Un utilisateur veut relâcher la contrainte π1. Cela implique de continuer la dérivation 1 avec $G_{3,1}$ et la dérivation 2 avec $G_{2,2}$. Car π1 fait partie des contraintes de relâchement des étapes $G_{3,1}$ et $G_{2,2}$.

Soit un problème Pj avec les contraintes $C\backslash\pi$ et l'espace de recherche S. Soit un problème Pk avec les contraintes C et le même espace de recherche S.

Comment chercher toutes les solutions de Pj en résolvant Pk puis en utilisant le mécanisme de relâchement? Il faut :

- mémoriser les points S^C. Ils sont solutions de Pk. Ils font partis de $S^{C\backslash\pi}$. Ils sont solutions de Pj.
- analyser les dérivations en pseudo-échec. Il faut chercher celles qui ont un ensemble de contraintes de relâchement inclus dans celui des contraintes relâchées π. Nous redémarrons celles qui y correspondent.
- les solutions globales de Pj sont les points de S^C plus les solutions des dérivations redémarrées.

☞ Redémarrage d'une dérivation

Comment redémarrer une dérivation en pseudo-échec avec le relâchement voulu des contraintes π?

Soit la dérivation mémorisé : G_i = <Wi, Ø, C, HG (X, HA)>, le relâchement de π implique que la nouvelle étape est : G_{i+1} = <Wi, Ø, $C\backslash\pi$, HG (X, HA)>.

Le relâchement de π n'intervient pas dans les hyperarêtes de HG. L'ensemble des contraintes insatisfaites reste complet dans cette dérivation avec les hyperarêtes HA. Ceci nous permet de mesurer la dérivation avec les divers relâchements effectués.

ii) Mesure sur les "solutions"

Mesurons une dérivation avec ses contraintes insatisfaites sur un point ψ spécifique. Si cette dérivation est en pseudo-échec, cette mesure porte sur la cause de son arrêt. En général, la mesure m d'une dérivation porte sur sa dernière étape :

Soit G_i = <Wi, Ø, C, HG (X, HA)>,

$m(\psi, G_i) = g(\psi, h, w)$, $\psi \subseteq h^S$, $h \in$ HA, w est une note attribuée aux contraintes donc aux hyperarêtes h.

Par exemple, une mesure peut caractériser le nombre de contraintes insatisfaites

en ψ : $g(\psi, h, w) = -\text{Card}(\{h / \psi \subseteq h^S\})$. La note w ici ne sert à rien.

N.B. : *Si la mesure s'applique à des dérivations redémarrées, elle tient compte des contraintes relâchées par l'hypergraphe HG.*

iii) Meilleurs relâchements ≈ meilleures solutions

Nous voulons obtenir des meilleures solutions. Trois cas peuvent se produire :

- le problème a des solutions exactes. Elles sont décrétées meilleures.
- le problème n'a pas de solutions, ni exactes, ni partielles. Aucune dérivation n'a été mémorisée. Alors il n'y a pas de solution.
- le problème n'a pas de solution exacte et a des solutions partielles. Il existe des dérivations mémorisées. Par la suite, nous nous plaçons dans ce cas.

Le principe est traduit par le processus suivant :

① Chercher le meilleur relâchement de contraintes élémentaires dans les dérivations en pseudo-arrêt.
② Continuer une des dérivations possibles en pseudo-arrêt avec le meilleur relâchement.
③ Si l'arrêt de cette dérivation est un échec ou un pseudo-échec, refaire le processus à partir de ①.
④ Si l'arrêt est un succès, arrêter le processus. Les solutions trouvées sont alors les meilleures.

Un meilleur relâchement est une relaxation de contraintes de relâchement. Un point est donc libéré. La mesure sur cette dérivation et sur les points libérés donne alors la meilleure note.

Nous avons donc deux calculs :

-calculer la mesure sur les contraintes de relâchement puis,
-calculer la meilleure mesure sur toutes celles calculées.

La fonction f de recherche de meilleure solution doit avoir une "garantie". Il faut que la mesure m à l'étape G_i en un point ψ soit meilleure ou égale que celle de toute étape suivante G_{i+1} sur ce même point : $m(\psi, G_i)$ meilleure ou égale par f que $m(\psi, G_{i+1})$. En d'autres termes, la mesure ne peut que s'empirer.

Par exemple, prenons la mesure calculant le nombre de contraintes insatisfaites ($-\text{Card}(\{h / \psi \subseteq h^S\})$), f est une fonction de Minimisation.

D.4.5 Recommendations

Cette division (D.4) exprime une nouvelle résolution. Trois mécanismes forment principalement celle-ci. Le premier mémorise les solutions réalisables insatisfaisant les contraintes secondaires. Le deuxième relâche des contraintes secondaires. Le troisième fournit les meilleures solutions.

Nous montrons que la résolution P.L.C. ordinaire n'est pas changé. Nous montrons que le relâchement de contraintes est fait d'une façon sûre et saine. Nous montrons enfin que les meilleures solutions le sont effectivement.

a) Construction

Supposons que nous ayons l'espace de recherche S et l'ensemble de contraintes C. Nous comparons la résolution normale à la résolution avec mémorisation. Nous voulons montrer que les cas de succès donnent les mêmes solutions. Les arrêts se font au même moment.

Dans le cas de la résolution P.L.C. normale, les cas de succès interviennent quand nous avons $S^C \neq \emptyset$. Il existe donc des points satisfaisant toutes les contraintes de C. C'est aussi le cas pour la nouvelle résolution.

Dans la résolution sans mémorisation, les cas d'arrêt en échec d'une dérivation interviennent quand $S^C = \emptyset$. C'est aussi le cas pour la nouvelle résolution avec une mémorisation quand $S^{C_0} \neq \emptyset$.

La seule différence est sur la méthode locale de satisfaction d'une contrainte :

- Dans le cas de la résolution normale, les points satisfaisant toutes les contraintes de C (S^C) sont testés par la méthode de satisfaction de chaque contrainte.

- Dans le cas de la résolution avec mémorisation, les points testés sont ceux satisfaisant les contraintes absolues (S^{C_0}). Les tests sont donc plus nombreux.

Les solutions partielles ($S^{C'} \cap S^{C_0}$) sont stockées dans l'hypergraphe HG. Les solutions réalisables (S^{C_0}) sont mémorisées dans les domaines des variables. Ceci permet de retrouver les points de S^C : $S^{C_0} \setminus (S^{C'} \cap S^{C_0})$.

Ainsi aucun point n'est perdu. Nous pouvons même différencier les solutions exactes des solutions approchées.

b) Relâchement

Soit un espace de recherche S et un ensemble hiérarchisé C de contraintes. Soit π un ensemble de contraintes secondaires.

Soit un problème Pj avec S et $C \setminus \pi$. Soit Pk avec S et C.

Nous prouvons que la résolution de Pj est identique à la résolution de Pk en utilisant le mécanisme de relâchement sur l'ensemble π. Dans les deux problèmes, nous voulons trouver $S^{C \setminus \pi}$.

Il est logique que $S^C \subseteq S^{C \setminus \pi}$. Les points satisfaisant toutes les contraintes C sont inclus dans l'ensemble des points ne satisfaisant que les contraintes de $C \setminus \pi$.

Les solutions réalisables de Pk (S^{C_0}) sont les mêmes que celles de Pj. Les contraintes absolues dans C sont les mêmes que dans $C \setminus \pi$. Car π représente des contraintes secondaires.

Nous cherchons les dérivations en pseudo-échec dont au moins un ensemble de contraintes de relâchement est inclus dans π. Le relâchement de π à partir de Pk consiste à redémarrer ces dérivations.

Déroulons en parallèle la résolution de ces deux problèmes Pk et Pj. Dans une instruction sur une dérivation (G_i -> G_{i+1}), nous avons :

- dans Pk, les points éliminés dans G_{i+1} sont ceux de G_i qui ne satisfont pas les contraintes absolues. Il en est de même pour Pj,
- dans Pk, les points de G_{i+1} sont ceux qui satisfont les contraintes absolues. Toutes les contraintes secondaires (même π) sont testées sur tous ces points. Ces points sont donc gardés. C'est exactement ce qui est fait pour Pj.

Examinons les succès, les échecs et les pseudo-échecs.

• Soit la résolution du problème Pk. Un échec dans une dérivation signifie que des contraintes absolues ont éliminé tous les points. L'échec est le même pour Pj.

• Admettons qu'une dérivation dans Pk s'arrête sur un succès. Les points agréés sont inclus dans S^C. Cependant, il faut mémoriser les points rejetés dans un contexte de pseudo-échec. Ceci revient à mémoriser l'hypergraphe HG.

Dans le problème Pj, les points de S^C font partis de $S^{C\setminus\pi}$. La différence vient des points rejetés dans le pseudo-échec.

Examinons les pseudo-échecs. Pour ceux-ci, dans Pk, il y a plusieurs configurations :
- les points rejetés provoquant le pseudo-échec, satisfont π,
- ces points rejetés ne satisfont pas π et uniquement lui,
- ils ne satisfont pas π et d'autres contraintes γ.

① Si ces points satisfont π, le rejet n'est dû à π. D'autres contraintes de $C\setminus\pi$ insatisfont les points rejetés. Ces derniers sont aussi rejetés par Pj.

② Si ces points ne satisfont pas π et uniquement lui, π représente alors des contraintes de relâchements. Ainsi ces dérivations sont redémarrées sans π. Leur déroulement par la suite, va être le même que dans Pj.

③ Si ces points ne satisfont pas π et d'autres contraintes γ, π ne représente pas à lui seul, des contraintes de relâchement. Effectivement γ appartient à $C\setminus\pi$. Ces dérivations ne sont pas redémarrées. Les points concernés ne deviennent pas exactes. Ceci est aussi le cas pour Pj. Car les contraintes γ de $C\setminus\pi$ sont insatisfaites en ces points.

D'abord, nous résolvons Pk. Puis nous redémarrons les dérivations en pseudo-échec où il existe des contraintes de relâchement appartenant à π. Nous avons prouvé que la résolution de Pj équivaut à faire ces deux actions.

Il faut cependant s'assurer que les dérivations redémarrées se déroulent normalement. Des problèmes peuvent intervenir quand des contraintes P.L.C. redéclenchent des contraintes élémentaires relâchées. Ceci explique les hypothèses sur les algorithmes des méthodes locales de satisfaction des contraintes.

Ainsi il faut prendre plusieurs hypothèses afin que le relâchement soit sain :

- les points passés de G_i à G_{i+1} sont des solutions réalisables (S^{C_0}),

- l'hypothèse générale est respectée,
- les méthodes garantissent l'indépendance des contraintes élémentaires entre elles,
- Mémoriser le contexte, à chaque arrêt dû à une insatisfaction des contraintes C et dû à une satisfaction des contraintes absolues C_0.

Le langage à implanter doit respecter ces hypothèses.

c) Meilleures solutions approchées

Ce mécanisme s'applique sur des cas où la résolution donne un échec et crée des dérivations en pseudo-échec.

Une meilleure solution correspond à une solution approchée issue d'un meilleur relâchement. Rappelons le processus :

① Chercher le meilleur relâchement de contraintes élémentaires dans les dérivations en pseudo-arrêt.
② Continuer une des dérivations possibles en pseudo-arrêt avec le meilleur relâchement.
③ Si l'arrêt de cette dérivation est un échec ou un pseudo-échec, refaire le processus à partir de ①.
④ Si l'arrêt est un succès, arrêter le processus. Les solutions trouvées sont alors les meilleures.

En premier, si la résolution normale du programme ne boucle pas, ce processus se termine.

En second, rappelons que :

- une mesure ne peut que s'empirer au fur et à mesure du déroulement d'une dérivation,
- à une étape donnée, un point est considéré le meilleur de tous les points de toutes les dérivations en arrêt,
- si une dérivation redémarrée aboutit à un pseudo-échec, les contraintes relâchées sont remises en place. La mesure tient compte de toutes les insatisfactions même avec les contraintes relâchées avant.

Chaque relâchement n'est fait que si c'est le meilleur relâchement. Donc quand une étape aboutit à un succès, elle a des solutions. Soit ces solutions sont issues d'une succession de meilleurs relâchements. Ces solutions sont les "meilleures" (l'analyse est refaite à partir de zéro à chaque arrêt). Soit ces solutions satisfont toutes les contraintes initiales, elles sont considérées "d'office" les "meilleures".

D.5 Une implantation avec CLEF v1

Nous allons découvrir une implantation de ces mécanismes. Celle-ci est basée sur le langage CLEF v1. Le plan de cette division est identique à celui de la division D.4.

D.5.1 HCI et CLEF v1

a) Classement et hiérarchie de contraintes de CLEF v1

Nous avons déterminé deux classes de contraintes : absolues et secondaires.

Nous allons ajouter un prédicat spécifique. Il indique au système CLEF v1 que la contrainte passée en paramètre est secondaire. Un second paramètre fournit son niveau de hiérarchie.

Exemple : niveau (*C, *note).

Ce prédicat "niveau" déclare que la contrainte *C est secondaire et de niveau *note.

Toutes contraintes non déclarées sont considérées comme absolues.

Nous avons aussi distingué deux sortes de contraintes : élémentaire et compositionnelle.

En CLEF v1, les contraintes compositionnelles sont les contraintes P.L.C.. Nous prendrons pour contrainte élémentaire, la méthode locale à la contrainte P.L.C. s'associant à un déclenchement dû à une instanciation de variable. C'est donc l'ensemble des tests faits par la contrainte vis-à-vis de cette instanciation.

Exemple : ^diff est une contrainte compositionnelle. Elle donne naissance à au plus une contrainte élémentaire par dérivation.

^pas-plus-de-n est une contrainte compositionnelle. Elle peut avoir plusieurs déclenchements. Après un déclenchement, l'état de satisfaction n'est pas toujours connue. La contrainte reste présente dans le réseau. Celle-ci est composée d'autant de contraintes élémentaires que d'instanciations de variables impliquées.

b) Sommets de HCI ou valeurs "sûres" de CLEF v1

Les sommets de l'hypergraphe correspondent aux valeurs rejetées par les contraintes secondaires et agréées par les contraintes absolues.

Pour mémoriser les valeurs, nous ajoutons une structure évoluant tout au long de la résolution. Une telle structure est associée à chaque domaine de chaque variable. Si un backtrack a lieu, les domaines sont remis au bon contexte. Il en est de même pour la structure associée au domaine.

En CLEF v1, le domaine d'une variable est une liste de valeurs. A partir de maintenant, nous allons associer deux listes à chaque variable *vi sous domaine :

- une liste D_i des valeurs possibles (ou son domaine),
- une liste lvi de même cardinalité que D_i dont les éléments sont des listes de marques .

Une marque est formé par le symbole d'une contrainte P.L.C. et par le numéro de déclenchement élémentaire associé (lié à la variable).

Exemple : Supposons la variable *vi de domaine D_i = [1, 2, 3] dont la valeur 2 est marquée par une contrainte "^pas-plus-de-n" dans un premier déclenchement. Nous avons : D_i = [1, 2, 3] et lvi = [[],[(^pas-plus-de-n,1)],[]].

c) Hyperarêtes de HCI ou contraintes élémentaires secondaires

Si lors d'un déclenchement d'une contrainte secondaire, des valeurs sont rejetées, la raison est mémorisée dans la seconde liste lvi associée aux variables.

Dans notre exemple ci-dessus, la contrainte secondaire "^pas-plus-de-n" rejette la valeur 2 de *vi dans un déclenchement vis-à-vis de l'instanciation de la variable 1.

D.5.2 Construction d'un HCI

a) Hypothèses sur les méthodes locales de satisfaction

i) Hypothèse sur les contraintes

L'hypothèse stipule que tout déclenchement de contrainte CLEF v1 doit appliquer des contraintes élémentaires différentes et indépendantes. Or ceci n'est pas le cas pour toutes les contraintes.

L'algorithme de la méthode locale de satisfaction doit mémoriser les raisons de déclenchement (instanciation d'une variable) et ne pas refaire des tests.

Les contraintes "^pas-plus-de-n" et "^pas-moins-de-n" vérifient cette hypothèse. L'instance d'une variable *vi provoque les tests de cohérence uniquement avec cette variable. Une instance d'une autre variable *vj n'implique pas de refaire les tests faits avec *vi. L'algorithme garantit l'hypothèse avec la variable interne "lrang" (cf C.2.2).

ii) Hypothèse sur les ensembles d'agrément

C'est l'hypothèse générale présentée en (D.3.1 d). Si l'espace de recherche est un produit cartésien $S = D_1 \times ... \times D_n$ alors l'ensemble d'agrément d'une contrainte secondaire élémentaire c est aussi un produit cartésien : $S^c = D_1{}^c \times ... \times D_n{}^c$.

Les méthodes de satisfaction de toutes les contraintes CLEF v1 satisfont cette hypothèse et donc tous les ensembles d'agrément (cf B.4).

b) Les sommets de HCI

Les méthodes de satisfaction des contraintes CLEF v1 réalisent l'élimination des valeurs incohérentes. Ainsi elles éliminent les points issus de ces valeurs. Aucun changement n'est donc fait dans le système.

Cette fois, il faut mémoriser les valeurs incohérentes dues aux contraintes secondaires. La méthode locale est la même. Mais nous marquons les incohérences dans la structure spécifique attachée à chaque variable.

Par exemple dans l'algorithme de ^pas-plus-de-n, l'instruction est changée (cf

C.2.2). Nous avons $D_{Fu}{}^{\wedge pas\text{-}plus\text{-}de\text{-}n} = D_{Fu} \setminus \{w \in D_{Fu} \ / \ [Fi..Fi+d_i-1] \cap [w..w+d_u-1] \neq \emptyset\}$. Nous la transformons par :

Si contrainte secondaire alors
    $D_{Fu}{}^{\wedge pas\text{-}plus\text{-}de\text{-}n} = D_{Fu}$; marquer($\{w \in D_{Fu} \ / \ [Fi..Fi+d_i-1] \cap [w..w+d_u-1] \neq \emptyset\}$)
sinon $D_{Fu}{}^{\wedge pas\text{-}plus\text{-}de\text{-}n} = D_{Fu} \setminus \{w \in D_{Fu} \ / \ [Fi..Fi+d_i-1] \cap [w..w+d_u-1] \neq \emptyset\}$
Fin si

c) Les hyperarêtes contraintes à travers l'étape de dérivation

Les contraintes secondaires forment les hyperarêtes de notre HCI.

Les étapes de dérivation réalisent ce que nous voulons. Le mécanisme de résolution n'est pas changé.

Il suffit de prendre en compte les variables avec leur domaine et la nouvelle structure associée. Il faut aussi changer les méthodes de satisfaction pour les contraintes secondaires.

Seuls les cas d'arrêts changent. Nous allons les étudier.

d) Construction d'un HCI par les arrêts d'une dérivation

Suivant le cas d'arrêt, il faut mémoriser ou non la dernière étape de dérivation en arrêt. La différence entre ces arrêts se fait sur les ensembles $S^{\mathcal{C}}$ et $S^{\mathcal{C}}0$. En CLEF v1, ceci se traduit par la détection sur les domaines courants.

Si un domaine devient vide, cela signifie qu'il n'existe plus de solution réalisable ($S^{\mathcal{C}}0 = \emptyset$). Nous avons un échec. C'est le comportement normal de CLEF v1.

Si un domaine a toutes ces valeurs marquées ($D^{\mathcal{C}} = \emptyset$ et $D^{\mathcal{C}}0 \neq \emptyset$), nous avons un pseudo-échec. Il faut mémoriser la dérivation.

Supposons que la dérivation s'arrête. Si tous les domaines ont au moins une valeur non marquées, alors c'est un succès. Les solutions sont issues du produit cartésien de ces valeurs. Il faut quand même mémoriser cette dérivation pour les valeurs marquées.

☞ Abordons la manière de mémoriser une étape en CLEF v1.

Rappel : L'étape mémorisé est G_{i+1} = <Wi, :- t0, ..., tn., $\mathcal{C} \cup \{mc(xi), xi \in X\}$, HG (X, HA)>

L'hypergraphe est mémorisé à travers le domaine D_j et la structure lv_j de marques attachés à chaque variable $*v_j$. Un prédicat "domaine_contraint ($*v_j$, $*D_j$, $*lv_j$)" déclare les variables avec leur domaine et leur marque.

Nous mémorisons l'étape G_{i+1} par une clause. Le corps a les termes t0, ..., tn, les contraintes $\mathcal{C}$ du réseau et une série de prédicats "domaine_contraint" pour décrire l'état courant de toutes les variables de cette étape.

Nous paramétrons cette clause par la raison de l'échec. Les premiers paramètres sont constituées de toutes les variables dont le domaine est entièrement marqué. Puis nous mettons dans les autres paramètres leurs domaines et marques associées. Ceci permet,

par la suite, d'analyser la dérivation pour connaître les contraintes de relâchement en ne tenant compte que des variables d'en-tête.

Exemple : Soit la seule variable $*v_j$ de domaine D_j entièrement marqué. $*lv_j$ est la liste de marques associées à $*v_j$. Soit toutes les variables *w1, ..., *wn sans $*v_j$. Soit l'étape à mémoriser G_i = <Wi, :- t0, ..., tn., C, HG (X, HA)>. Nous avons :

mémo ($*v_j$, $*D_j$, $*lv_j$) :-
domaine_contraint ($*w_1$, $*D_1$, $*lw_1$),
...
domaine_contraint ($*w_n$, $*D_n$, $*lw_n$),
C, t0, ..., tn.

N.B. : Les "mc(xi)" sont présents dans la structure de chaque domaine de chaque variable. Nous allons le voir quand nous abordons la manière de relâcher des contraintes. Les contraintes ne sont pas réellement enlevées. Nous n'appliquons plus leur méthode de satisfaction.

D.5.4 Utilisation des HCI

a) Analyses d'insatisfaction dans une étape de dérivation

i) Détection d'insatisfaction d'une contrainte élémentaire

Pour détecter les points insatisfaisant une contrainte élémentaire, il suffit de lister le domaine et la structure des variables. Nous ne retenons que les variables et les valeurs marquées par cette contrainte.

ii) Contraintes à relâcher

Les pseudo-échecs sont mémorisés par des clauses. Pour détecter les contraintes de relâchement de chaque pseudo-échec, nous analysons l'entête de ces clauses.

Nous prenons chaque variable avec son domaine et sa liste de marques. Chaque ensemble de marques associées à chaque valeur constitue un élément pour les contraintes de relâchement.

Prenons un élément par variable. L'union de ces éléments constitue des contraintes de relâchement. L'ensemble de toutes les unions de telle forme, constitue l'ensemble des contraintes de relâchement.

b) Relâchement et "meilleures" solutions

i) Relâchement

Admettons qu'un utilisateur veuille relâcher un ensemble quelconque π de contraintes. D'abord nous examinons les dérivations en pseudo-arrêt. Si leurs contraintes de relâchement sont inclus dans π, nous redémarrons ces dérivations.

Redémarrer la dérivation consiste à appeler la clause correspondante ("mémo").

Cependant, il faut mémoriser les contraintes qui viennent d'être relâchées. Nous

indiquons au système CLEF v1, quelles sont les contraintes relâchées. Par exemple, ceci peut se faire à l'aide d'un prédicat spécifique "relâche(π)". Toute référence aux contraintes relâchées dans les marques et dans les méthodes locales est ignorée.

Supposons que le relâchement porte sur une contrainte compositionnelle. Pendant la résolution, nous n'appliquons pas la méthode associée à cette contrainte.

Supposons que le relâchement porte sur une contrainte élémentaire. Le système CLEF v1 doit détecter si l'espace de recherche est entièrement marqué. Il faut tester $D_i^{C\setminus\pi} = \emptyset$ et non plus $D_i^C = \emptyset$. Donc nous ignorons les marques relatives aux contraintes π.

Cependant, les marques associées aux contraintes relâchées π restent présentes. Si la dérivation redémarrée s'arrête plus tard en pseudo-échec, nous mémorisons toutes les marques. Par ce fait, nous inhibons le relâchement des contraintes. Ceci correspond aux "$\cup$ mc(xi)" lors de la mémorisation d'une étape (cf D.4.3).

Si la dérivation redémarrée s'arrête en succès, tous les points non marqués et ceux marqués par π sont des solutions.

ii) mesure sur les "solutions"

Abordons la manière de mesurer une dérivation sur un ensemble de points.

La mesure est une fonction. Elle prend en paramètre les marques (ou contraintes) et les niveaux de hiérarchie associés aux contraintes. De cette fonction, nous en déduisons une note pour les points composés avec les valeurs incohérentes.

Appliquons la fonction de mesure avec des contraintes de relâchement. Ceci permet de mesurer les points ne satisfaisant pas uniquement ces contraintes.

Dans une dérivation en pseudo-échec, les points à mesurer sont ceux issus de variables dont le domaine est entièrement marqué (dans notre clause "mémo", ces variables sont en en-tête). Nous n'analysons que les marques associées à ces variables.

iii) Meilleurs relâchements ≈ meilleures solutions

Le principe consiste à chercher le meilleur relâchement dans les dérivations en pseudo-échec. Dans ces dernières, nous relaxons des contraintes de relâchement impliquant la meilleure mesure.

Le procédé liste les clauses correspondant aux pseudo-échecs. Nous cherchons alors la meilleure mesure avec les marques, paramètres de ces clauses. Puis les relâchements sont effectués en indiquant les contraintes relâchées au système. Nous appelons ensuite la clause choisie. Puis nous la supprimons du programme (ce n'est plus un pseudo-échec).

Si une solution est trouvée, c'est la meilleure. Si la clause appelée aboutit à un échec, alors le procédé recommence.

D.6 Un exemple d'utilisation

Nous allons présenter sur un petit exemple de problème d'emploi du temps, l'utilisation de ces mécanismes de relâchement et de recherche de "meilleures" solutions.

D.6.1 Présentation de l'exemple

Prenons trois cours. Leur heure de début à trouver est F1, F2 et F3. Respectivement, leur durée est d1, d2, d3 de 2 unités. Nous avons un ensemble de 4 périodes de début numérotées de 1 à 4. Le but est de trouver le début de chaque cours avec l'ensemble hiérarchisé de contraintes suivant :

☞ Contraintes absolues : $\forall$ Fi, Fi $\geq$ 1 et $\forall$ Fi,. Fi + (di - 1) $\leq$ 4 et F1 $\geq$ F2 + 2 (donc 7 contraintes absolues)

☞ Contraintes secondaires : les cours ne se chevauchent pas dans le temps.

$$[F1..F1+d1-1] \cap [F2..F2+d2-1] = \emptyset,$$
$$[F1..F1+d1-1] \cap [F3..F3+d3-1] = \emptyset,$$
$$[F3..F3+d3-1] \cap [F2..F2+d2-1] = \emptyset.$$

Soit les variables horaires $*v_{HFi}$ des cours Fi. Nous utilisons les contraintes CLEF v1, ^≥ et ^≤ comme contraintes absolues et ^pas-plus-de-n comme contraintes secondaires.

Soit les prédicats spécifiques suivants :

- **niveau** (*c, *note) pour spécifier que *c est une contrainte secondaire de niveau *note.
- **relâche** ($*\pi$) indique au système que π est un ensemble de contraintes relâchées.
- **domaine_contraint** ($*v_i$, $*D_i$, $*lv_i$) signifie que la variable $*v_i$ a comme domaine $*D_i$ et la structure de marques $*lv_i$.

Le programme est le suivant :

```
Exemple :-
        domaine (*vHF1, [1, 2, 3, 4]),
        domaine (*vHF2, [1, 2, 3, 4]),
        domaine (*vHF3, [1, 2, 3, 4]),
        ^≤ ((+ *vHF1 1), 4), ^≤ ((+ *vHF2 1), 4), ^≤ ((+ *vHF3 1), 4),
        ^≥ (*vHF1, 1), ^≥ (*vHF2, 1), ^≥ (*vHF3, 1),
        ^≥ (*vHF1, (+ *vHF2 2)),
        niveau (^pas-plus-de-n (1, [*vHF1, *vHF2, *vHF3], [2, 2, 2],
                                   [1, 1, 1]]), 8),
        d-valuer (*vHF1),
        d-valuer (*vHF2),
        d-valuer (*vHF3).
```

D.6.2 Un cas d'insatisfaction

Arrêtons un instant la résolution après le prédicat "d-valuer ($^*v_{HF1}$)". Supposons que seule la contrainte secondaire ait été déclenchée. Examinons les domaines des variables avec leur hypergraphe (ou marques).

Par commodité, la raison des insatisfactions est indiquée par le symbole de la contrainte P.L.C. et le numéro de déclenchement élémentaire. Sachant qu'une variable n'a qu'une instanciation dans une dérivation, ce numéro est 1 pour une instantiation de F1, 2 pour F2 et 3 pour F3.

Schématiquement, nous avons le contexte suivant :

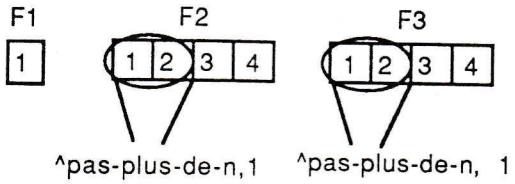


Figure 31. Insatisfaction d'une contrainte

Nous voyons que F1 est placé en 1. F2 ne peut se placer ni en 1, ni en 2. La contrainte ^pas-plus-de-n à son premier déclenchement les rejette. Mais F2 peut se placer en 3 et en 4. Il en est de même pour F3.

D.6.3 Résolution et arrêts

Cette fois, nous allons dérouler la résolution. Nous décrivons l'état des variables après chaque instanciation. Un cas constitue une instanciation d'une variable à une valeur cohérente de son domaine.

La valeur 4 de toutes les variables est éliminée par les contraintes absolues : $^{\wedge}\leq$ ((+ $^*v_{HF1}$ 1), 4), $^{\wedge}\leq$ ((+ $^*v_{HF2}$ 1), 4), $^{\wedge}\leq$ ((+ $^*v_{HF3}$ 1), 4).

☞ Cas 1 : placement de F1 à 1.

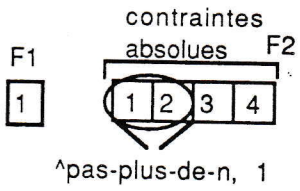


Figure 32. Placement de F1 à 1

Toutes les valeurs de F2 sont éliminées par des contraintes absolues. La contrainte $^{\wedge}\leq$ ((+ F2 1), 4) élimine la valeur 4 et $^{\wedge}\geq$ ($^*v_{HF1}$, (+ $^*v_{HF2}$ 2)) élimine les valeurs de 1 à 4.

Ainsi $D^{C}0_{HF2} = \emptyset$. C'est un échec. Aucune clause n'est mémorisée.

☞ Cas 2 : placement de F1 à 2.

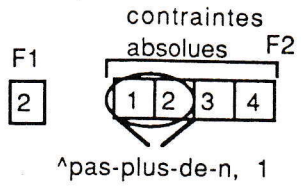


Figure 33. Placement de F1 à 2

De même que le cas précédent, F2 n'a aucune possibilité. Toutes ses valeurs sont éliminées par des contraintes absolues.

☞ Cas 3 : placement de F1 à 3.

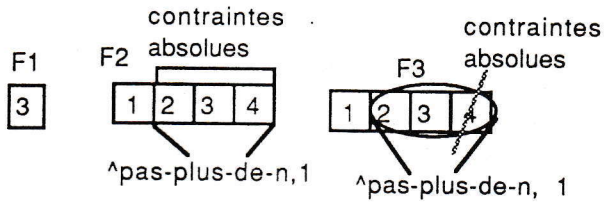


Figure 34. Placement de F1 à 3

Après cette instanciation, aucun échec n'est détecté. F1 est placé en 3, F2 a la seule possibilité 1. F3 a le domaine réduit {1, 2, 3}. Les valeurs 2 et 3 sont marquées par (^pas-plus-de-n, 1).

☞ Sous-Cas : placement de F2 à 1.

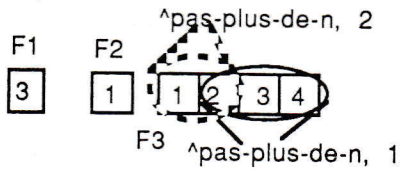


Figure 35. Placement de F2 à 1

Nous nous apercevons que le domaine de F3 (D_{HF3}) devient entièrement contraint.

Cependant, les valeurs de ce domaine ne sont pas éliminées. Elles satisfont les contraintes absolues déclenchées. Ce cas correspond donc à un pseudo-échec. Une clause "mémo" est donc générée :

```
mémo (*vHF3, [1, 2, 3],
            [   [^pas-plus-de-n, 2],
                [[^pas-plus-de-n, 1], [^pas-plus-de-n, 2]],
                [[^pas-plus-de-n, 1]
            ]) :-
    domaine_contraint (*vHF1, [3], []),
    domaine_contraint (*vHF2, [1], []),
```

```
domaine_contraint (*v_HF3, [1, 2, 3],
                        [ [^pas-plus-de-n, 2],
                          [[^pas-plus-de-n, 1], [^pas-plus-de-n, 2]],
                          [[^pas-plus-de-n, 1]
                        ]),
niveau (^pas-plus-de-n (1, [*v_HF1, *v_HF2, *v_HF3], [2, 2, 2], [1, 1, 1]]),
        8),
d-valuer(*v_HF3).
```

La valeur 4 pour F1 insatisfait la contrainte absolue ^≤ ((+ $*v_{HF1}$ 1), 4). La résolution donc termine par un échec avec une clause mémorisée.

D.6.4 Les solutions approchées

Nous voulons connaître les solutions approchées dans cette résolution. Il faut relâcher des contraintes dans la seule clause mémorisée.

Relâcher (^pas-plus-de-n, 1) donne la solution <3, 1, 3> pour F1, F2 et F3.
Relâcher (^pas-plus-de-n, 2) donne la solution <3, 1, 1> pour F1, F2 et F3.
Relâcher (^pas-plus-de-n, 2) et (^pas-plus-de-n, 1) fournit les solutions {<3, 1, 1>, <3, 1, 2>, <3, 1, 3>} pour F1, F2 et F3.

Rappelons que relâcher une contrainte comme (^pas-plus-de-n, 1) appelle la clause "relâche ([^pas-plus-de-n, 1])" puis la clause "mémo" (en la supprimant). Le test de cohérence devient $D^{C\setminus[\text{^pas-plus-de-n, 1}]} = \emptyset$.

D.6.5 Meilleures solutions suivant différentes mesures

Rechercher la meilleure solution consiste à chercher le meilleur relâchement entre {(^pas-plus-de-n, 2)}, {(^pas-plus-de-n, 1)} et {(^pas-plus-de-n, 2), (^pas-plus-de-n, 1)}.

Pour faire ce choix, il faut disposer de mesures et d'une fonction de comparaison.

• Prenons la mesure qui calcule le nombre de contraintes insatisfaites. Associons à la meilleure mesure, une fonction de comparaison. Celle-ci cherche le plus petit nombre de contraintes insatisfaites.

Les meilleures solutions approchées correspondent au relâchement de (^pas-plus-de-n, 1) ou (^pas-plus-de-n, 2) soit <3, 1, 1> et <3, 1, 3>.

• Supposons qu'un utilisateur attribue maintenant un poids à chaque contrainte. Le poids 1 est donné à (^pas-plus-de-n, 1) et le poids 2 à (^pas-plus-de-n, 2).

La mesure fournit le poids maximal des contraintes insatisfaites en chaque point.

Sur le point <3, 1, 1>, il faut relâcher (^pas-plus-de-n, 2). La mesure est 2.

Sur le point <3, 1, 2>, il faut relâcher (^pas-plus-de-n, 2) et (^pas-plus-de-n, 1. La mesure est Maximum(1, 2) = 2.

Sur le point <3, 1, 3>, il faut relâcher (^pas-plus-de-n, 1). La mesure est 1.

- La fonction de meilleure solution est celle qui fournit la mesure minimum. La meilleure solution est donc <3, 1, 3>. Il faut relâcher (^pas-plus-de-n, 1).

Ces deux mesures et fonctions de comparaison ne sont que des exemples. Cependant, l'utilisateur peut choisir ses points (donc ses relâchements).

D.7 Critiques

Les mécanismes ainsi proposés permettent de relâcher et de chercher des meilleures solutions d'une façon sûre et saine.

Ces mécanismes ont évités les "points épineux" présentés dans à la section D.2.3 :

- dans une dérivation, un point n'est pas éliminé par une contrainte secondaire. Ce point est donc testé par les autres contraintes. Supposons une contrainte secondaire relâchée en ce point. Ceci n'entraine pas, à nouveau, une application des contraintes déjà déclenchées.

- dans la résolution, nous mémorisons les dérivations en arrêt. Si le relâchement de contraintes libère des points, alors ces dérivations sont redémarrées.

- le relâchement d'une contrainte peut se faire sur une partie de ces effets et donc sur une partie des points qu'elle rejette.

- les contraintes P.L.C. restantes ne testent pas deux fois les mêmes points pour les mêmes raisons dans une même dérivation.

- A chaque arrêt d'une dérivation, nous connaissons les contraintes secondaires réduisant totalement l'espace de recherche. Cet espace de recherche satisfait toutes les contraintes absolues.

- A chaque arrêt, l'ensemble des contraintes insatisfaites est mémorisé, y compris les contraintes relâchées.

- Le relâchement ne change pas la complexité de la résolution dans sa totalité.

Cette proposition a été simulé en CLEF v1. Le temps d'exécution est changé et est fonction du temps de la mémorisation. Nous avons simulé ces mécanismes uniquement pour tester la fonctionnalité. Nous ne nous sommes pas intéressés au temps de mémorisation.

Un autre inconvénient provient du nombre de clauses (donc d'étapes) caractérisant les pseudo-échecs. Il serait intéressant d'utiliser un S.G.B.D. pour mieux les stocker et bien les gérer.

Nous nous apercevons que les fonctionnalités proposées permettent d'aider un concepteur d'emploi du temps. Celui-ci peut relâcher certaines contraintes sur le cours qu'il désire.

De plus, il connaît les raisons des impossibilités de placement pour chacun des cours. Il ne travaille plus en aveugle.

D.8 Conclusion

Dans ce chapitre, nous nous sommes aperçus que les résolutions existantes pour le relâchement de contraintes et la recherche de meilleures solutions ne nous conviennent pas. Si nous prenons un problème d'emploi du temps qui n'admet aucune solution, nous sommes aveugles. Nous ne savons pas quelles sont les contraintes à relâcher.

C'est pourquoi j'ai introduit une proposition pour détecter les contraintes à relâcher et pour obtenir sûrement les "meilleures" solutions.

Cette proposition s'applique à des langages intégrant des domaines discrets et finis, intégrant une hiérarchie de contraintes à deux niveaux et respectant des hypothèses précises.

Certains langages existants de P.L.C. intégrent déjà la notion de domaine fini discret. Ils respectent à priori certaines hypothèses. Avec l'exemple d'implantation des mécanismes de relâchement et de "meilleures" solutions dans CLEF V1, nous constatons que ces langages peuvent aisément avoir accès à ces mécanismes. En résumé, nous leur ajoutons les notions suivantes :

- une hiérarchie de contraintes (par exemple, déclarer un poids aux contraintes),
- une construction d'HCI issus des contraintes secondaires (par exemple, marquer les valeurs incohérentes sans les rejeter),
- une fonctionnalité de relâchement (par exemple, démarquer les valeurs en cause et inhiber la méthode des contraintes relâchées),
- une mémorisation d'un contexte d'arrêt (par exemple, par une clause),
- une mesure liée aux contraintes d'un contexte (par exemple, une fonction portant sur le poids de leur niveau de hiérarchie),
- un comparateur de mesures.

Nous sommes alors capables de trouver des solutions avec des relâchements. En associant des mesures pour comparer deux solutions, nous pouvons trouver la "meilleure" solution. Là encore des hypothèses sont faites sur ces mesures (monotonie).

Un premier travail futur est de proposer un jeu de mesures pour différents types de problèmes qui sont souvent rencontrés dans la vie réelle.

Un deuxième travail futur est d'étendre la proposition à des cas où les langages intégrent des domaines infinis et continus.

Mais d'ores et déjà, grâce à ces mécanismes, nous pouvons proposer des solutions satisfaisant un utilisateur particulier. Ce dernier est alors à même de trouver sa "bonne" solution pour son problème précis.

Conclusion

Dans le chapitre C, nous avons proposé un modèle d'application de la P.L.C. au problème d'emploi du temps. Cette proposition se base sur une coloration multi-dimensionnelle. Nous avons implanté ce modèle sur un exemple illustratif. Un maquettage, suite de cette implantation, nous a permis de conclure que la P.L.C. résoud bien les problèmes d'emploi du temps.

Malgré tout, la P.L.C. a des inconvénients. Elle ne fait qu'indiquer l'existence de solutions satisfaisant toutes les contraintes en place. Pour ces problèmes d'emploi du temps trop contraints, la P.L.C. ne permet pas d'indiquer des solutions approchées au problème posé par un utilisateur.

C'est pourquoi nous avons proposé dans le chapitre D, un mécanisme de relâchement de contraintes et de recherche de "meilleures" solutions. Cette proposition se restreint au cas où les langages de P.L.C. intégrent des domaines discrets et finis avec une hiérachie de contraintes.

Grâce à cela, nous pouvons conclure que les problèmes d'emploi du temps sont bien résolus par la P.L.C. même dans le cas où le problème posé est trop contraint.

Effectivement, un concepteur d'emplois du temps peut mieux formaliser son problème en introduisant des nuances dans l'importance des contraintes. Il peut donc "surcontraindre" son problème. La solution obtenue lui fournira un emploi du temps réalisable mais ne respectant pas forcément toutes les contraintes en place.

Conclusion générale

Tout ce travail de thèse montre que divers "points noirs" concernant les problèmes d'emploi du temps sont abordés favorablement en utilisant un langage de type Programmation en Logique avec Contraintes (diversité des données, aspect combinatoire, aspect fortement contraint, imprécision des critères, imprévisibilité des changements).

L'apport (backtrack, possibilité de définir des heuristiques ou des stratégies, critère implicite, contraintes symboliques et numériques, retardement automatique d'évaluation, ...) de l'utilisation de la Programmation en Logique avec Contraintes permet d'alléger la résolution et surtout de donner une solution aux "points noirs" du problème d'emploi du temps.

Le but de mon travail a été de proposer une méthodologie se basant sur une nouvelle modélisation des problèmes d'emploi du temps et de l'appliquer avec la P.L.C. sur un exemple complet. Cette méthodologie comprend ma proposition sur les mécanismes de relâchement de contraintes et de recherche de "meilleures" solutions en P.L.C..

Comparativement aux résolutions existantes présentées au premier chapitre, la modélisation que je propose, permet de mieux représenter les problèmes d'emploi du temps. La prise en compte d'un problème est plus précise et l'ensemble de problèmes représentés est plus général. L'application avec la P.L.C. montre que les problèmes sont résolus pius efficacement.

De plus, un concepteur d'emplois du temps peut bien formaliser son problème en introduisant des nuances dans l'importance de ses contraintes. La solution obtenue lui fournit un emploi du temps le satisfaisant mais ne respectant pas forcément toutes les contraintes en place.

D'un autre coté, dans notre application, nous proposons un jeu de contraintes pour des problèmes d'emploi du temps les plus couramment rencontrés. Mais nous ne sommes pas à l'abri d'un problème où aucune des contraintes présentes n'est adéquate.

C'est pourquoi, il est intéressant d'aller plus loin en proposant un jeu de propriétés de base. Ces propriétés seraient utilisées par tous les problèmes d'emploi du temps. Ceci impliquerait un ensemble de contraintes de base.

N.B. : *Nous pouvons déjà remarquer que les deux nouvelles contraintes introduites*

permettent de représenter un nombre important de propriétés.

Deuxièmement, nous avons proposé un mécanisme de recherche de "meilleures" solutions suivant une mesure donnée. Il faudrait recenser un jeu de mesures pour proposer une mesure adaptée suivant la classe du problème voulu.

Troisièmement, dans notre cas, nous résolvons le "glissement" de cours par le mécanisme de backtrack. Nous nous doutons qu'il serait utile d'intégrer un mécanisme moins onéreux en terme de backtrack (donc de retours arrières).

Bien entendu, ces trois futures études ne représentent que des grandes lignes de travaux. Tout au long de ce document, des affinements aux propositions peuvent être apportés (par exemple, nous pourrions étendre les HCI pour des langages intégrant des domaines infinis et continus ou améliorer la propriété B de la multi-coloration).

Références

[AKK 73] **E.A. Akkoyunlu**, *A linear algorithm for computing the optimum university timetable*, Computer J. Vol. 16 No. 4 (1973) 347-350.

[AUB 86] **J. Aubin and J. Ferland**, *A large scale timetabling problem*, Publication No. 568, Université de Montréal (Mai 1986).

[AUS 76] **RJ. Aust**, *An improvement algorithm for school timetabling*, Computer J. Vol. 19 No. 4 (1976) 339-343.

[BAR 78] **A.M. Barham and J.B. Westwood**, *A simple heuristic to facilitate course timetabling*, J. operat. res. soc. 29 No. 11 (1978) 1055-1060.

[BER 63] **C. Berge**, *Théorie des graphes et ses applications*, Dunod, Paris (1963).

[BER 73] **C. Berge**, *Graphes et Hypergraphes,* Dunod, Paris (1973).

[BER 87] **C. Berge**, *Hypergraphes, Combinatoire des ensembles finis,* Bordas, Paris (1987).

[BOR 89] **A. Borning and M. Maher and A. Martindale and M. Wilson**, *Constraint Hierarchies and Logic Programming*, proceedings of the sixth international conference, (Lisbonne 1989).

[CAR 86] **M.W. Carter**, *a survey of practical applications of examination timetabling algorithms*, Operations research 34 No. 2 (1986) 193-202.

[CHA 89] **N. Chahal and D. de Werra**, *An interactive system for constructing timetables on a PC*, European J. of operational research 40 (1989) 32-37.

[COL 83] **A. Colmerauer et H. Kanoui et M. Van Caneghem**, *PROLOG, Bases théoriques et Développements actuels,* TSI 2, No. 4 (1983) 271-311.

[COL 86] **A. Colmerauer**, *Notes sur Prolog III*, Séminaire en Programmation en Logique (1986) 159-173.

[COL 87] **A. Colmerauer**, *Opening the Prolog-III Universe*, BYTE Magazine, 12 (9), (1987).

[COR 88] **J.M. Cornilly**, *Couplage prolog/bases de données relationnelles : un modèle asynchrone*, Thèse de l'Université de Rennes I, France (1988).

[COU 89] **X. Cousin**, *Emploi du Temps : Problème mathématique ou Problème pour la Programmation en Logique avec Contraintes?*, [NT/LAA/SLC/313, CNET, (Lannion, 1989)] et [Rapport interne 494, IRISA, (Rennes 1989)].

[COU 90] **X. Cousin**, *Relâchement de contraintes en Programmation en Logique avec Contraintes,* Séminaire de Programmation en Logique de Trégastel (Mai 1990).

[COU 91] **X. Cousin**, *Meilleures solutions en Programmation en Logique avec Contraintes,* Onzième Conférence Internationale de Système Expert et leur Application, Conférence Générale, Vol. 1, (Avignon 91) (Juin 1991), 247-260.

[DEF 78], **A. Defrenne**, *The Timetabling Problem : A Survey.* Cahier Centre et Recherche Opérationnelle 20, No. 2, (1978) 163-169.

[DEG 81] **O.B. De Gans**, *A computer timetabling system for secondary schools in the netherlands*, Europ. j. oper. res. 7 No. 2 (1981) 175-182.

[DES 71] **J.F. Desnos**, *Elaboration automatique d'emplois du temps*, Thèse de l'Université scientifique et médicale de Grenoble (1971).

[DEW 75] **D. de Werra**, *On a Particular Conference Scheduling Problem.* Infor 13, No. 3 (1975) 308-315.

[DEW 8_] **D. de Werra**, *Graphs, Hypergraphs and Timetabling,* Methods of Operations Research 49, (198_) 201-213.

[DEW 85a] **D. de Werra**, *An introduction to timetabling*, European j. of operational research 19 (1985) 151-162.

[DEW 85b] **D. de Werra**, *Some Uses of Hypergraphes in Timetabling,* Asia-Pacific J. of Oper. Res. 2 (1985) 2-12.

[DIN 84] **M. Dincbas et JP. Lepape**, *Metacontrol of Logic Programming in METALOG*, Proc. Int. Conf. FGCS (Tokyo, 1984) 361-370.

[DIN 86] **M. Dincbas**, *Constraints, Logic Programming and Deductive Databases,* France-Japan Artificial Intelligence and Computer Science Symposium 86, (Tokyo 1986).

[DIN 87a] **M. Dincbas and H. Simonis and P. Van Hentenryck**, *Extending equation solving and constraint handling in logic programming*, Colloquium on resolution of equations in algebraic structures, (MCC, Texas, 1987).

[DIN 87b] **M. Dincbas and H. Simonis and P. Van Hentenryck**, *Solving large combinatorial problems in Logic Programming*, Technical report TR-LP-21 (ECRC, 1987).

[DIN 88a] **M. Dincbas and P. Van Hentenryck and H. Simonis and A. Aggoun and T. Graf,** *Applications of CHIP to industrial and engineering problems,* First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (Tullahoma, Tenessee, USA, June 1988).

[DIN 88b] **M. Dincbas and H. Simonis and P. Van Hentenryck,** *Solving the car-sequencing problem in Constraint Logic Programming,* ECAI (Munich August 1988).

[DIN 88c] **M. Dincbas and H. Simonis and P. Van Hentenryck,** *Solving a cutting-stock problem in Constraint Logic Programming,* International Conference on Logic Programming, (Seattle August 1988).

[DIN 88d] **M. Dincbas and P. Van Hentenryck and H. Simonis and A. Aggoun and T. Graf and F. Berthier,** *The Constraint Logic Programming Language CHIP,* International conference on fifth generation computer systems, ICOT (1988).

[EME 87] **T. Emery and P. Trigano and J.P Barthes,** *Sedlex : extension d'une base de données vers un sytème de représentation des connaissances : application à la planification,* Université de technologie de Compiègne (1987).

[FEL 89] **R. Feldman and M.C. Golumbic,** *Constraint satisfiability algorithms for interactive student scheduling,* Proc. Eleventh Int'l Joint Conf. on Artificial Intelligence (IJCAI-89), (August 1989) 1010-1016.

[FEL 90] **R. Feldman and M.C. Golumbic,** *Optimization Algorithms for Student Scheduling via Constraint Satisfiability,* The Computer Journal 33, No. 4 (1990) 356-364.

[FER 85] **J. Ferland and S. Roy and Tran Gia Loc,** *The timetabling Problem.* Publication 531, Université de Montréal, (Québec 1985).

[FER 86] **J. Ferland et G. Babin et J. Aubin,** *Système de confection d'horaires de cours.* Publication 582, Université de Montréal, (Québec 1986).

[GON 85] **M. Gondran and M. Minoux,** *Graphes et algorithmes* (Eyrolles, Paris, 1985).

[HAN 88] **C. Hanachi,** *Constitution de l'emploi du temps par une approche couplant système expert et SGBD,* Les bases de données : etat de l'art et nouvelles perspectives, JISI'88 Receuil des communications (1988)

[HER 86] **D. Hermann,** *Bases de données, Cours C56,* Université de Rennes I (1986).

[HOL 64] **A.G. Holzman and W.R. Turkes,** *Optimal Scheduling in Educational Institutions Cooperative Research Project 1323.* University of Pittsburgh, (Pennsylvania 1964).

[HOU 81] **J. Houdebine,** *LOGIQUE, Cours C28,* Université de Rennes I (1981).

[JAF 86] **J. Jaffar and S. Michaylov,** *Methodology and implementation of a CLP*

system, IBM-Yoktown Heights and Monash University (1986).

[JAF 87] **J. Jaffar and J.L. Lassez**, *Constraint Logic Programming*, IBM-Yorktown Heights (1987).

[JAF 88] **J. Jaffar and J.L. Lassez**, *From unification to constraint*, IBM thomas J. Watson research center (1988).

[LAM 85] **O. Lambert and J. Nomura and J. VanDruten and M. Ahlstrom**, *An Intelligent Rule-Based Scheduling System for the Office Environment*, (CSC 1985).

[LAP 84] **G. Laporte and S. Desroches**, *Examination timetabling by computer*, Computers and Oper. Res. 11 No. 4 (1984) 351-360.

[LAU 71] **J.L. Laurière**, *Sur la coloration de certains hypergraphes*, Thèse de Doctorat de 3ème cycle, Université de Paris VI (1971).

[LAU 74a] **J.L. Laurière**, *Problèmes d'emploi du temps et algorithme de coloration des hypergraphes*, Compte rendu de l'académie des sciences 278 No 18 série A (1974) 1159-1162.

[LAU 74b] **J.L. Laurière**, *Problèmes combinatoires d'emploi du temps*, Personnel et carrières III, Aide à la décision AFCET, No 1 (1974) 197-215.

[LAU 78] **J.L. Laurière**, *A language and a program for stating and solving combinatorial problems*, Artificial intelligence 10 (1978) 29-127.

[LEP 90] **JP. Lepape et D. Ranson**, *La Programmation en Logique avec Contraintes ; Le système CLEF version 1*, NT/LAA/SLC/315, CNET, (Lannion, 1990).

[LEK 90] **Le Kang**, *A Logic Approach to Conflict Resolution in University Timetabling*. Thesis on Partial Fulfillment of the Requirement for the Degree of Master, University of Ottawa (January 1990).

[LIO 66] **J. Lions**, *A Counter-example for Gotlieb's Method for the Construction of School Timetables.* C. A. C. M. 9, No.9 (September 1966) 697-698.

[MAH 89] **M. Maher and P. Stuckey**, *Expanding query power in Constraint Logic Programming Languages*, Proceedings of the NACLP, Cleverland, (USA, 1989).

[MAC 77] **A-K. Mackworth**, *Consistency in Networks of Relations*, Artificial Intelligence 8 (1977) 99-118.

[MON 74] **U. Montanari**, *Networks of constraints : fundamental properties and applications to picture processing*, Information Sciences 7 (1974) 95-132.

[NAI 83] **L. Naish**, *Automatic Generation of Control for Logic Programs*, Technical Report 83/6, Dept. of Computer Science, University of Melbourne (1983).

[PLA 74] **J.M. Pla**, *Problèmes Combinatoires d'Emploi du Temps*, Personnel et Carrières III. Aide à la Décision Congrès AFCET, No. 1 (1974) 215-227.

[POR 88] **M.C. Portmann**, *Polycopie de présentation des problèmes*

d'ordonnancement et des différentes approches de résolution. Thèse d'état, Université de Nancy (1988).

[RIT 88] **J.F. Rit,** *Modélisation et propagation de contraintes temporelles pour la planification*, Thèse de Docteur-Ingénieur, Université de Grenoble (1988).

[ROY 88] **B. Roy and D. Bouyssou,** *Aide à la décision*, AFCET/INTERFACES No. 65 (1988) 4-13.

[SCH 80] **G. Schmidt and T. Stroehlein,** *Timetable Construction - An Annotated Bibliography*, Comput. J. 23 No.4 (1980) 307-316.

[SCH 87] **J.L. Scharbarg,** *Conception d'emploi du temps pour les écoles*, Stage de DESS, Université de Rennes I (1987).

[SMI 76] **LD. Smith,** *The Application of an Interactive Algoritm to Develop Cyclical Rotational Schedules for Nursing Personnel,* Infor 14, No. 1 (1976) 53-70.

[SMI 79] **LD. Smith and A. Wiggins and D. Bird,** *Post-Implementation Experience with Computer-Assisted Nurse Scheduling in a Large Hospital.* Infor 17, No. 4 (1979) 309-321.

[STE 81] **M. Stefik,** *Planning with constraints. (MOLGEN : Part 1) & Planning and Meta-Planning. (MOLGEN : Part 2)*, Artificial Intelligence 16 (1981) 111-170.

[SUS 80] **G.L. Sussman and G.L. Steele,** *Constraints-A language for expressing almost-hierarchical descriptions*, Artificial Intelligence 14 (1980) 1-39.

[TRI 80] **A. Tripathy,** *A Lagrangean Relaxation Approach to Course Timetabling*, J. Operat. Res. Soc. 31, No. 7 (1980) 599-603.

[TRI 84] **A. Tripathy,** *School Timetabling-A Case in Large Binary Integer Linear Programming.* Management Science 30 (1984) 1473-1489.

[VAL 84] **M. Valtorta and B. Smith and D. Loveland,** *The Graduate Course Advisor : A Multi-Phase Rule-Based Expert System.* IEEE (1984).

[VAN 84] **M. Van Caneghem,** *L'anatomie de PROLOG II, Thèse de Doctorat d'état*, Aix-Marseille II, (1984).

[VAN 89] **P. Van Hentenryck,** *Constraint satisfaction in Logic Programming* (MIT press, Cambridge, 1989).

[VAN 90] **P. Van Hentenryck,** *Incremental Constraint Satisfaction in Logic Programming*, ICLP'90, (1989).

[WHI 75] **DJ. White,** *a note on faculty timetabling*, Oper. res. Quart. Vol. 26 No. 4 Partie 2 (1975) 875-878.